DevTest Solutions

Agents

Version 8.0

Support technique

Pour une assistance technique en ligne et une liste complète des sites, horaires d'ouverture et numéros de téléphone, contactez le support technique à l'adresse http://www.ca.com/worldwide.

Table des matières

Chapitre 1: Agent Java pour DevTest

L'agent Java pour DevTest est une technologie de serveur que vous pouvez installer dans un processus Java, y compris dans les conteneurs Java EE. L'agent permet à DevTest de contrôler et de surveiller les activités côté serveur.

L'agent peut effectuer les mêmes actions réalisées par la plupart des générateurs de profils : surveiller des classes/objets chargés, l'utilisation de l'UC, l'utilisation de la mémoire, des threads, suivre des appels de méthode, etc. Toutefois, il fonctionne sur plusieurs machines virtuelles Java et avec DevTest pour utiliser ses fonctionnalités uniques lors des tests.

L'agent rend notamment visible les actions effectuées par les serveurs en arrière-plan et entraînées par un test ou une étape de test. Cette fonctionnalité permet d'identifier des bogues et des goulots d'étranglement. Cette fonctionnalité est similaire aux fonctions de CA Continuous Application Insight. Toutefois, l'agent fonctionne sur tous les protocoles utilisés par les applications Java sans qu'il soit nécessaire d'instrumenter un code ni des fichiers de configuration.

L'agent prend également en charge CA Service Virtualization. L'agent active l'enregistrement et la lecture du trafic et des appels de méthode sur les protocoles. L'agent octroie à CA Service Virtualization le contrôle complet des zones de l'application cible que vous voulez virtualiser. L'agent offre une structure unifiée pour exécuter cette fonctionnalité quel que soit le protocole.

Ce chapitre traite des sujets suivants :

Architecture de l'agent Java

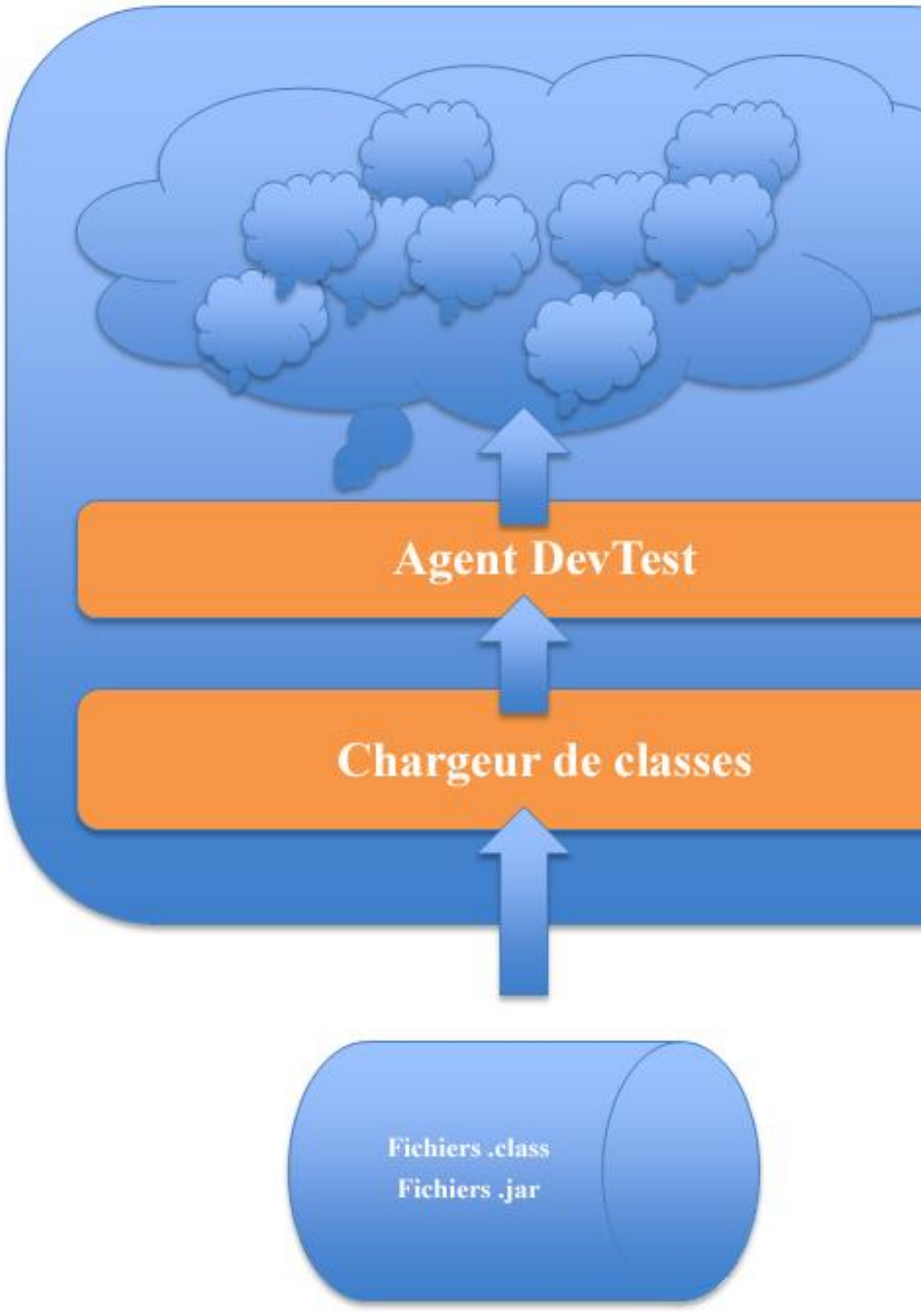
Cette section comprend les rubriques suivantes :

Technologie de l'agent JAVA

L'agent Java pour DevTest est un type d'agent Java.

Les agents Java sont des programmes intégrés dans une machine virtuelle Java. Ces programmes peuvent être conçus de manière à prendre en charge de nombreuses fonctions, telles que la collecte d'informations sur une application en cours d'exécution ou la virtualisation de parties d'une application.

Dans le graphique suivant, le grand conteneur représente la machine virtuelle Java. La machine virtuelle Java inclut un agent Java et le chargeur de classes, qui charge des fichiers de classe Java lors de l'exécution. Les nuages représentent des ressources chargées par l'application Java.


Agent DevTest
Chargeur de classes
Fichiers .class
Fichiers .jar

Un agent est mis en package dans un fichier .jar. La machine virtuelle Java doit être configurée avec le chemin d'accès au fichier .jar. A partir de Java 1.5, vous devez utiliser une chaîne commençant par **-javaagent** pour spécifier le chemin d'accès et les options de votre choix. Par exemple :

```
-javaagent:C:\myagent.jar=option1=true,option2=false
```

Une machine virtuelle Java unique peut inclure plusieurs agents.

Composants de l'agent Java

Le client Java pour DevTest comprend les composants suivants :

- L'agent même, exécuté de manière intégrée dans un processus Java
- L'intermédiaire
- Les consoles (ou clients)
- La base de données

L'intermédiaire est un concentrateur qui envoie des messages JMS entre les agents, les consoles et le client intégré.

Le client intégré effectue le suivi des propriétés de l'ensemble du réseau/de fractionnement d'agents, telles que l'ensemble d'agents, leurs files d'attente ouvertes, ou les connexions au réseau actuelles. En tant que tel, il peut regrouper des données de transaction partielles en transactions complètes pour les consoles.

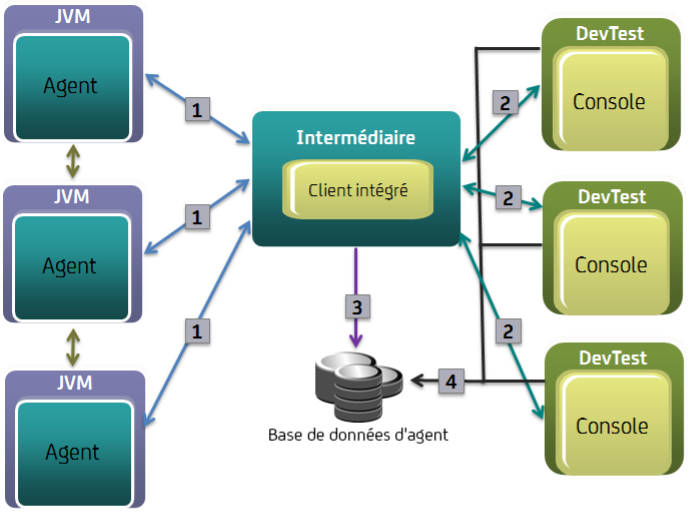
Une fois les données assemblées, elles sont envoyées aux consoles (court terme) et conservées dans la base de données pour le stockage à long terme.

Ce manuel ignore la distinction et fait référence au client intégré en tant qu'intermédiaire.

Les **consoles** obtiennent leurs données finalisées à partir de l'intermédiaire (si elles sont récentes) et de la base de données (si elles sont plus anciennes) pour l'affichage et l'interaction de l'utilisateur. Une console n'est pas nécessairement un composant d'interface utilisateur graphique. Les consoles incluent la station de travail DevTest Workstation, l'environnement VSE et des simulateurs.

Flux de données de l'agent Java

Le graphique suivante illustre les composants de l'agent Java et leur interaction.



Les destinations JMS suivantes spécifient le flux des données :

- La rubrique **lisa.agent.info** est transmise via les connexions 1 et 2, produite par les agents et consommée par l'intermédiaire et les consoles. Cette rubrique permet à l'intermédiaire et aux consoles d'afficher les agents actuellement en ligne et de connaître leurs propriétés de base.
- La rubrique **lisa.agent.port** est transmise via la connexion 1, produite par les agents et consommée par l'intermédiaire. Cette rubrique permet à l'intermédiaire d'afficher les connexions actuellement actives entre plusieurs agents.
- La rubrique **lisa.agent.api** est transmise via les connexions 1 et 2, produite par les consoles et consommée (et envoyée comme réponse) par les agents. Cette rubrique permet aux consoles d'appeler les API d'agent via JMS.
- La rubrique **lisa.broker.api** est transmise via la connexions 2, produite par les consoles et consommée (et envoyée comme réponse) par l'intermédiaire. Cette rubrique permet aux consoles d'appeler les API de l'intermédiaire via JMS.
- La rubrique **lisa.stats** est transmise via les connexions 1 et 2, produite par les agents et consommée par l'intermédiaire et les consoles. Cette rubrique donne aux consoles une idée du type de charge sous laquelle se trouvent les agents. Cette rubrique permet également à l'intermédiaire de les conserver dans la base de données.

- La rubrique **lisa.vse** est transmise via les connexions 1 et 2, produite par les agents et consommée par les consoles. Lorsque le VSE est activé, les consoles reçoivent des trames VSE et leur répondent en mode de lecture.
- La file d'attente **lisa.tx.partial** est transmise via la connexion 1, produite par les agents et consommée par l'intermédiaire. Lorsqu'un agent capture une transaction partielle (toutes les trames produites dans sa machine virtuelle Java), l'agent l'envoie à l'intermédiaire pour l'assemblage.
- La rubrique **lisa.tx.full** est transmise via la connexion 2, produite par l'intermédiaire et consommée par les consoles. Lorsque l'intermédiaire termine l'assemblage des transactions partielles reçues via **lisa.tx.partial**, l'intermédiaire envoie les transactions complètes aux consoles.
- La rubrique **lisa.tx.incomplete** est transmise via la connexion 2, produite par l'intermédiaire et consommée par les consoles. Cette rubrique est similaire à **lisa.tx.full**, mais est utilisée pour des transactions qui n'ont pas pu se terminer avant l'expiration du délai autorisé.
- **La connexion** JDBC 3 est utilisée lorsque l'intermédiaire enregistre des objets **StatsFrame** ou des objets **TransactionFrame** complètement assemblés.
- Les consoles utilisent la **connexion** JDBC 4 pour effectuer leurs requêtes de transactions ou de statistiques qui ne sont plus conservées dans la mémoire.

L'ordre de démarrage des agents, de l'intermédiaire et des consoles importe peu, car toutes les communications sont asynchrones et peuvent se rétablir automatiquement. Ce concept est notamment important pour que les agents puissent éviter des problèmes de performance en cas d'arrêt de l'intermédiaire, par exemple.

Lorsqu'un agent se connecte, il commence à envoyer ses informations via la rubrique **lisa.agent.info** à intervalles standard (courts).

Si un intermédiaire n'est pas disponible, l'agent ne tente pas de notifier les écouteurs d'aucune autre information jusqu'à ce qu'une connexion soit établie ou rétablie.

Si l'intermédiaire est disponible, toutes les parties intéressées seront rapidement notifiées des agents en ligne. L'intermédiaire et les consoles conservent une liste en cours d'exécution de ces agents. Lorsqu'ils cessent d'envoyer leurs informations, ils expirent et sont supprimés de la liste après un certain temps.

Schéma de base de données de l'agent Java

L'intermédiaire crée automatiquement le schéma de base de données de l'agent Java pour DevTest.

Pour créer le schéma manuellement, obtenez les instructions DDL en exécutant la commande suivante :

```
java -jar LisaAgent.jar -ddl <oracle|sqlserver|mysql|derby|db2>
```

Vous pouvez également obtenir les instructions DDL à partir des fichiers suivants dans le répertoire **LISA_HOME\database** :

- **db2_pathfinder.ddl**
- **derby_pathfinder.ddl**
- **mysql_pathfinder.ddl**
- **oracle_pathfinder.ddl**
- **sqlserver_pathfinder.ddl**

En général, vous créez le schéma manuellement pour des raisons de sécurité ou de traitement, ou pour des questions de migration ou liées à la documentation.

Catégories des données de l'agent Java

L'agent Java pour DevTest peut capturer les catégories de données suivantes :

- Client
- EJB
- Interface utilisateur graphique (AWT, Swing, SWT)
- JCA
- JDBC
- JMS
- Logging (Journalisation)
- REST
- RMI
- SAP
- Thread (trame synthétique affichant un assemblage de threads)
- Levables
- TIBCO ActiveMatrix BusinessWorks
- Serveur d'intégration webMethods
- WebSphere MQ
- Web (HTTP/S)
- Service Web

Installation de l'agent Java

Cette section décrit la procédure d'installation de l'agent Java pour DevTest.

Il existe deux version de l'agent Java pour DevTest :

- Agent Java pur
- Agent natif

L'agent Java pur ne contient aucun code dépendant d'une plate-forme.

L'agent natif requiert un module de bibliothèque dépendant d'une plate-forme.

Important : Installez l'agent de Java pur sauf si vous devez créer des références TIBCO BusinessWorks. Cette fonctionnalité requiert l'agent natif.

Cette section comprend les rubriques suivantes :

Installation de l'agent Java pur

Cette rubrique décrit la procédure d'installation de la version Java pur de l'agent Java pour DevTest dans un processus Java.

Remarque : Vous devez d'abord examiner l'environnement cible pour vous assurer qu'il a été testé, ou testez-le vous-même. La plupart des problèmes de l'agent Java sont liés au système d'exploitation ou à une machine virtuelle Java, plutôt qu'à une application. Veillez à suivre les instructions indiquées dans cette documentation. Si vous avez besoin d'informations complémentaires, consultez la rubrique Dépannage de l'agent Java (page 86) avant de contacter le service de support.

Procédez comme suit:

1. Obtenez les fichiers suivants à partir du répertoire **LISA_HOME\agent** :
   - **LisaAgent.jar**
   - **InsightAgent.jar** (pour toutes les plates-formes autres que Oracle WebLogic Server)
   - **LisaAgent2.jar** (pour Oracle WebLogic Server)
2. Placez les fichiers à l'emplacement de votre choix sur le disque disposant de droits de lecture à partir du processus Java et qui n'est pas automatiquement chargé dans le classpath d'applications (par exemple, les répertoires **\WEB-INF\lib**).
3. Accédez à la section Installation de l'agent spécifique aux plates-formes (page 23) et suivez les instructions selon la plate-forme que vous utilisez.

   Pour toutes les plates-formes autres que Oracle WebLogic Server, vous spécifierez une chaîne de paramètres d'agent au format suivant :

   ```
   -javaagent:<chemin_d'accès_au_fichier_InsightAgent.jar>[=url=<url_intermédiaire>][,name=<nom_agent>]
   ```

   Pour Oracle WebLogic Server, vous spécifierez une chaîne de paramètres d'agent au format suivant :

   ```
   -javaagent:<chemin_d'accès_au_fichier_LisaAgent2.jar>[=url=<url_intermédiaire>][,name=<nom_agent>]
   ```
4. Démarrez l'application Java.

Installation de l'agent natif

Cette rubrique décrit la procédure d'installation de la version native de l'agent Java pour DevTest.

Remarque : Vous devez d'abord examiner l'environnement cible pour vous assurer qu'il a été testé, ou testez-le vous-même. La plupart des problèmes de l'agent Java sont liés au système d'exploitation ou à une machine virtuelle Java, plutôt qu'à une application. Veillez à suivre les instructions indiquées dans cette documentation. Si vous avez besoin d'informations complémentaires, consultez la rubrique Dépannage de l'agent Java (page 86) avant de contacter le service de support.

Si vous voulez créer des références TIBCO BusinessWorks, installez l'agent natif. De plus, ajoutez l'option **heap (segment de mémoire)** à la chaîne de paramètres de l'agent et définissez la valeur sur true.

La table suivante contient les noms de fichiers des modules de bibliothèque spécifiques aux plates-formes pour l'agent natif :

Module	**File Name (Nom du fichier)**
Windows (JVM 32 bits)	**JavaBinder.dll**
Windows (JVM 64 bits)	**JavaBinder.amd64.dll**
Mac OS X (x86/x64) 32/64 bits	**libJavaBinder.jnilib**
Linux (x86) 32 bits	**libJavaBinder.x86.so**
Linux (x64) 64 bits	**libJavaBinder.x64.so**
Solaris (x86) 32 bits	**libJavaBinder.solaris.x86.so**
Solaris (x64) 64 bits	**libJavaBinder.solaris.x64.so**
Solaris (Sparc) 32 bits	**libJavaBinder.solaris.sparc32.so**
Solaris (Sparc) 64 bits	**libJavaBinder.solaris.sparc64.so**
HP-UX RISC 32 bits	**libJavaBinder.hp-ux.sl**

Procédez comme suit:

1. Obtenez les fichiers suivants à partir du répertoire **LISA_HOME\agent** :
   - **LisaAgent.jar**
   - Téléchargez le module de bibliothèque spécifique à la plate-forme pour l'environnement cible.
2. Placez les fichiers à l'emplacement de votre choix sur le disque disposant de droits de lecture à partir du processus Java et qui n'est pas automatiquement chargé dans le classpath d'applications (par exemple, les répertoires **\WEB-INF\lib**).
3. Accédez à la section Installation de l'agent spécifique aux plates-formes (page 23) et suivez les instructions selon la plate-forme que vous utilisez. Vous spécifierez une chaîne de paramètres d'agent au format suivant :

   ```
   -agentpath:<chemin_accès_fichier_JavaBinder>=jar=file:<chemin_accès_fichier_LisaAgent.jar>[,url=<url_intermédiaire>][,name=<nom_agent>]
   ```

4. Démarrez l'application Java.

Options pour la chaîne de paramètres d'agent

Lorsque vous installez l'agent Java pour DevTest, vous spécifiez une chaîne de paramètres d'agent. Cette rubrique décrit les options que vous pouvez inclure dans la chaîne.

Important : L'option **name** est requise.

url

Définit l'intermédiaire auquel l'agent se connecte. Utilisez le format suivant :

```
url=tcp://broker-host:broker-port
```

Définissez la partie **broker-host (hôte de l'intermédiaire)** sur le nom ou l'adresse IP de l'ordinateur sur lequel l'intermédiaire est déployé.

Définissez la partie **broker-port (port de l'intermédiaire)** sur le port TCP sur lequel l'intermédiaire écoute. La valeur par défaut est 2009.

name

Nom de l'agent. Utilisez le format suivant :

```
name=nom_agent
```

Veillez à fournir un nom descriptif.

Evitez d'utiliser le même nom pour plusieurs agents.

token (jeton)

Sécurise l'accès de l'agent. Utilisez le format suivant :

```
token=jeton-admin:jeton-utilisateur
```

Pour plus d'informations, consultez la section Sécurité de l'agent Java (page 81).

heap (Segment de mémoire)

L'option **heap** est disponible uniquement avec l'agent natif.

Spécifie si les API de consultation de segment de mémoire, requis pour les références TIBCO BusinessWorks, doivent être activées. Les valeurs valides sont true et false. La valeur par défaut est false.

jar

L'option **jar** est disponible uniquement avec l'agent natif.

Définit le chemin d'accès au fichier **LisaAgent.jar.**

Assistant d'installation de l'agent

L'assistant d'installation de l'agent est un utilitaire de ligne de commande qui permet de déterminer la valeur pour la chaîne de paramètres de l'agent.

L'assistant d'installation de l'agent vérifie l'exécution correcte de la machine virtuelle Java avec l'agent installé.

Remarque : Si vous voulez créer des références TIBCO BusinessWorks, ajoutez l'option **heap** (segment de mémoire) à la chaîne de paramètres de l'agent et définissez la valeur sur true.

Configuration automatique

L'assistant d'installation de l'agent permet de configurer automatiquement l'agent Java sur les plates-formes basées sur les valeurs spécifiées par l'utilisateur dans le fichier **JavaAgentInstaller.xml**. Ce fichier XML est un modèle qui contient les informations sur les plates-formes et les versions prises en charge par DevTest. Le fichier **JavaAgentInstaller.xml** se trouve dans le même répertoire que le fichier **LisaAgent.jar**.

Les étapes préalables suivantes doivent être effectuées afin d'utiliser l'option de configuration automatique de l'agent Java :

Procédez comme suit:

1. Recherchez la plate-forme basée sur le nom de la plate-forme et le type de système d'exploitation répertorié dans le fichier **JavaAgentInstaller.xml.**
2. Définissez la variable d'environnement, le cas échéant.

   La variable d'environnement est intégrée dans ${} ; par exemple : ${JBOSS_HOME}.
3. Remplacez toutes les valeurs personnalisées, le cas échéant.

   Les valeurs personnalisées sont placées entre {} ; par exemple :

   ```
   <ConfigFileName>${ORACLE_HOME}\user_projects\domains\{custom domain name}\bin\startWebLogic.cmd<ConfigFileName>
   ```

   Remplacez-les par votre nom du projet :

   ```
   <ConfigFileName>${ORACLE_HOME}\user_projects\domains\lisa_domain\bin\startWebLogic.cmd</ConfigFileName>
   ```
4. Vérifiez toutes les valeurs prédéfinies et modifiez celles qui dépendent de votre système.

Exécution de l'assistant d'installation de l'agent

La procédure suivante comporte différentes étapes pour saisir le chemin du fichier exécutable **Java,** l'URL de l'intermédiaire et le nom de l'agent. Vous pouvez toutefois spécifier l'une de ces valeurs dans la deuxième étape.

Procédez comme suit:

1. Accédez à l'ordinateur sur lequel vous installez l'agent.
2. Exécutez le fichier **LisaAgent.jar** avec l'option **-ia.**
   - Si vous installez l'agent Java pur, ajoutez l'option **-java.**
   - Si vous installez l'agent natif, ajoutez l'option **-native.**
3. Lorsque vous y êtes invité, saisissez le chemin du fichier exécutable **java**. La valeur par défaut est le chemin du fichier exécutable **java** en cours d'exécution.
4. Lorsque vous y êtes invité, saisissez l'URL d'intermédiaire. La valeur par défaut est **tcp://localhost:2009**.
5. Lorsque vous y êtes invité, saisissez le nom d'agent. La valeur par défaut est **{{x}}**, c'est-à-dire un nom unique généré lors de l'exécution.
6. Vérifiez la sortie.
7. (Facultatif) pour configurer l'agent de manière automatique, saisissez **Yes** et suivez les invites.

Pour plus d'informations, consultez les exemples suivants :

- Exemple 1 : Agent Java pur (page 21)
- Exemple 2 : Agent Java pur avec configuration automatique (page 22)
- Exemple 3 : Agent natif (page 23)

Exemple 1 - Agent Java pur

Cet exemple est basé sur l'agent Java pur. L'option de configuration automatique n'est pas utilisée.

```
java -jar LisaAgent.jar -ia -java
Entrez le chemin du fichier exécutable Java [C:\Program Files\Java\jre7\bin\java] :


Entrez l'URL de l'intermédiaire DevTest [tcp://localhost:2009] :

Remarque : Si le nom de l'agent DevTest contient la séquence de caractères suivantes
:

{{x}}

elle sera remplacée par un identificateur unique.

Entrez le nom de l'agent pour DevTest [{{x}}] :


=====

Vérification du système terminée.  Pour démarrer l'agent DevTest, ajoutez manuellement
ces options à la ligne de commande java :


-javaagent:C:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name={{x}}

Avez-vous besoin d'assistance pour configurer automatiquement l'agent DevTest ? Si
oui, entrez [Yes/Y]. Dans le cas contraire, entrez une autre lettre, puis entrez :
n


Pour démarrer l'agent DevTest, ajoutez manuellement ces options à la ligne de commande
Java :
```

Exemple 2 : Agent Java pur avec configuration automatique

Dans cet exemple, la configuration automatique de l'agent Java pur JBoss 4.2 pour Windows est utilisée.

```
java -jar LisaAgent.jar -ia -java -broker tcp://localhost:2009 -name TestAgent66
Entrez le chemin du fichier exécutable Java [C:\Program
Files\Java\jdk1.7.0_40\jre\bin\java] :
=====
Vérification du système terminée. Pour démarrer l'agent DevTest, ajoutez manuellement
ces options à la ligne de commande java :

-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAgent.jar=url=tcp://l
ocalhost:2009,name=TestAgent66
Avez-vous besoin d'assistance pour configurer automatiquement l'agent DevTest ? Si
oui, entrez [Yes/Y]. Dans le cas contraire, entrez une autre lettre, puis entrez :
y
Entrez la valeur d'index entre [] pour sélectionner la plate-forme :
[1] JBoss
[2] Serveur d'applications IBM WebSphere
[3] Serveur Oracle WebLogic
[4] TIBCO BusinessWorks
[5] Service Windows du serveur d'intégration WebMethods
[X] Quitter
1
La version 4.2 est détectée sur votre système.
Entrez le chemin d'accès au fichier de configuration de l'agent Java
[C:\DevTest\jboss\bin\run.bat] :
La configuration automatique de l'agent Java est terminée. Démarrez le service.
Les informations de configuration ont été mises à jour. Configuration précédente :
SET
JAVA_TOOL_OPTIONS=-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent7 %JAVA_TOOL_OPTIONS%
La configuration précédente a été enregistrée dans le fichier
C:\DevTest\jboss\bin\run.bat.2014-54-25_09-54-22.
Configuration de l'agent actuelle :
SET
JAVA_TOOL_OPTIONS=-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent66 %JAVA_TOOL_OPTIONS%
```

Exemple 3 - Agent natif

Cet exemple est basé sur l'agent natif Solaris (Sparc) 32 bits.

```
$ java -jar dist/agent/LisaAgent.jar -ia -native tcp://localhost:2009 MyAgent
Saisissez le chemin du fichier exécutable Java [...]:
/usr/jdk/instances/jdk1.6.0/bin/java.

=====

Vérification du système terminée.  Ajoutez ces options à la ligne de commande java
pour démarrer l'agent DevTest :


-agentpath:/tmp/user/dist/agent/libJavaBinder.solaris.sparc32.so=jar=file:/tmp/us
er/dist/agent/LisaAgent.jar,url=tcp://localhost:2009,name=MyAgent

Avez-vous besoin d'assistance pour configurer automatiquement l'agent DevTest ? Si
oui, entrez [Yes/Y]. Dans le cas contraire, entrez une autre lettre, puis entrez :

n

Pour démarrer l'agent DevTest, ajoutez manuellement ces options à la ligne de commande
Java :
```

Installation de l'agent spécifique aux plates-formes

Cette section contient des informations spécifiques aux plates-formes pour l'installation l'agent Java pour DevTest.

Cette section se rapporte aux opérations suivantes :

Serveur d'applications IBM WebSphere (page 24)
JBoss (page 26)
Jetty (page 26)
Serveur Oracle WebLogic (page 27)
TIBCO BusinessWorks (page 29)
Serveur d'intégration webMethods (page 30)

Serveur d'applications IBM WebSphere

Cette rubrique s'applique au serveur d'applications IBM WebSphere 7.0 et 8.5.

Pour configurer le serveur d'applications IBM WebSphere pour utiliser l'agent Java pour DevTest, utilisez l'une des approches suivantes :

- Console d'administration
- fichier server.xml
- Outil wsadmin

Pour chaque approche, vous devez spécifier une chaîne de paramètres d'agent en tant qu'argument de machine virtuelle Java générique. Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la valeur requise. L'exemple suivant est basé sur l'agent Java pur :

```
-javaagent:/home/itko/agent/InsightAgent.jar=url=tcp://172.24.255.255:2009,name=was70_linux32
```

Console d'administration

Dans cette procédure, vous utilisez la console d'administration Web pour spécifier la chaîne de paramètres de l'agent.

Procédez comme suit:

1. Accédez à la console d'administration.
2. Accédez à votre serveur d'applications.
3. Développez Java and Process Management et dans l'onglet Configuration, cliquez sur le lien Process Definition (Définition de processus).
4. Cliquez sur Java Virtual Machine (Machine virtuelle Java) sous Additional Properties (Propriétés supplémentaires).
5. Spécifiez la chaîne de paramètres de l'agent dans le champ Generic JVM arguments (Arguments de la machine virtuelle Java génériques).

Fichier server.xml

Dans cette procédure, vous utilisez le fichier **server.xml** pour spécifier la chaîne de paramètres de l'agent.

Procédez comme suit:

1. Accédez au répertoire **WAS_HOME/AppServer/profiles/AppSrv01/config/cells/<cell_name>/nodes/<node_name>/servers/server1**.
2. Ouvrez le fichier **server.xml.**
3. Au bas du fichier, modifiez l'entrée **genericJvmArguments.**

   ```
   <jvmEntries... genericJvmArguments="<agent_parameters_string>" .../>
   ```

Outil wsadmin

Vous pouvez utiliser l'outil **wsadmin.**

Dans l'exemple suivant, la chaîne de paramètres de l'agent est spécifiée avec la commande de modification de l'objet **AdminConfig**. Le langage de script Jacl est utilisé.

```
C:\IBM\WebSphere70\AppServer\bin>hostname
cam-aa74651f617

C:\IBM\WebSphere70\AppServer\bin>wsadmin
WASX7209I: Connected to process "server1" on node cam-aa74651f617Node01 using SOAP
connector;  The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"

wsadmin>set server1 [$AdminConfig getid
/Cell:cam-aa74651f617Node01Cell/Node:cam-aa74651f617Node01/Server:server1/ ]
server1(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/serve
r1|server.xml#Server_1255494205517)

wsadmin>set jvm [$AdminConfig list JavaVirtualMachine $server1]
(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/server1|serv
er.xml#JavaVirtualMachine_1255494205517)

wsadmin>$AdminConfig modify $jvm genericJvmArguments "<agent_parameters_string>"

wsadmin>$AdminConfig save

wsadmin>quit
```

JBoss

Cette rubrique s'applique à JBoss 4.2 et 7.

Pour configurer JBoss pour utiliser l'agent Java pour DevTest, modifiez le script de démarrage de JBoss.

Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la valeur de la chaîne de paramètres d'agent.

Procédez comme suit:

1. Ouvrez le fichier de JBoss **run.bat** ou **run.sh** dans un éditeur de texte.
2. Avant d'appeler le fichier exécutable Java, définissez la propriété suivante :

   ```
   set JAVA_OPTS=<agent_parameters_string>
   ```
3. Enregistrez le fichier.

Jetty

Cette rubrique s'applique à Jetty 8 et 9.*x*.

Pour configurer Jetty pour utiliser l'Agent Java pour DevTest, spécifiez la chaîne de paramètres d'agent dans la variable d'environnement **JAVA_TOOL_OPTIONS.**

Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la valeur de la chaîne de paramètres d'agent.

```
set JAVA_TOOL_OPTIONS=<agent_parameters_string>
"%JAVA_HOME%/bin/java.exe" -DSTOP.PORT=28282 -DSTOP.KEY=secret -jar
start.jar etc/jetty-logging.xml
```

Serveur Oracle WebLogic

Cette rubrique s'applique à Oracle WebLogic Server 10.3 et au 12.1.1.

Pour configurer Oracle WebLogic Server pour utiliser l'agent Java pour DevTest, utilisez l'une des approches suivantes :

- Console d'administration
- fichier config.xml
- Script de démarrage

Pour chaque approche, vous devez spécifier une chaîne de paramètres d'agent en tant qu'argument de démarrage du serveur. Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la valeur requise. L'exemple suivant est basé sur l'agent Java pur :

```
-javaagent:/export/home/wls/agent/LisaAgent2.jar=url=tcp://172.24.255.255:2009,name=wls
```

Console d'administration

Pour suivre la méthode officiellement prise en charge pour ajouter des arguments à la machine virtuelle Java, spécifiez la chaîne de paramètres de l'agent à partir de la console d'administration WebLogic.

Procédez comme suit:

1. Ouvrez la console d'administration WebLogic.
2. Dans le panneau Domain Structure (Structure de domaine), développez le noeud Environments (Environnements) et sélectionnez le noeud Servers (Serveurs).
3. Sélectionnez l'onglet Configuration, puis le sous-onglet Server Start (Démarrage de serveur).
4. Dans le champ Arguments, ajoutez la chaîne de paramètres de l'agent.
5. Enregistrez vos modifications.

Fichier config.xml

Une autre méthode consiste à modifier le fichier de configuration central pour le domaine.

Procédez comme suit:

1. Accédez au répertoire **WEBLOGIC_HOME\user_projects\domains\base_domain\config.**

2. Ouvrez le fichier **config.xml.**
3. Ajoutez la chaîne de paramètres de l'agent au noeud **<server>/<server-start>/<arguments>** du serveur cible.

Script de démarrage

Vous pouvez également modifier le script de démarrage WebLogic.

Dans l'exemple suivant, la variable d'environnement **JAVA_TOOL_OPTIONS** est utilisée pour définir la chaîne de paramètres de l'agent. Ces lignes sont disponibles dans le fichier **startWebLogic.sh**, après la vérification de la version Java et avant l'appel WebLogic réel.

```
JAVA_TOOL_OPTIONS=<chaîne_paramètres_agent>
export JAVA_TOOL_OPTIONS
```

Si le script de démarrage a été personnalisé, la variable d'environnement **JAVA_TOOL_OPTIONS** ne sera pas utilisée.

TIBCO BusinessWorks

Cette rubrique s'applique à TIBCO BusinessWorks 5.*x*.

Pour configurer l'agent pour toutes les applications, modifiez le fichier suivant :

```
...\tibco\bw\5.x\bin\bwengine.tra
```

La modification de ce fichier n'affecte pas les applications déjà créées.

Pour configurer l'agent pour une application existante, modifiez le fichier suivant :

```
...\tibco\tra\domain\<domain-name>\application\<application-name>\<applicatio
n-name>-<service-name>.tra
```

Où :

- <domain-name> est le nom de votre domaine TIBCO.
- <application-name> est le nom de l'application TIBCO déployée.
- <service-name> est le nom du service déployé.

Par exemple :

```
D:\tibco\tra\domain\itko-tibcobw\application\MyBWProjectXml\MyBWProjectXml-it
koUserCRUD.tra
```

Dans l'un des fichiers, ajoutez la ligne suivante :

```
java.extended.properties=<agent-parameters-string>
```

Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la chaîne. Si vous voulez créer des références TIBCO BusinessWorks, veillez à ajouter l'option **heap (segment de mémoire)** et à définir la valeur sur **True**.

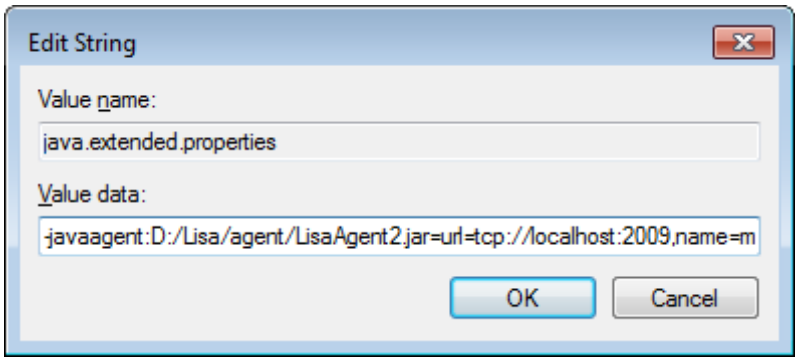
Différence de configuration pour le service Windows

Si le processus TIBCO est exécuté en tant que service Windows, n'ajoutez pas la ligne **java.extended.properties** au fichier .tra. A la place, spécifiez les informations requises dans le registre Windows.

Dans l'éditeur du registre Windows, recherchez la clé de registre pour le service TIBCO BusinessWorks pour l'application. Le nom habituel est **nom-domaine.nom-application**.

Sous cette clé de registre, recherchez la clé Parameters (Paramètres). Sous la clé Parameters (Paramètres), créez ou modifiez une valeur de chaîne avec le nom **java.extended.properties**. Définissez les données de valeur sur la chaîne de paramètres de l'agent.

Le graphique suivant illustre un exemple de définition des données de valeur.



Serveur d'intégration webMethods

Cette rubrique s'applique au serveur d'intégration webMethods 9.0 et 9.5.

Spécifiez la chaîne de paramètres de l'agent dans le fichier **server.bat** ou **server.sh**. Vous pouvez utiliser l'assistant d'installation de l'agent (page 19) pour déterminer la valeur requise. L'exemple suivant est basé sur l'agent Java pur :

```
set
JAVA_TOOL_OPTIONS=-javaagent:E:/agent/InsightAgent.jar=url=tcp://172.24.255.255:2
009,name=wm
```

Procédez comme suit:

1. Recherchez le répertoire **bin** dans l'installation du serveur d'intégration webMethods.
2. Ouvrez le fichier **server.bat** ou **server.sh.**
3. Ajoutez la chaîne de paramètres de l'agent avant la ligne qui appelle le fichier exécutable **Java**.
4. Enregistrez le fichier.

Service Windows

Si le serveur d'intégration webMethods est défini en tant que service Windows, vous pouvez modifier le fichier **server.bat** en insérant la chaîne de paramètres de l'agent dans la valeur **/jvmargs** définie dans le commutateur **if "1%1"=="1-service".** Par exemple :

```
"%IS_DIR%\bin\SaveSvcParams.exe" /svcname %2 /jvm "%JAVA_DIR%\.." /binpath "%PATH%"
/classpath %CLASSPATH% /jvmargs
"-javaagent:c:/DevTest/agent/InsightAgent.jar=name=MyIS %JAVA2_MEMSET%" /progargs
"%IS_DIR%\bin\ini.cnf"#"-service
%2"#%PREPENDCLASSES_SWITCH%#%PREPENDCLASSES%#%APPENDCLASSES_SWITCH%#%APPENDCLASSE
S%#%ENV_CLASSPATH_SWITCH%#%ENV_CLASSPATH%#%3#%4#%5#%6#%7#%8#%9
```

Agent Wily Introscope

L'agent Java pour DevTest peut coexister avec l'agent Wily Introscope.

L'agent Wily est un agent Java pur, généralement configuré à l'aide de la syntaxe **-javaagent**.

Si vous utilisez la syntaxe **-javaagent** pour les deux agents, spécifiez l'agent Java pour DevTest *avant* l'agent Wily Introscope.

Pour un serveur ou un conteneur d'applications habituel, l'exemple suivant est approprié :

```
set
JAVA_OPTS=-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,n
ame=DevTestAgent -javaagent:e:/wily/Agent.jar %JAVA_OPTS%
```

L'exemple suivant convient également :

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar %JAVA_OPTS%
set
JAVA_TOOL_OPTIONS=-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhos
t:2009,name=DevTestAgent
```

En revanche, l'exemple suivant est incorrect :

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar
-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name=DevTes
tAgent %JAVA_OPTS%
```

Rappelez-vous d'effectuer l'action suivante à un endroit de la ligne :

```
-Dcom.wily.introscope.agentProfile=e:/wily/core/config/IntroscopeAgent.profile
```

Utilisation de l'agent Java

Cette section comprend les rubriques suivantes :

Démarrage de l'intermédiaire

Si l'intermédiaire n'est pas installé comme service Windows, démarrez-le manuellement.

Pour afficher les agents dans le tableau de bord Enterprise Dashboard et le portail DevTest, l'intermédiaire doit être exécuté.

Procédez comme suit:

1. Vérifiez que le registre est en cours d'exécution.
2. Effectuez l'une des actions suivantes :
   - Ouvrez une invite de commande, accédez au répertoire **LISA_HOME\bin** et lancez l'exécutable **Broker**.
   - Si votre installation contient un dossier dans le menu Démarrer, cliquez sur le menu Démarrer, Tous les programmes, DevTest Solutions, Broker (Intermédiaire).

Fichier rules.xml

Le fichier de configuration pour l'agent Java pour DevTest est appelé **rules.xml**.

Ce fichier se trouve dans le répertoire **LISA_HOME** de l'ordinateur sur lequel l'intermédiaire est exécuté. Le fichier **rules.xml** est généré lors du premier démarrage de l'intermédiaire. Tous les agents reliés à l'intermédiaire utilisent les paramètres.

Par défaut, le fichier **rules.xml** inclut uniquement un ensemble d'exemple de propriétés de configuration. Toutes les propriétés et leurs valeurs par défaut sont conservées en interne.

Vous pouvez afficher ces propriétés dans un fichier nommé **rules.xml.sample**. Le fichier **rules.xml.sample** est généré lors du premier démarrage de l'intermédiaire. Ce fichier se trouve dans le même répertoire que le fichier **rules.xml**.

Chaque propriété inclut un commentaire, le nom et la valeur. Par exemple :

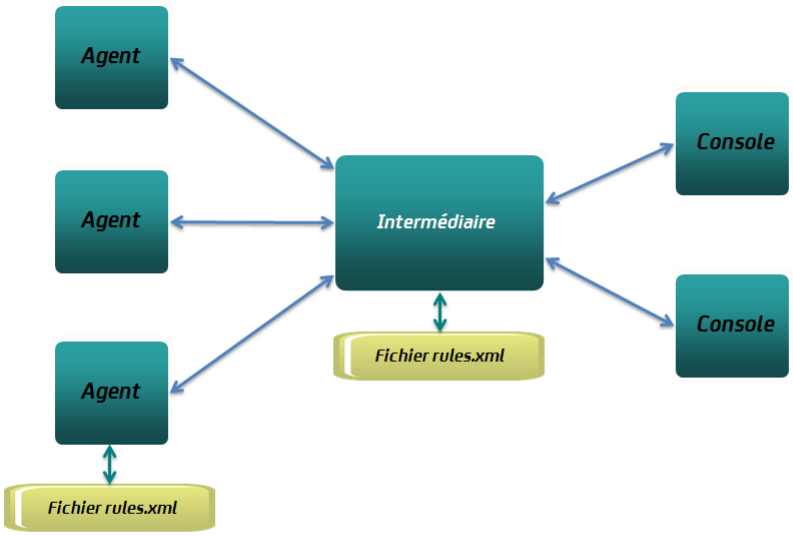
```
<property comment="Keep track of file descriptors"
key="lisa.agent.tracking.disabled" value="false"/>
```

Le code XML suivant représente le format général du fichier **rules.xml.** L'élément **agent** contient les propriétés d'un agent. L'élément **broker** contient les propriétés de l'intermédiaire. L'élément **console** contient les propriétés des consoles.

```
&lt;?xml version="1.0" encoding="UTF-8"?&gt;
<rules>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

Vous pouvez également enregistrer un fichier **rules.xml** côté agent ou côté console. Les paramètres de ces fichiers remplacent toutes les informations relatives à l'agent ou à la console du côté de l'intermédiaire.

Le graphique suivant illustre un exemple de configuration. L'intermédiaire accède au fichier **rules.xml** disponible sur le même ordinateur. Plusieurs agents et consoles sont connectés à l'intermédiaire. L'un d'eux possède son propre fichier **rules.xml.**



Remarque : Vous pouvez configurer les propriétés suivantes uniquement côté agent, car l'agent les lit avant d'établir une connexion à l'intermédiaire :

- lisa.agent.agent.log
- lisa.agent.log.max.size
- lisa.agent.log.max.archives
- lisa.agent.security.manager.disabled
- lisa.agent.transaction.auto.start
- lisa.agent.transaction.auto.start.max.delay
- lisa.broker.encryption.token
- lisa.log.level

Vous pouvez ajouter des *directives* au fichier **rules.xml** pour effectuer certaines fonctions. Pour plus d'informations sur les directives, consultez les rubriques suivantes :

- Directive **category** (Catégorie) : Paramètres de catégorie (page 44)
- **database directive (directive de base de données)** : Configuration d'un récepteur de base de données (page 43)
- Directive **exclude** (Exclure) : Exclusion de l'interception et de la virtualisation (page 47)
- Directive **feature** (Fonctionnalité) : Configuration de poids de protocole (page 38)

- Directive **intercept** (Interception) : Ajout d'une méthode d'interception (page 46)
- Directive **loadbalancer** (Equilibreur de charge) : Equilibreurs de charge et serveurs Web natifs (page 80)
- Directive **response** (Réponse) : Génération de réponses automatiques d'agent Java (page 49)
- Directive **track** (Suivi) : Ajout d'une classe pour le suivi (page 45)
- Directive **virtualize** (Virtualisation) : Règles d'instrumentation pour le VSE (page 41)

Gestion d'agents dans des groupes

Vous pouvez créer un *groupe* qui contient plusieurs agents.

Vous pouvez alors appliquer des changements de configuration au niveau du groupe, au lieu d'effectuer les mêmes modifications pour chaque agent.

Remarque : Un agent ne peut pas appartenir à plusieurs groupes.

Un agent qui n'appartient à aucun groupe est un agent *autonome*.

Le code XML suivant représente le format général du fichier **rules.xml** contenant un groupe. L'élément **group** contient les propriétés du groupe. L'élément **agent** contient les propriétés d'un agent. Si l'agent fait partie d'un groupe, l'élément **agent** s'affiche dans l'élément **group.** L'élément **broker** contient les propriétés de l'intermédiaire. L'élément **console** contient les propriétés des consoles.

```
&lt;?xml version="1.0" encoding="UTF-8"?&gt;
<rules>
   <group name="G1">
      ...
      <agent name="G1A1">
         ...
      </agent>
   </group>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

Si un élément **agent** dans un élément **group** a une ou plusieurs propriétés, elles remplaceront les paramètres du groupe.

Ne placez pas l'élément **broker** ou **console** dans l'élément **group.**

Lorsque vous mettez à jour une propriété d'agent à partir du portail DevTest, le paramètre est enregistré dans la section d'agent autonome du fichier **rules.xml**.

Remarque : Pour plus d'informations sur le portail DevTest, consultez la rubrique *Utilisation de CA Continuous Application Insight*.

Exemple de configuration d'un groupe dans le fichier rules.xml

Le code XML suivant représente un fichier **rules.xml** contenant un groupe. Le groupe a trois agents. Ce fichier comprend également un agent autonome.

```
&lt;?xml version="1.0" encoding="UTF-8"?&gt;
<rules>
   <group name="group1">
      <property key="lisa.agent.jms.poll.int" value="5000"/>
      <property key="lisa.agent.transaction.auto.start.max.delay"
value="60000"/>
      <property key="lisa.agent.virtualize.jit.enabled"
value="false"/>
      <intercept class="com.itko.examples.entity.Account"
method="setName" signature="(Ljava/lang/String;)V"/>
      <agent name="agent1" guid="1234567">
         <property key="lisa.agent.stats.alarm.threshold.permgen"
value="90"/>
         <property key="lisa.agent.stats.sampling.interval"
value="1000"/>
      </agent>
      <agent name="agent2" guid="2345678" />
      <agent name="agent3" guid="3456789" />
   </group>
   <agent guid="-2032180703" name="DEFAULT">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

Configuration de poids de protocole

La directive **feature** (fonctionnalité) permet de modifier le niveau de capture pour chaque protocole que l'agent Java pour DevTest peut capturer.

Remarque : La définition de niveaux de capture différents n'est pas prise en charge pour le client de file d'attente et la communication de le serveur ; par exemple, WebSphere MQ et JMS.

Vous pouvez également modifier les niveaux de capture à partir du portail DevTest. Pour plus d'informations, reportez-vous à la rubrique *Utilisation de CA Continuous Application Insight*.

La directive **feature** a le format suivant :

```
<feature name="protocol_name" weight="weight"/>
```

Vous pouvez placer la directive **feature** dans l'élément **group** ou **agent** du fichier **rules.xml.**

Définissez l'attribut **weight** sur 0, 4 ou 8. La valeur 0 correspond au niveau **Counts** (Nombres). La valeur 4 correspond au niveau **Counts and Paths** (Nombres et chemins). La valeur 8 correspond au niveau **Full Data** (Données complètes).

L'exemple suivant définit le protocole JDBC au niveau **Full Data** :

```
<feature name="JDBC" weight="8"/>
```

Spécification de signature

Les directives suivantes du fichier **rules.xml** incluent une spécification de signature :

- exclude (exclure)
- intercept (intercept)
- respond (répondre)
- virtualize (virtualiser)

La spécification de signature permet de décrire les caractéristiques d'une signature de méthode Java.

La première partie de la spécification est le terme **signature**, suivi du signe égal et d'un point d'interrogation.

La deuxième partie de la spécification contient les arguments, placés entre parenthèses. Même si la méthode ne contient aucun argument, les parenthèses doivent être présentes.

La troisième partie de la spécification contient le type de retour, suivi par un point d'interrogation. Si le type de retour est vide, utilisez la lettre V.

N'incluez pas d'espaces dans la spécification.

Pour spécifier un type de primitive dans les arguments ou le type de retour, utilisez l'une des lettres suivantes :

Lettre	Type de primitive
Z	booléen
B	octet
C	car
D	double
F	flottant
I	int
J	long
S	entier court

Pour spécifier une classe complète, suivez la procédure suivante :

- Ajoutez la lettre L au début.
- Utilisez une barre oblique comme séparateur, au lieu du point.
- Ajoutez un point-virgule à la fin.

Par exemple :

```
Ljava/lang/String;
```

Exemple : One Argument, Returns Void (Un argument, renvoie une valeur vide)

Supposons que vous voulez intercepter la méthode **onMessage()** de l'interface **javax.jms.MessageListener.** Cette méthode a la signature suivante :

- L'argument est un objet **javax.jms.Message.**
- Le type de retour est vide.

La spécification de signature dans la règle d'intersection se présenterait comme suit :

```
signature="(Ljavax/jms/Message;)V"
```

Exemple : No Arguments, Returns Primitive Type (Aucun argument, renvoie le type de primitive)

Supposons que vous voulez intercepter la méthode **getPriority()** de l'interface **javax.jms.MessageProducer.** Cette méthode a la signature suivante :

- La méthode ne contient aucun argument.
- Le type de retour est un nombre entier.

La spécification de signature dans la règle d'intersection se présenterait comme suit :

```
signature="()I"
```

Règles d'instrumentation pour le VSE

Vous pouvez personnaliser la fonctionnalité du VSE en ajoutant la directive **virtualize** (virtualiser) au fichier **rules.xml** de l'agent.

Vous pouvez placer la directive **virtualize** dans l'élément **group** ou **agent** du fichier **rules.xml.**

Ajout d'une classe pour la virtualisation (modes d'enregistrement et de lecture)

```
<virtualize class="class_name"/>
```

Exemple :

```
<virtualize class="javax.ejb.SessionBean"/>
```

Indication à l'agent de la méthode permettant de déterminer une session d'un protocole spécifique

Pour effectuer cette tâche, fournissez un extrait de code qui renvoie un identificateur de session.

```
<virtualize>
   <track class="class_name" method="method_name"
signature="signature" push="true|false">
      <code><![CDATA[" and ends with "]]></code>
   </track>
</virtualize>
```

Le format de la signature est décrit dans la section Spécification de signature (page 39).

Exemples :

```
<virtualize>
   <track class="javax.servlet.http.HttpServlet" method="service"
signature="(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/
http/HttpServletResponse;)V" push="false">
      <code><![CDATA["return $1.getSession().getId();"]]></code>
   </track>
</virtualize>
```

```
<virtualize>
   <track class="javax.ejb.SessionBean" method="setSessionContext"
signature="(Ljavax/ejb/SessionContext)V" push="true">
```

```
      <code><![CDATA["return
$1.getEJBObject().getHandle().toString();"]]></code>
   </track>
</virtualize>


<virtualize>
   <track class="javax.ejb.EntityBean" method="setEntityContext"
signature="(Ljavax/ejb/EntityContext)V" push="true">
      <code><![CDATA["return
$1.getPrimaryKey().toString();"]]></code>
   </track>
</virtualize>
```

Ces exemples sont codés de manière irréversible dans l'agent ; ils sont donc inutiles dans votre fichier **rules.xml.** Toutefois, ces lignes indiquent le fonctionnement du processus pour son implémentation pour des protocoles, autres que les protocoles HTTP et EJB, sans recompiler l'agent.

La valeur de l'attribut **class** est la classe avec laquelle vous accédez à une session.

La valeur des attributs **method** et **signature** déterminent la méthode qui calcule l'identificateur de session lorsqu'elle est appelée. Ce calcul utilise la valeur de l'attribut **code**. **$0** représente l'objet source. **$1**, **$2**, etc., sont les arguments de méthode.

L'attribut **push** (transmettre) définit la méthode de stockage de cette session pour son utilisation ultérieure par les trames du VSE.

- **push="true"** indique que le conteneur définit la session et qu'un mappage de l'objet est conservé pour la session.
- **push="false"** signifie que la session est limitée par un thread et que le stockage est effectué dans une variable de thread local.

La session la plus profonde (identificateur) est renvoyée via le VSE dans la méthode **com.itko.lisa.remote.vse.VSEFrame getSessionId()**. Pour plus d'informations, consultez le document JavaDocs dans le répertoire **LISA_HOME\doc**.

Configuration d'un récepteur de base de données

Vous pouvez ajouter la directive **database** (Base de données) à l'élément **broker** du fichier **rules.xml.**

Ce paramètre sera utilisé pendant la durée de vie de l'agent.

```
<database driver="mySinkDriver" url="mySinkURL" user="mySinkUser"
password="mySinkPassword"/>
```

Exemple :

```
<database driver="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/agtdb" user="postgres"
password="postgres"/>
```

Paramètres de catégorie

L'agent Java pour DevTest affecte une catégorie à chaque trame de transaction. Si l'agent ne peut pas déterminer la catégorie, la catégorie par défaut est affectée.

Vous pouvez configurer l'agent pour associer une classe ou une interface interceptée à l'une des catégories autres que par défaut. Ajoutez la directive **category** (catégorie) au fichier **rules.xml** de l'agent. Spécifiez le nom de la classe ou de l'interface et le numéro de la catégorie.

La directive **category** a le format suivant :

```
<category class="nom_classe_ou_interface"
value="numéro_catégorie"/>
```

La directive **category** doit être placée dans l'élément **broker.**

Les numéros de catégorie valides sont les suivants :

- CATEGORY_DEFAULT = 0
- CATEGORY_THREAD = 5
- CATEGORY_THROWABLE = 10
- CATEGORY_GUI = 7
- CATEGORY_GUI_SWT = 9
- CATEGORY_LOGGING = 15
- CATEGORY_WEB_HTTP = 20
- CATEGORY_WEB_HTTPS = 21
- CATEGORY_WS_HTTP = 22
- CATEGORY_WS_HTTPS = 23
- CATEGORY_REST_HTTP = 24
- CATEGORY_RMI = 30
- CATEGORY_RMI_HTTP = 31
- CATEGORY_RMI_SSL = 32
- CATEGORY_EJB = 40
- CATEGORY_JDBC = 50
- CATEGORY_JCA = 55
- CATEGORY_JMS = 60
- CATEGORY_MQ = 61
- CATEGORY_WM = 62
- CATEGORY_TIBCO = 64

- CATEGORY_AMX = 70
- CATEGORY_FRAMEWORK = 80
- CATEGORY_CLIENT = 90
- CATEGORY_CICS = 100
- CATEGORY_WPS = 120
- CATEGORY_SAP = 130
- CATEGORY_SYNTHETIC_ROOT = 140
- CATEGORY_DN_DEFAULT = 1000
- CATEGORY_DN_REMOTING = 1010
- CATEGORY_DN_SQL = 1020

Exemple :

```
<category class="com.mycompany.GuiClass" value="7"/>
```

Remarque : Bien que cela soit possible, il n'est pas recommandé de modifier les catégories de trames de transaction déjà affectées à une catégorie autre que par défaut par l'agent.

Ajout d'une classe pour le suivi

La directive **track** (Suivi) permet de facilement récupérer des instances du segment de mémoire ultérieurement.

Vous pouvez placer la directive **track** dans l'élément **group** ou **agent** du fichier **rules.xml.**

```
<track class="nom_classe"/>
```

Exemple :

```
<track class="java.io.File"/>
```

Ajout d'une méthode d'interception

Vous pouvez ajouter une méthode d'interception à l'agent Java pour DevTest en ajoutant la directive **intercept** (intercepter) au fichier **rules.xml** de l'agent.

Le format de la directive **intercept (intercepter)** se présente comme suit :

```
<intersection class="nom_classe" method="nom_méthode"
signature="signature"/>
```

Vous pouvez placer la directive **intercept** dans l'élément **group** ou **agent** du fichier **rules.xml.**

Le format de la signature est décrit dans la section Spécification de signature (page 39).

Les hiérarchies de classes sont prises en compte. Supposons que la classe B étend la classe A et que les deux classes définissent la méthode M. Si vous interceptez la méthode M de la classe A, la méthode M de la classe B sera également capturée.

Par défaut, CA Continuous Application Insight ne capture pas les méthodes getter et setter. Pour ajouter une méthode getter ou setter, incluez l'attribut **delay** avec la valeur définie sur true.

Exemples

Dans l'exemple suivant, la méthode **setName()** de la classe **com.itko.examples.entity.Account** est ajoutée. La méthode contient un objet **java.lang.String** comme argument. La méthode renvoie une valeur vide.

```
<intercept class="com.itko.examples.entity.Account"
method="setName" signature="(Ljava/lang/String;)V" delay="true"/>
```

Dans l'exemple suivant, la méthode **getTransactions()** de la classe **com.itko.examples.entity.Account** est ajoutée. La méthode ne contient aucun argument. La méthode renvoie un objet **java.util.Collection.**

```
<intercept class="com.itko.examples.entity.Account"
method="getTransactions" signature="()Ljava/util/Collection;"
delay="true"/>
```

Dans l'exemple suivant, la méthode **getRequestTypes()** de la classe **com.itko.examples.airline.ws.jaxws.Request** est ajoutée. La méthode ne contient aucun argument. La méthode renvoie un tableau d'objets **java.lang.String.**

```
<intercept class="com.itko.examples.airline.ws.jaxws.Request"
method="getRequestTypes" signature="()[Ljava/lang/String;"
delay="true"/>
```

Exclusion de l'interception et de la virtualisation

Vous pouvez empêcher l'agent Java pour DevTest d'intercepter ou de virtualiser une méthode, une classe ou un package en ajoutant la directive **exclude** (exclure) au fichier **rules.xml** de l'agent.

Le format de la directive **exclude (exclure)** se présente comme suit :

```
<exclude class="nom_classe" method="nom_méthode"
signature="signature"/>
```

Vous pouvez placer la directive **exclude** dans l'élément **group** ou **agent** du fichier **rules.xml.**

Le format de la signature est décrit dans la section Spécification de signature (page 39).

Les hiérarchies de classes ne sont pas prises en compte. Supposons que la classe B étend la classe A et que les deux classes définissent la méthode M. Si vous excluez la classe A, la méthode M de la classe A ne sera pas capturée. Toutefois, la méthode M de la classe B le sera.

Si vous voulez exclure une classe ou un package sans tenir compte de la méthode ou de la signature, vous pouvez procéder de l'une des manières suivantes :

- Spécifiez **method="*" signature="*".**
- Ignorez les attributs **method** et **signature.**

Si vous ajoutez ou supprimez la directive **exclude** vous devez redémarrer l'agent.

Par défaut, CA Continuous Application Insight ne capture pas les méthodes getter et setter. Il n'est pas nécessaire d'exclure les méthodes getter et setter.

Exemples

Dans l'exemple suivant, la méthode **serviceComposition()** de la classe **com.itko.examples.ejb3.OrdinaryBean** est exclue. La méthode ne contient aucun argument. La méthode renvoie un objet **java.lang.String.**

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean"
method="serviceComposition" signature="()Ljava/lang/String;"/>
```

Dans l'exemple suivant, la classe **com.itko.examples.ejb3.OrdinaryBean** est exclue.

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean"/>
```

Dans l'exemple suivant, le package **com.itko.examples.ejb** est exclu. Vous pouvez observer l'utilisation d'un caractère générique unique.

```
<exclude class="com.itko.examples.ejb.*"/>
```

L'exemple suivant exclut le package **com.itko.examples** et tous ses sous-packages. Vous pouvez observer l'utilisation de deux caractères génériques.

```
<exclude class="com.itko.examples.**"/>
```

Génération de réponses automatiques d'agent Java

Vous pouvez utiliser l'agent Java pour DevTest pour enregistrer des transactions, même lorsque l'arrière-plan n'est pas disponible.

Pour configurer cette fonctionnalité, ajoutez la directive **respond** (Répondre) au fichier **rules.xml.** La directive **respond** a le format suivant :

```
<respond class="class_or_interface_name" method="method_name"
signature="signature" source="string_value" args="string_value"
return="class_name"/>
```

Cette directive doit être placée dans l'élément **agent** du fichier **rules.xml.**

L'attribut **class** est le seul attribut requis.

Utilisez les attributs suivants pour spécifier la ou les méthodes que l'agent doit intercepter :

- **class** : nom de la classe ou de l'interface qui définit la ou les méthodes. Si vous spécifiez une interface, les classes qui implémentent l'interface sont également incluses.
- **method** : nom d'une méthode. Si vous n'incluez pas cet attribut, toutes les méthodes de la classe ou de l'interface sont incluses.
- **signature** : signature de la méthode. Le format est décrit dans la section Spécification de signature (page 39).
- **source** : si l'appel de la méthode **toString()** sur l'objet contient cette valeur de chaîne, la méthode est interceptée.
- **args** : si l'appel de la méthode **toString()** sur les arguments contient cette valeur de chaîne, la méthode est interceptée.

Lorsque l'agent intercepte un appel de méthode qui correspond à la spécification, il effectue les actions suivantes :

- Détermine le type d'objet à renvoyer.
- Renvoie une version générique de l'objet. L'objet contient des valeurs aléatoires.

Vous pouvez utiliser l'attribut **return** pour spécifier le type d'objet et remplacer ainsi la première étape.

Si une valeur contient un signe moins, remplacez-le par le texte suivant :

```
&lt;
```

Si une valeur contient un signe supérieur à, remplacez-le par le texte suivant :

```
&gt;
```

Exemple : Génération de réponses automatiques pour les appels EJB

La méthode Java suivante crée un objet de contexte JNDI, appelle un serveur pour exécuter les objets EJB, puis appelle les API.

```
public void doEJBCall() throws Exception
    {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        /* premier appel distant (facultatif) */
        Context ctx = new InitialContext(props);

        /* second appel distant */
        EJB3UserControlBeanRemote remote = (EJB3UserControlBeanRemote)
ctx.lookup("EJB3UserControlBean/remote");

        /* troisième appel distant */
        User u = remote.getUser("lisa_simpson");

        /* Local calls... */
        System.out.println(u.getEmail());
    }
```

Supposez que le serveur n'est pas disponible. Si vous exécutez cette méthode, le code renvoie un échec , car il ne peut pas établir de connexion.

Vous pouvez utiliser les directives **respond** suivantes pour forcer la réussite du code. La première directive intercepte la création de l'objet **InitialContext.** La seconde directive intercepte la méthode **lookup()** de l'objet **Context.** La troisième directive intercepte les méthodes de l'objet **EJB3UserControlBeanRemote.**

```
<respond class="javax.naming.InitialContext" method="&lt;init&gt;"
args="1099"/>

<respond class="javax.naming.Context" method="lookup"
args="EJB3UserControlBean/remote"/>

<respond class="com.itko.examples.ejb3.EJB3UserControlBeanRemote"/>
```

Développement au niveau de l'agent Java

L'interface principale de l'agent Java pour DevTest côté client est **com.itko.lisa.remote.client.AgentClient**. Cette classe comprend des méthodes d'appel d'API sur les agents ou l'intermédiaire, pour détecter des agents et connaître les changements de statut (en ligne/hors ligne).

Cette classe permet également d'accéder aux classes responsables de l'interaction de l'agent dans les zones principales de la fonctionnalité :

- **com.itko.lisa.remote.client.AgentClient**
- **com.itko.lisa.remote.client.DiscoveryClient**
- **com.itko.lisa.remote.client.TransactionsClient**
- **com.itko.lisa.remote.client.VSEClient**

Remarque : Vous pouvez consulter des informations détaillées sur ces classes dans le document JavaDocs pour l'agent. Les documents JavaDocs sont disponibles dans le répertoire **LISA_HOME\doc**.

Cette section comprend les rubriques suivantes :

API générales de l'agent

Remarque : Vous pouvez consulter des informations détaillées sur l'interface et la classe décrite dans cette rubrique dans le document JavaDocs pour l'agent. Les documents JavaDocs sont disponibles dans le répertoire **LISA_HOME\doc**.

La plupart des API clients doivent spécifier un agent en tant que cible. Cette spécification s'effectue via la transmission d'un paramètre de type **com.itko.lisa.remote.IAgentInfo**. Ce paramètre représente un objet qui identifie de manière unique un agent et certaines de ses informations de base.

```
/** Identificateur unique de cet agent ou de cette console. S'il est nommé, le
GUID sera conservé pendant la durée de vie de la machine virtuelle */
 public long getGuid();
 public void setGuid(long guid);

 /** nom lisible permettant d'identifier cet agent, attribué manuellement ou
généré à partir de propriétés système */
 public String getName();
 public void setName(String name);

 /** Nom de la machine sur laquelle cet objet a été généré (si disponible) */
 public String getMachine();
 public void setMachine(String machine);

 /** Adresse IP de la machine sur laquelle cet objet a été généré (si disponible)
*/
 public String getIp();
 public void setIp(String ip);

 /** Répertoire de travail de la machine virtuelle Java sur laquelle cet objet
est exécuté */
 public String getWorkingDir();
 public void setWorkingDir(String workingDir);

 /** Classpath tel que renvoyé par la propriété java.class.path */
 public String getClasspath();
 public void setClasspath(String classpath);

 /** Chemin de la bibliothèque tel que renvoyé par la propriété java.library.path
*/
 public String getLibpath();
 public void setLibpath(String libpath);

 /** Pour des agents, renvoie la classe Java contenant la méthode principale
appelée */
 public String getMainClass();
 public void setMainClass(String mainClass);
```

```
 /** Version abrégée de la ligne de commande (à des fins de représentation, car
elle peut être imprécise) */
 public String getCommandLine();

 /** Champ transitoire pour assurer le suivi du moment de l'envoi ou de la réception
de cet objet */
 public Date getGenerationTime() ;
 public void setGenerationTime(Date time);

 /** Champ transitoire pour assurer le suivi d'un mot de passe requis pour appeler
les API sur cet agent via JMS */
 public String getToken();
 public void setToken(String token);
```

Nous pouvons maintenant consulter certains des API directement dans **com.itko.lisa.remote.client.AgentClient**. Cette liste n'est pas exhaustive, mais elle couvre une grande partie des besoins de la plupart des clients.

```
/** Obtient la classe responsable de toutes les détections et informations pour
un agent donné */
 public DiscoveryClient getDiscoveryClient();

 /** Obtient la classe responsable de toutes les opérations liées aux transactions
*/
 public TransactionsClient getTransactionClient();

 /** Obtient la classe responsable de toutes les opérations liées au VSE */
 public VSEClient getVSEClient();

 /**
  * Obtient tous les agents détectés - il s'agir de l'API principale pour obtenir
les IAgentInfos utilisées dans tous les autres appels d'API
  * @param tokens carte de guid ou noms d'agents sur des jetons, valeur nulle si
aucun agent ne dispose de la sécurité activée par jeton
  * @return renvoie un ensemble d'objets IAgentInfo représentant les agents
actuellement en ligne.
  */
  public Set getRemoteAgentInfos(Map tokens);

 /**
  * Transmet les informations des agents en ligne aux écouteurs enregistrés.
  * @param indique à l'agent qui vient d'être détecté de se connecter.
  */
  public void onAgentOnline(IAgentInfo info);

 /**
  * Transmet les informations des agents hors ligne aux écouteurs enregistrés.
  * @param indique à l'agent qui vient d'être détecté de se déconnecter.
  */
  public void onAgentOffline(IAgentInfo info);
```

```
 /**
  * Evalue le code arbitraire sur l'agent spécifié.
  * @param indique à l'agent que ce code sera évalué.
  * @param saisit le code à exécuter. La variable '_agent' représente l'instance
de l'agent.
  * @renvoie la valeur renvoyée par le code.
  * @renvoie JMSInvocationExceptionthrown si le code est renvoyé sur l'agent.
  */
 public Object eval(IAgentInfo info, String input) throws JMSInvocationException
```

API de détection d'agents

La classe **com.itko.lisa.remote.client.DiscoveryClient** fournit des méthodes liées à la détection de données sur un agent. Vous pouvez obtenir la classe **DiscoveryClient** à l'aide de l'appel suivant : **AgentClient.getInstance().getDiscoveryClient()**.

Remarque : Vous pouvez consulter des informations détaillées sur cette classe dans le document JavaDocs pour l'agent. Les documents JavaDocs sont disponibles dans le répertoire **LISA_HOME\doc**.

```
/**
  * Propriétés système de l'agent spécifié
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Map getVMProperties(IAgentInfo info) throws JMSInvocationException;

 /**
  * Renvoie une liste de StatsFrames enregistrés pour l'agent spécifié entre les
dates From (Du) et To (Au).
  * @param agentInfo l'agent qui requiert la récupération des statistiques
  * @param from      date de début du filtre
  * @param to        date de fin du filtre
  * @return          renvoie le liste des StatsFrames requis, classés par date
décroissante (commençant à partir de la date To (Au))
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to);

 /**
  * Renvoie une liste de StatsFrames enregistrés pour l'agent spécifié entre les
dates From (Du) et To (Au).
  * @param agentInfo l'agent qui requiert la récupération des statistiques
  * @param from      date de début du filtre
  * @param to        date de fin du filtre
  * @param interval   méthode de cumul des résultats en secondes. 10 signifie la
moyenne des résultats toutes les 10 secondes, etc...
  * @param limit     nombre maximum de résultats
  * @return          renvoie le liste des StatsFrames requis, classés par date
décroissante (commençant à partir de la date To (Au))
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to, int
interval, int limit);

 /**
  * TODO : implémenter de nouveau - actuellement renvoie
  * @param info
  * @return
  * @throws JMSInvocationException
  */
```

```
 public Topology getTopology(IAgentInfo info) throws JMSInvocationException;

 /**
  * Les points de sortie sont des MethodInfo qui capturent des classes/méthodes
qui effectuent des appels réseau au bas de la pile
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Set getExitPoints(IAgentInfo info) throws JMSInvocationException;

 /**
  * Renvoie le nom du conteneur J2EE (ou Java s'il ne s'agit pas d'un conteneur
J2EE)
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public String getServerInfo(IAgentInfo info) throws JMSInvocationException;

 /**
  * Renvoie les applications Web déployées dans le conteneur J2EE spécifié
  * @param info
  * @return
  */
 public WebApplication[] getWebApps(IAgentInfo info) throws
JMSInvocationException;

 /**
  * Renvoie la hiérarchie JNDI sur l'agent spécifié représenté par une
arborescence ClassNode (noeud de classes)
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getJNDIRoot(IAgentInfo info) throws JMSInvocationException;

 /**
  * Threads actuels sur la machine virtuelle de l'agent
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ThreadInfo[] getThreadInfos(IAgentInfo info) throws
JMSInvocationException;

 /**
  * Piles de threads actuels sur la machine virtuelle de l'agent
  * @param info
```

```
     * @return
     */
    public String[] dumpThreads(IAgentInfo info) throws JMSInvocationException;

    /**
     * Ensemble de tous les fichiers dans le classpath de l'agent spécifié
     * @param info
     * @return
     * @throws JMSInvocationException
     */
    public Set getClasspath(IAgentInfo info) throws JMSInvocationException;

    /**
     * Renvoie la hiérarchie de classes sous le chemin spécifié
     * @param info
     * @param fromPath
     * @return
     */
    public ClassNode getClassNodes(IAgentInfo info, String fromPath) throws
JMSInvocationException;

    /**
     * Hiérarchie de classes détectée dans l'archive à l'URL indiquée
     * @param info
     * @param url
     * @return
     * @throws JMSInvocationException
     */
    public ClassNode getArchiveNodes(IAgentInfo info, URL url) throws
JMSInvocationException;

    /**
     * Ensemble contenant des données sur la classe (fields/methods/src)
     * @param info
     * @param className
     * @return
     * @throws JMSInvocationException
     */
    public Set getClassInfo(IAgentInfo info, String className) throws
JMSInvocationException;

    /**
     * Décompile et renvoie la source vers une classe
     * @param info
     * @param clazz
     * @param loc décompile sur le client ou l'agent
     * @return
     * @throws JMSInvocationException
     */
```

```
 public String getClassSrc(IAgentInfo info, String clazz, boolean loc) throws
JMSInvocationException, IOException;

 /**
  * Renvoie la hiérarchie à laquelle cette classe appartient, à savoir, tous les
ancêtres, mais également tous les responsables d'extension/implémentation
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode[] getClassHierarchy(IAgentInfo info, String className) throws
JMSInvocationException;

/**
  * Renvoie (référence) tous les objets dans le segment de mémoire de la classe
spécifiée - utiliser avec précaution
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getInstancesView(IAgentInfo info, String className) throws
JMSInvocationException;

 /**
  * Renvoie (référence) tous les objets sur le segment de mémoire suivi par
l'agent.
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getTrackedObjects(IAgentInfo info) throws
JMSInvocationException;

 /**
  * Une représentation graphique rudimentaire d'un objet (champs calculés de façon
récursive)
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getObjectGraph(IAgentInfo info, String clazz, int hashCode)
throws JMSInvocationException;

 /**
  * Obtient le chemin d'un objet vers une racine GC
```

```
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getRootPath(IAgentInfo info, String clazz, int hashCode) throws
JMSInvocationException;

 /**
  * Obtient un fichier sur le système de fichiers de l'agent, le télécharge vers
le client à un emplacement temporaire et lui renvoie un descripteur
  * @param info
  * @param file
  * @return
  * @throws JMSInvocationException
  */
 public File getFile(IAgentInfo info, String file) throws
JMSInvocationException, IOException;
```

API de transaction d'agent

Remarque : Vous pouvez consulter des informations détaillées sur ces classes dans le document JavaDocs pour l'agent. Les documents JavaDocs sont disponibles dans le répertoire LISA_HOME\doc.

Une transaction est un chemin de code exécuté par un ou plusieurs serveurs comme résultat d'une demande client. Une transaction est représentée par une arborescence associée à une racine du client initialisant la demande. Les noeuds de l'arborescence sont des objets **com.itko.lisa.remote.transactions.TransactionFrame.** Ces objets contiennent des informations concernant une classe, une méthode et des arguments appelés pendant le traitement du serveur. Les trames contiennent également des informations complémentaires, telles que la durée, l'heure de l'exécution et le thread dans lequel elle a eu lieu.

Une transaction peut être assimilée à une pile d'appel de méthode. L'une des différences réside dans le fait que les transactions traversent les limites des thread, des processus, ou même des ordinateurs. Autre différence, la pile contient toutes les méthodes impliquées dans l'exécution de code d'un thread, tandis que les transactions omettent certains niveaux et contiennent des trames uniquement pour des méthodes d'intérêt de leur choix. Ces méthodes sont également appelées *méthodes interceptées*.

Le principal moyen pour obtenir et utiliser des objets **TransactionFrame** s'effectue via des surcharges des API suivantes fournies par **com.itko.lisa.remote.client.TransactionsClient**, qui à leur tour peuvent être obtenues avec l'appel suivant : **AgentClient.getInstance().getTransactionsClient()**.

```
/**
  * Démarre l'enregistrement de transactions
  * @param indique à l'agent de démarrer l'enregistrement.
  */
 public void startXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * Arrête l'enregistrement de transactions
  * @param indique à l'agent d'arrêter l'enregistrement.
  */
 public void stopXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * Lance le suivi de l'utilisation de socket sur le client à utiliser pour le
rapprochement des transactions client et serveur.
  * Cet appel doit être effectué avant l'appel que nous ajoutons dans
addClientTransaction.
  * @param global installation sur tous les sockets ou uniquement sur ce thread
  */
 public void installSocketTracker(boolean global);

 /**
```

```
  * Arrête le suivi de l'utilisation de socket sur le client à utiliser pour le
rapprochement des transactions client et serveur.
  * Cet appel doit être effectué après l'appel que nous ajoutons dans
addClientTransaction.
  * @param global désinstallation sur tous les sockets ou uniquement sur ce thread
  */
 public void uninstallSocketTracker(boolean global);

 /**
  * En tant que client, vous pouvez appeler cette méthode pour associer une
arborescence de transactions à une nouvelle transaction cliente créée à l'aide
  * des paramètres spécifiés.
  * Remarque : Vous devez appeler installSocketTracker avant d'initialiser la
transaction côté client que vous ajoutez ici.
  * De même, il est recommandé d'appeler uninstallSocketTracker après avoir
terminé l'appel réseau.
  * @param stamp     chaîne unique que vous pouvez utiliser ultérieurement pour
identifier la transaction racine
  *      (dans des appels getTransactions par exemple)
  * @param stepInfo  chaîne lisible et conviviale qui indique l'action de la
transaction (peut être nulle)
  * * @param args     paramètres transmis par le client à la transaction (peut
être vide)
  * @param result    résultat de la transaction (peut être nul)
  * @param duration  vue du client de la durée de la transaction
  */
 public void addClientTransaction(String  stamp, String stepInfo, Object[] args,
Object result, long duration);

 /** Supprime toutes les transactions et les données dépendantes générées par ce
client. */
 public void clearTransactions();

 /**
  * Obtient une liste non hiérarchique de TransactionFrames reçues à partir de
tous les agents qui satisfont les critères de filtrage transmis en tant
qu'arguments.
  * @param offset    décalage
  * @param limit     nombre maximum de résultats
  * @param category  filtre par catégorie de transaction (consultez
TranactionFrame.CATEGORY_XXX - 0 pour aucun filtre)
  * @param clazz     filtre par nom de classe (nul ou "" pour aucun filtre)
  * @param method    filtre par nom de méthode (nul ou "" pour aucun filtre)
  * @param minTime   filtre par durée de trame supérieure à minTime
  * @return        liste de trames TransactionFrames satisfaisant les critères
fournis et classées par heure décroissante
  */
 public List getTransactions(int offset, int limit, int category, String clazz,
String method, int minTime);
```

```
 /**
  * Obtient une liste d'arborescences de transactions associées aux transactions
spécifiées et satisfaisant les critères spécifiés
  * @param offset    décalage
  * @param limit     nombre maximum de résultats
  * @param stamps    tableau de transactions de client racines - renvoie all si
cet élément est vide.
  * @param minTime   durée en dessous de laquelle les transactions sont nettoyées
des résultats
  * @return ret      liste d'arborescences de transactions correspondant aux
critères et classées par heure décroissante
  */
 public List getTransactionsTree(int offset, int limit, String[] stamps, int
minTime)
```

Les paramètres vers ces API ne spécifient aucun **Agent** ni **AgentInfo**, car les transactions peuvent couvrir plusieurs agents.

Ces API renvoient des listes **com.itko.lisa.remote.transactions.TransactionFrame** (ou leurs arborescences) ; observez les données qu'elles contiennent :

```
/** Identificateur unique de cette trame */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** ID de la trame parente de cette trame
 public String getParentId();
 public void setParentId(String parentId);

 /** Identificateur partagé par toutes les trames appartenant à la même
transaction (identique à l'ID de trame racine global) */
 public String getTransactionId();
 public void setTransactionId(String transactionId);

 /** Objet TransactionFrame parent */
 public TransactionFrame getParent();
 public void setParent(TransactionFrame parent);

 /** Liste d'objets TransactionFrame enfants */
 public List getChildren();
 public void setChildren(List children);

 /** Identificateur unique de l'agent dans lequel cette trame a été enregistrée
*/
 public long getAgentGuid();
 public void setAgentGuid(long agentGuid);

 /** Compteur croissant qui permet de classer les trames (l'heure peut ne pas être
très précise) */
```

```
 public long getOrdinal();
 public void setOrdinal(long ordinal);

 /** Trames réseau entrantes ou sortantes -définissez ce paramètre pour indiquer
l'adresse IP à partir de laquelle elles communiquent */
 public String getLocalIP();
 public void setLocalIP(String ip);

 /** Trames réseau entrantes ou sortantes. Définissez ce paramètre pour indiquer
le port à partir duquel elles communiquent. */
 public int getLocalPort();
 public void setLocalPort(int port);

 /** Trames réseau entrantes ou sortantes -définissez ce paramètre pour indiquer
l'adresse IP avec laquelle elles communiquent */
 public String getRemoteIP();
 public void setRemoteIP(String ip);

 /** Trames réseau entrantes ou sortantes -définissez ce paramètre pour indiquer
le port avec lequel elles communiquent */
 public int getRemotePort();
 public void setRemotePort(int port);

 /** Nom du thread dans lequel cette trame a été enregistrée */
 public String getThreadName();
 public void setThreadName(String threadName);

 /** Nom de la classe ou de l'interface dans laquelle cette trame a été enregistrée
(en fonction de la spécification d'interception). */
 public String getClassName();
 public void setClassName(String className);

 /** Nom de la classe de l'objet réel dans laquelle cette trame a été enregistrée
*/
 public String getActualClassName();

 /** Nom de la méthode dans laquelle cette trame a été enregistrée */
 public String getMethod();
 public void setMethod(String method);

 /** Signature de la méthode dans laquelle cette trame a été enregistrée */
 public String getSignature();
 public void setSignature(String signature);

 /** Représentation formatée de la méthode dans laquelle cette trame a été
enregistrée */
 public String getSource();
 public void setSource(Object source);
```

```
 /** Représentation formatée des arguments pour la méthode dans laquelle cette
trame a été enregistrée */
 public String[] getArguments();
 public void setArguments(Object[] arguments);

 /** Représentation formatée du résultat de la méthode dans laquelle cette trame
a été enregistrée */
 public String getResult();
 public void setResult(Object result);

 /** Nombre de duplications de cette trame dans son parent */
 public long getHits();
 public void setHits(long hits);

 /** Heure du serveur lors de l'enregistrement de la trame */
 public long getTime();
 public void setTime(long time);

 /** Durée horloge d'exécution de cette trame */
 public long getClockDuration();
 public void setClockDuration(long clockDuration);

 /** Durée en termes d'UC d'exécution de cette trame */
 public long getCpuDuration();
 public void setCpuDuration(long cpuDuration);

 /**Représentation personnalisée de l'état associé à cette trame */
 public String getState();
 public void setState(Object state);

 /** Informations LEK formatées comme codées/décodées par la classe LEKEncoder
*/
 public String getLekInfo();
 public void setLekInfo(String lekInfo);

 /** Catégorie précalculée à laquelle cette trame appartient - consultez
TransactionFrame.CATEGORY_XXX */
 public int getCategory();
 public void setCategory(int category);

 /** Combinaison associée par une expression OU au niveau du bit de divers
informations internes - consultez TransactionFrame.FLAG_XXX */
 public long getFlags();
 public void setFlags(long flags);
```

API VSE de l'agent

Le VSE permet de mettre fin à des processus, des services, en totalité ou en partie, le long de périmètres bien définis. Les éléments internes de ces processus et ces services peuvent être remplacés par une couche exécutée par DevTest en fonction de règles personnalisées définies par l'utilisateur. Ces couches exécutées par DevTest sont appelées modèle. En général, le point de départ par défaut pour ces règles est obtenu à partir d'un enregistrement d'interactions de système dynamiques.

DevTest prend déjà en charge le VSE pour le protocole HTTP (permettant la virtualisation d'applications Web et de services Web), JMS et JDBC dans une certaine mesure. L'agent fournit des API pour activer la virtualisation directement à partir des processus de serveur, ce qui le rend indépendant par rapport aux protocoles. Vous pouvez activer la virtualisation pour HTTP, JMS et JDBC, mais également pour RMI, EJB ou tout objet Java personnalisé.

DevTest (ou un autre client de l'agent) peut réaliser cette virtualisation à l'aide des API suivantes définies dans **com.itko.lisa.remote.client.VSEClient**, telles qu'obtenues par la méthode **AgentClient.getInstance().getVSEClient()**.

Remarque : Vous pouvez consulter des informations détaillées sur cette classe dans le document JavaDocs pour l'agent. Les documents JavaDocs sont disponibles dans le répertoire **LISA_HOME\doc**.

```
/**
  * Renvoie une liste de tous les noms de classe/d'interface dont le nom correspond
à l'expression régulière fournie
  * ou étend/implémente une classe/interface dont le nom correspond à l'expression
régulière fournie
  * si l'implémentation est définie sur true. La recherche d'annotations est prise
en charge via la syntaxe suivante :
  * class regex@annotation regex (e.g. ".*.Remote@.*.Stateless").
  * @param agentInfo
  * @param regex
  * @param impl
  * @return
  */
 public String[] getMatchingClasses(IAgentInfo info, String regex, boolean impl)
throws JMSInvocationException

 /**
  * Enregistre un rappel du VSE avec l'agent spécifié dont la méthode onFrameRecord
sera appelée
  * en mode d'enregistrement pour toutes les méthodes virtualisées et dont la
méthode onFramePlayback sera appelée.
  * en mode de lecture pour toutes les méthodes virtualisées.
  * @param info
  * @param callback
  */
 public void registerVSECallback(IAgentInfo info, IVSECallback callback);
```

```
 /**
  * Annule l'enregistrement d'un rappel de VSE avec l'agent spécifié.
  * @param info
  * @param callback
  */
 public void unregisterVSECallback(IAgentInfo info, IVSECallback callback);

 /**
  * Lance l'appel à notre rappel d'enregistrement de virtualisation sur l'agent
spécifié.
  * @param agentInfo
  * @throws RemoteException
  */
 public void startVSERecording(IAgentInfo agentInfo) throws
JMSInvocationException

 /**
  * Lance l'appel à notre rappel de lecture de virtualisation sur l'agent spécifié.
  * @param agentInfo
  * @throws RemoteException
  */
 public void startVSEPlayback(IAgentInfo agentInfo) throws
JMSInvocationException

 /**
  * Arrête la virtualisation sur l'agent spécifié.
  * @param agentInfo
  * @throws RemoteException
  */
 public void stopVSE(IAgentInfo agentInfo) throws JMSInvocationException

 /**
  * Virtualise la classe/l'interface spécifiée et tous ses descendants sur l'agent
spécifié.
  * @param agentInfo
  * @param className
  * @return
  */
 public void virtualize(IAgentInfo agentInfo, String className) throws
JMSInvocationException
```

L'interface pour les API de rappel est définie par **com.itko.lisa.remote.vse.IVSECallback** et définit les méthodes suivantes :

```
/**
  * Il s'agit de la méthode appelée par des agents dont l'enregistrement de VSE
est activé
  * lorsqu'une méthode de virtualisation est appelée. La trame VSE dispose de
toutes les informations requises
```

```
  * pour lire ultérieurement la méthode en mode de lecture.
  * @param frame
  * @throws RemoteException
  */
 void onFrameRecord(VSEFrame frame) throws RemoteException;

 /**
  * Il s'agit de la méthode appelée par des agents dont la lecture de VSE est
activée.
  * lorsqu'une méthode de virtualisation est appelée. La trame VSE dispose de
toutes les informations requises
  * pour correspondre à une trame enregistrée existante de sorte que son résultat
(et par arguments de référence)
  * puisse être défini de façon appropriée.
  * @param frame
  * @return
  * @throws RemoteException
  */
 VSEFrame onFramePlayback(VSEFrame frame) throws RemoteException;
```

Finalement, l'objet **com.itko.lisa.remote.vse.VSEFrame** est un objet POJO avec des accesseurs get et set pour les propriétés suivantes :

```
/** Obtient/définie un identificateur unique pour cette trame */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** ID d'agent dont est issue cette trame */
 public long getAgentGuid();
 public void setAgentGuid(long agentId);

 /** Nom du thread sur lequel cette méthode de trame a été invoquée */
 public String getThreadName();
 public void setThreadName(String threadName);

 /** Nom de la classe sur laquelle cette méthode de trame a été invoquée. */
 public String getClassName();
 public void setClassName(String className);

 /**Identificateur unique qui effectue un suivi des objets pour la durée de la
vie de la machine virtuelle */
 public String getSourceId();
 public void setSourceId(String srcId);

 /** ID de session la plus profonde du protocole limité par session qui englobe
cette trame */
 public String getSessionId();
 public void setSessionId(String sessionId);

 /** Nom de la méthode qui a été appelée */
```

```
 public String getMethod();
 public void setMethod(String method);

 /** Tableau d'arguments convertis au format XStream pour la méthode qui a été
appelée. */
 public String[] getArgumentsXML();
 public void setArgumentsXML(String[] argumentsXML);

 /** Résultat converti au format XStream de la méthode qui a été appelée. */
 public String getResultXML();
 public void setResultXML(String resultXML);

 /** Heure (serveur) d'appel de la méthode */
 public long getTime();
 public void setTime(long time);

 /** Durée d'exécution de la méthode */
 public long getClockDuration();
 public void setClockDuration(long duration);

 /** Indique d'utiliser getCode ou ResultXML pour calculer le résultat attendu
en mode de lecture. */
 public boolean isUseCode();
 public void setUseCode(boolean useCode);

 /**
  * Code à exécuter sur le serveur si isUseCode est défini sur True. Il peut s'agir
d'un code arbitraire.
  * ayant accès à l'objet ($0) et aux arguments de la méthode ($1, $2...)
  */
 public String getCode();
 public void setCode(String code);
```

Exemples d'API d'agent

Exemple 1 d'API d'agent

Le code suivant génère une transaction à partir du code client et récupère l'arborescence de transactions générée :

```
private static void testAddTransaction() throws Exception
 {
 String request =
"http://localhost:8080/examples/servlets/servlet/HelloWorldExample";
 TransactionsClient tc = AgentClient.getInstance().getTransactionClient();

 tc.installSocketTracker(false);
 long start = System.currentTimeMillis();

 String response = testMakeRequest(request);

 long end = System.currentTimeMillis();
 tc.uninstallSocketTracker(false);

 String frameId = tc.addClientTransaction("test", new Object[] { request },
response, end - start);
 TransactionFrame frame = tc.getTransactionTree(frameId);

 System.out.println(frame.isComplete() ? "Yes!" : "No!");
 }
```

Dans cet exemple de code, la fonction d'utilitaire suivante est utilisée, sans lien avec l'agent, mais répertoriée à des fins d'exhaustivité :

```
/** Supposons que Tomcat est en cours d'exécution sur l'hôte local :8080 */
 private static String testMakeRequest(String url) throws IOException
 {
 StringBuffer response = new StringBuffer();
 HttpURLConnection con = (HttpURLConnection) new URL(url).openConnection();

 con.setRequestMethod("GET");
 con.setDoOutput(true);
 con.setUseCaches(false);

 BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));
 for (String line = br.readLine(); line != null; line = br.readLine())
response.append(line);
 br.close();

 return response.toString();
 }
```

Exemple 2 d'API d'agent

Le code suivant enregistre un rappel de VSE de journalisation :

```
AgentClient.getInstance().addListener(new IAgentEventListener()
 {
  public void onAgentOffline(final IAgentInfo info) {}
  public void onAgentOnline(final IAgentInfo info)
  {
   AgentClient.getInstance().getVSEClient().registerVSECallback(info, new
IVSECallback()
   {
    public void onFrameRecord(VSEFrame frame) { System.out.println("Recorded: "
+ frame); }
    public VSEFrame onFramePlayback(VSEFrame frame) {
System.out.println("Played back: " + frame); return frame; }
    public int hashCode() { return 0; }
    public boolean equals(Object o) { return o instanceof IVSECallback; }
  });

  try { AgentClient.getInstance().getVSEClient().startVSERecording(info); }
catch (JMSInvocationException e) {}
 });
```

Extensions d'agent Java

Vous pouvez créer les types d'extension suivants pour l'agent Java pour DevTest :

- Extensions d'agent (page 71)
- Extension de l'intermédiaire (page 77)
- Extensions Java de VSE (page 79)

Extension de l'agent

Pour personnaliser le comportement de l'agent, vous pouvez écrire des classes Java qui implémentent l'interface **com.itko.lisa.remote.transactions.interceptors.IInterceptor** ou étendent la classe **com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor.** Pour de plus amples informations, reportez-vous au document JavaDocs disponible dans le dossier **doc** du répertoire d'installation.

```
public interface IInterceptor {
    /**
     * Indique si cet intercepteur personnalisé est actuellement désactivé
     */
    public boolean isDisabled();

    /**
     * Indique si l'interception doit être renvoyée (true) ou de continuer (false)
     */
    public boolean block(boolean wayIn, Object src, String spec, String clazz,
String method, String signature, Object[] args, Object ret);

    /**
     * Appelé après l'entrée de la méthode pour éventuellement modifier la trame
actuelle en fonction de la logique d'intercepteur
     */
    public boolean preProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);

    /**
     * Appelé avant la sortie de la méthode pour éventuellement modifier la trame
actuelle en fonction de la logique d'intercepteur
     */
    public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);
}
```

Si vous voulez arrêter la capture de données, écrasez la méthode **block().** Cette technique est similaire à l'ajout d'une directive **exclude** au fichier **rules.xml**, mais offre plus de flexibilité.

Si vous voulez que l'agent effectue la logique immédiatement avant la capture d'une méthode, écrasez la méthode **preProcess().**

Si vous voulez que l'agent applique la logique immédiatement après la capture d'une méthode, écrasez la méthode **postProcess().**

Ces méthodes sont automatiquement appelées pour toutes les classes et méthodes interceptées ou suivies. Pour intercepter une méthode ou suivre une classe, vous pouvez la spécifier à l'aide de la syntaxe mentionnée dans le fichier **rules.xml.** Vous pouvez également spécifier l'interception à l'aide d'un programme dans le constructeur de la classe d'extension. L'avantage de la dernière approche réside dans le fait que l'extension est indépendante.

Le premier argument de la méthode **block()** est **boolean wayIn**. La méthode **block()** est appelée deux fois par méthode, une fois à l'entrée et une fois à la sortie. Lorsque l'appel d'entrée est effectué, la valeur de l'argument **wayIn** est true. Lorsque l'appel de sortie est effectué, la valeur de l'argument **wayIn** est false.

Le tableau suivant décrit les arguments communs aux méthodes **block()**, **preProcess()** et **postProcess()** :

Argument	**Description**
Object src	L'objet pour lequel la méthode est appelée.
String spec	Nom de la classe ou l'interface spécifiée pour l'instrumentation de l'API.
String clazz	Nom de la classe qui définit la méthode interceptée.
String method	Nom de la méthode interceptée.
String signature	Signature (au format de machine virtuelle Java) de la méthode interceptée.
Object[] args	Arguments transférés à la méthode interceptée.
Object ret	Valeur de retour de la méthode interceptée. Lorsque l'argument wayIn est true, la valeur de retour est nulle.

La différence entre les arguments **src**, **spec** et **clazz** peut être expliquée par un exemple.

Supposez que vous avez les définitions d'interface et de classe suivantes :

```
public interface A {
     void m();
}

public class B implements A {
     void m() {}
}

public class C extends B {
}
```

Si les règles spécifient **intercept("A", "m", "*")** et le code appelle **C.m()**, les informations suivantes sont vraies :

- **spec** est A
- **clazz** est B
- **src** est une instance de C

Déploiement

Pour déployer une extension, compilez-la et mettez-la en package dans un fichier .jar avec un fichier manifeste contenant l'entrée suivante :

```
Agent-Extension: nom de la classe d'extension
```

Lorsque vous déplacez ce fichier .jar dans le répertoire JAR de l'agent, l'agent le récupère automatiquement. Si vous ajoutez le fichier après le démarrage de l'agent, un mécanisme de chargement à chaud permet d'assurer l'application du fichier. Si vous mettez à jour l'extension .jar lors de l'exécution de l'agent, les classes JAR seront rechargées de façon dynamique. Le rechargement dynamique simplifie et accélère le test de votre code d'extension sans redémarrer le serveur.

Les fichiers .jar d'extension sont chargés par un chargeur de classes qui peut consulter toutes les classes utilisées dans la classe de l'extension. Vous pouvez compiler votre source d'extension au niveau du **LisaAgent.jar** et tous les fichiers .jar du conteneur qui définissent des classes que vous voulez utiliser. Il n'est pas nécessaire d'utiliser la réflexion dans votre extension.

Exemples

Dans l'exemple suivant, la méthode **block()** est utilisée pour empêcher l'agent de capturer une méthode dont le nom de thread commence par **Event Sink Thread Pool** (Pool de threads du récepteur d'événements).

```
public class MyInterceptor extends AbstractInterceptor {

   /** renvoie True si cet appel ne doit pas être intercepté */
   public boolean block(boolean wayIn, Object src, String spec, String clazz,
String method, String signature, Object[] args, Object ret) {
      if (Thread.currentThread().getName().startsWith("Event Sink Thread
Pool")) {
         return true;
      }
      return super.block(wayIn, src, spec, clazz, method, signature, args, ret);
   }

   ...

}
```

Dans l'exemple suivant, la méthode **postProcess()** est utilisée pour capturer des données pour la ligne Response (Réponse) de la fenêtre Transactions. Cet exemple appelle la méthode **setResponse()** de la classe **com.itko.lisa.remote.transactions.TransactionFrame.** Pour de plus amples informations, reportez-vous au document JavaDocs disponible dans le dossier doc du répertoire d'installation.

```
public class MyInterceptor extends AbstractInterceptor {

   ...

   public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
      if (clazz.equals("com.itko.lisa.training.NotCaptured") &&
method.equals("doRequest")) {

         frame.setResponse((String) ret);
      }
      return super.postProcess(frame, src, spec, clazz, method, signature, args,
ret);
   }
```

L'exemple suivant illustre la procédure d'impression du contenu XML d'un objet WebMethods **com.wm.data.IData** lorsqu'il est appelé dans un flux du serveur d'intégration. L'agent prend déjà en charge WebMethods, une extension n'est donc pas utile.

```
package com.itko.lisa.ext;

 import java.io.ByteArrayOutputStream;
 import com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor;
 import com.itko.lisa.remote.transactions.TransactionFrame;
```

```
 import com.wm.app.b2b.server.ServiceManager;
 import com.wm.app.b2b.server.BaseService;
 import com.wm.data.IData;
 import com.wm.util.coder.IDataXMLCoder;

 public class IDataInterceptor extends AbstractInterceptor {

 public IDataInterceptor() {
     super.intercept("com.wm.app.b2b.server.ServiceManager", "invoke",
"(Lcom/wm/app/b2b/server/BaseService;Lcom/wm/data/IData;Z)Lcom/wm/data/IData;
");
 }

 public boolean preProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
     if (ServiceManager.class.getName().equals(clazz) &&
"invoke".equals(method)) {
         doCustomLogic((BaseService) args\[0\], (IData) args\[1\], false);
     }

     return super.preProcess(frame, src, spec, clazz, method, signature, args,
ret);
 }

 public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
     if (ServiceManager.class.getName().equals(clazz) &&
"invoke".equals(method)) {
         doCustomLogic((BaseService) args\[0\], (IData) ret, true);
     }

     return super.postProcess(frame, src, spec, clazz, method, signature, args,
ret);
 }

 private void doCustomLogic(BaseService flow, IData p, boolean output) {
     ByteArrayOutputStream baos = new ByteArrayOutputStream(63);

     try {
         new IDataXMLCoder().encode(baos, p);
     } catch (IOException e) {
         e.printStackTrace();
     }

     System.out.println("Flow: " + flow.getNSName());
     System.out.println((output ? "Output" : " Input") + "Pipeline: " + baos);

 }
 }
```

Pour créer cette extension vous-même, compilez ce code au niveau du fichier **LisaAgent.jar**, du fichier **wm-isclient.jar** et du fichier **wm-isserver.jar**, puis mettez en package JAR le fichier de classe avec un fichier manifeste contenant la ligne suivante :

```
Agent-Extension: com.itko.lisa.ext.IDataInterceptor
```

Ajout de balises à des trames de transaction

Vous pouvez associer une trame de transaction à des paires clé-valeur arbitraires. Les paires clé-valeur sont connues comme des *balises*.

Les objets **TransactionFrame** fournissent une méthode **setTag().** La méthode comprend deux arguments de chaîne : la clé et la valeur.

Le code suivant indique la procédure d'ajout d'une balise dans la méthode **postProcess()** d'une extension d'agent :

```
// somewhere in postProcess
frame.setTag("tagname", "tagvalue");
```

Par exemple, vous pouvez créer une extension qui saisit la valeur du paramètre de demande **username** lors de la connexion à un site Web et appelle **frame.setTag("username", value**).

Les balises sont stockées dans la table **FRAME_TAGS** de la base de données.

Dans la console CAI, vous pouvez appliquer un filtre par balises. Pour plus d'informations, reportez-vous à la rubrique *Utilisation de CA Continuous Application Insight*.

Extension de l'intermédiaire

Les extensions d'intermédiaire sont chargées et appelées par l'intermédiaire lorsqu'un agent envoie un fragment de transaction.

Les extensions d'intermédiaire permettent d'effectuer les opérations suivantes :

- Modification des données contenues dans des trames
- Ajout ou suppression de certaines trames
- Personnalisation de l'algorithme d'assemblage L'algorithme d'assemblage définit la procédure d'assemblage des fragments de transaction.

Pour étendre l'intermédiaire, implémentez l'interface **com.itko.lisa.remote.plumbing.IAssemblyExtension**. Cette interface définit la méthode **onTransactionReceived().** Pour de plus amples informations, reportez-vous au document JavaDocs disponible dans le dossier doc du répertoire d'installation.

```
public interface IAssemblyExtension {
     /**
      * Méthode appelée avant l'assemblage
      * @param frame Racine de transaction partielle envoyée par un agent
      * @return renvoie la valeur True pour omettre l'assemblage normal (si
l'extension veut la traiter).
      */
     public boolean onTransactionReceived(TransactionFrame frame);
 }
```

Pour déployer une extension d'intermédiaire, compilez l'extension et mettez-la en package dans un fichier .jar avec un fichier manifeste contenant l'entrée suivante :

```
Broker-Extension: nom de la classe d'extension
```

Lorsque vous déplacez ce fichier .jar dans le répertoire (registre) de l'intermédiaire, celui-ci le récupère automatiquement. Si vous ajoutez le fichier après le démarrage de l'intermédiaire (registre), un mécanisme de chargement à chaud permet d'assurer l'application du fichier. Si vous mettez à jour l'extension .jar lors de l'exécution de l'intermédiaire (registre), les classes JAR seront rechargées de façon dynamique. Le rechargement dynamique simplifie et accélère le test de votre code d'extension sans redémarrer l'intermédiaire (registre).

Exemple

Exemple d'utilisation habituelle : l'algorithme d'assemblage normal utilisant des TCP/IP et des ports est confondu par un équilibreur de charge placé entre des agents et une adresse IP virtuelle lui est affectée, ou des agents utilisent une bibliothèque native en tant qu'E/S et l'accès direct aux adresses IP et aux ports en cours d'utilisation n'est pas disponible. Dans ce cas, les champs d'adresse et de port des trames sont vides ou incorrects et les trames doivent être assemblées dans une extension :

```
import com.itko.lisa.remote.plumbing.IAssemblyExtension;
import com.itko.lisa.remote.transactions.TransactionFrame;
import com.itko.lisa.remote.utils.Log;
import com.itko.lisa.remote.utils.UUID;

public class LoadBalancerExtension implements IAssemblyExtension {

    private TransactionFrame m_lastAgent1Frame;
    private TransactionFrame m_lastAgent2Frame;

    public boolean onTransactionReceived(TransactionFrame frame) {

    if (frame.getClassName().equals("Class1") &&
frame.getMethod().equals("method1")) {
        m_lastAgent2Frame = frame;
        if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
{
            stitch(m_lastAgent1Frame, m_lastAgent2Frame);
        }
    }

    if (frame.getClassName().equals("Class2") &&
frame.getMethod().equals("method2")) {
        m_lastAgent1Frame = frame;
        if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
{
            stitch(m_lastAgent1Frame, m_lastAgent2Frame);
        }
    }

        return false;
    }

    private void stitch(TransactionFrame parent, TransactionFrame child) {
    String newFrameId = UUID.newUUID();
    parent.setFrameId(newFrameId);
    child.setParent(parent);
    parent.getChildren().add(child);
}
}
```

Extension Java de VSE

Dans une extension Java de VSE, vous pouvez écraser l'une des méthodes suivantes :

- **onPreRecord() :** cette méthode est appelée pendant l'enregistrement avant le démarrage de l'exécution de la méthode virtualisée.
- **onPostRecord() :** cette méthode est appelée pendant l'enregistrement après l'arrêt de l'exécution de la méthode virtualisée.
- **onPreHijack() :** cette méthode est appelée pendant la lecture avant le transfert de la méthode virtualisée vers le VSE.
- **onPostHijack() :** cette méthode est appelée pendant la lecture après le renvoi de la méthode virtualisée par le VSE.
- **onNewStream() :** cette méthode est appelée lors de chaque conversion d'un objet au format XML.

Exemple

Dans l'exemple suivant, la méthode **onPostRecord()** est utilisée pour changer le nom d'une classe pendant l'enregistrement. Dans cet exemple, la méthode **setClassName()** de la classe **com.itko.lisa.remote.vse.VSEFrame** est appelée. Pour de plus amples informations, reportez-vous au document JavaDocs disponible dans le dossier **doc** du répertoire d'installation.

```
public class MyInterceptor extends AbstractVSEInterceptor {

   ...

   public boolean onPostRecord(VSEFrame frame, Object src, String clazz, String
method, String signature, Object[] args, Object ret) {
      frame.setClassName(clazz.replaceAll("\\.", "_"));
      return super.onPostRecord(frame, src, clazz, method, signature, args,
ret);
   }

}
```

Equilibreurs de charge et serveurs Web natifs

Le processus consistant à intégrer des transactions partielles dans des transactions complètes est appelé *assemblage*.

Si des agents sont déployés sur un réseau sur lequel sont exécutés des équilibreurs de charge ou des serveurs Web natifs, il est possible que l'algorithme d'assemblage utilisé par CAI ne fonctionne pas correctement. Dans ce scénario, ajoutez la directive **loadbalancer** au fichier **rules.xml** de l'intermédiaire. Spécifiez l'adresse IP de chaque appliance installée entre des agents. Par exemple :

```
<loadbalancer ip="172.16.0.0"/>
<loadbalancer ip="172.31.255.255"/>
```

La directive **loadbalancer** doit être placée dans l'élément **broker.**

Une autre approche consiste à activer la propriété **Always Wait** (Toujours attendre).

Vous pouvez configurer cette propriété à partir de la fenêtre Agents du portail DevTest. La propriété s'affiche dans l'onglet Settings (Paramètres).

Sécurité de l'agent Java

L'agent Java pour DevTest fournit un mécanisme de sécurité pour les extensions et les appels de code distants.

L'agent installe son propre gestionnaire de sécurité. Tous les codes d'agent personnalisés sont exécutés en vertu d'autorisations spéciales appliquées par le gestionnaire de sécurité. Si l'application utilise déjà son propre gestionnaire de sécurité, le gestionnaire de sécurité de l'agent encapsule le gestionnaire de sécurité existant et le délègue pour le code d'application standard.

Extensions

Les extensions de l'agent pour CA Continuous Application Insight et le CA Service Virtualization sont exécutées avec les restrictions suivantes :

- L'accès au fichier est limité au répertoire d'agent et au répertoire temporaire.
- La création de processus est désactivée.
- La sortie de processus est désactivée.

En conséquence, le système testé est placé dans le bac à sable et séparé du reste de l'ordinateur.

Certaines applications peuvent explicitement vérifier que le gestionnaire de sécurité est nul et prennent un chemin de code différent selon le résultat. Dans ce cas, le gestionnaire de sécurité de l'agent peut entraîner des problèmes sans solution. Vous pouvez désactiver le gestionnaire de sécurité de l'agent à l'aide de l'une des méthodes suivantes :

- Dans la ligne de commande, spécifiez **-Dsecurity.manager.disabled=true**.
- Assurez que la propriété **Disable security manager** (Désactiver le gestionnaire de sécurité) est activée.

Vous pouvez configurer cette propriété à partir de la fenêtre Agents du portail DevTest. La propriété s'affiche dans l'onglet Settings (Paramètres).

Remarque : La désactivation du gestionnaire de sécurité désactive également les vérifications d'autorisation. Toutefois, vous pouvez encore activer l'authentification de jeton.

Appel de code distant

Tous les API d'agent utilisent l'appel de code distant, car l'agent se trouve dans un système distant.

La sécurité de l'appel de code distant est gérée à l'aide de jetons.

Utilisez l'option **token (jeton)** pour définir un ou deux jetons pour un agent. Le premier jeton représente un rôle d'administration. Le deuxième jeton représente un rôle d'utilisateur. Si vous spécifiez deux jetons, utilisez les deux-points pour les séparer.

Exemples :

- **token=asdf1** spécifie un jeton d'administrateur avec la valeur **asdf1**.
- **token=asdf1:asdf2** spécifie un jeton d'administrateur avec la valeur **asdf1** et un jeton d'utilisateur avec la valeur **asdf2**.

Le jeton d'administrateur ou d'utilisateur peut inclure tout type de caractères sauf la virgule, le signe égal et l'espace. La longueur maximum d'un jeton est de 16 caractères.

Vous pouvez utiliser le concept de jeton avec des consoles. Vous pouvez le spécifier avec la syntaxe de ligne de commande **-Dlisa.token=xxxx** ou **-Dlisa.token=xxxx:xxxx**. Dans le cas de consoles, il n'existe aucun gestionnaire de sécurité personnalisé, les deux jetons disposent donc de toutes les autorisations. La sécurité est réalisée, car le fait de ne pas indiquer de jeton de console empêche toute action.

Configuration de l'agent Java pour l'utilisation du protocole SSL

Vous pouvez utiliser le protocole SSL (Secure Sockets Layer) pour sécuriser les communications entre l'agent et l'intermédiaire.

L'intermédiaire hérite des paramètres que le registre contient pour le protocole SSL. L'agent utilise le même mot de passe chiffré et le même référentiel de clés.

Sélectionnez l'une des approches suivantes.

Référentiel de clés par défaut

Cette approche utilise le référentiel de clés par défaut de DevTest.

Procédez comme suit:

1. Ouvrez le fichier **local.properties** dans le répertoire **LISA_HOME** et supprimez la mise en commentaire de la ligne suivante :

   ```
   lisa.net.default.protocol=ssl
   ```

2. Enregistrez le fichier **local.properties**.
3. Démarrez ou redémarrez le registre.
4. Lorsque vous spécifiez l'URL de l'intermédiaire, utilisez le schéma **ssl.** Par exemple :

   ```
   ssl://localhost:2009
   ```

Référentiel de clés personnalisé

Cette approche repose sur un référentiel de clés personnalisé.

Remarque : Pour plus d'informations sur le portail DevTest, consultez la rubrique Utilisation de CA Continuous Application Insight.

Procédez comme suit:

1. Configurez un référentiel de clés personnalisé en suivant les instructions de la rubrique Utilisation du protocole SSL pour sécuriser la communication.
2. Placez le référentiel de clés dans le répertoire **LISA_HOME\agent.**
3. Ouvrez le portail DevTest.
4. Dans le volet de navigation gauche, sélectionnez Settings (Paramètres), Agents.
5. Dans la partie gauche, sélectionnez l'agent.
6. Cliquez sur l'onglet Settings (Paramètres).

7. Définissez les propriétés d'agent suivantes pour configurer un référentiel de clés personnalisé qui met en miroir les propriétés de DevTest. Vous pouvez copier les mots de passe chiffrés à partir du fichier **local.properties.**
   - Emplacement du référentiel de clés
   - Mot de passe du référentiel de clés (chiffré)
   - Emplacement du référentiel d'approbations
   - Mot de passe du référentiel d'approbations (chiffré)
8. Dans la partie gauche, sélectionnez l'intermédiaire.
9. Définissez les mêmes propriétés que pour l'agent. Le chemin d'accès au référentiel de clés sera différent.
10. Lorsque vous spécifiez l'URL de l'intermédiaire, utilisez le schéma **ssl.** Par exemple :

    ```
    ssl://localhost:2009
    ```

Fichiers journaux de l'agent Java

Vous pouvez utiliser les fichiers journaux suivants pour résoudre (page 86) des problèmes :

- Le fichier journal de l'agent est appelé **devtest_agent_pid.log**. Ce fichier est enregistré dans le même répertoire que le fichier **LisaAgent.jar.**
- Lors de chaque démarrage de l'intermédiaire, un fichier journal nommé **devtest_broker_pid.log** est créé. De plus, l'intermédiaire a un fichier journal consolidé nommé **pfbroker.log**. Ces fichiers sont écrits dans le même répertoire que les principaux fichiers journaux de DevTest.
- Les fichiers journaux de la console sont appelés **devtest_agent_console_pid.log**. La station de travail DevTest Workstation, le portail DevTest, les simulateurs et l'envrionnement VSE sont chacun associés à un fichier journal. Ces fichiers sont écrits dans le même répertoire que les principaux fichiers journaux de DevTest.

La partie **pid** de chaque nom de fichier représente l'identificateur du processus qui effectue la journalisation.

Remarque : Pour plus d'informations sur les principaux fichiers journaux de DevTest et leur emplacement, reportez-vous à la rubrique *Administration.*

Les propriétés de configuration suivantes vous permettent de contrôler la journalisation :

Java logging level (Niveau de journalisation Java)

Définit le niveau de journalisation au démarrage.

Maximum log size (Taille maximale du journal)

Définit la taille maximum d'un journal d'agent avant son basculement.

Maximum log archives (Nombre maximum d'archives de journal)

Définit le nombre maximum de journaux basculés archivés à conserver.

Journal de l'agent

Permet d'écraser l'emplacement par défaut.

Vous pouvez configurer ces propriétés dans la fenêtre Agents du portail DevTest. Les propriétés s'affichent dans l'onglet Settings (Paramètres).

Si vous devez configurer ces propriétés à partir du fichier **rules.xml,** les noms de propriété correspondants sont les suivants :

- **lisa.agent.java.logging.level**
- **lisa.agent.log.max.size**
- **lisa.agent.log.max.archives**
- **lisa.agent.agent.log**

Dépannage de l'agent Java

Important : Vous devez d'abord examiner l'environnement cible pour vous assurer qu'il a été testé, ou testez-le vous-même. La plupart des problèmes de l'agent Java sont liés au système d'exploitation ou à une machine virtuelle Java, plutôt qu'à une application. Veillez à suivre les instructions indiquées dans cette documentation. Si vous avez besoin d'informations complémentaires, cette rubrique aborde les problèmes les plus courants.

La plupart des problèmes graves impliquant l'agent Java pour DevTest surviennet lors du démarrage (par exemple, l'agent est introuvable, ou le processus tombe en panne ou se bloque). En règle générale, une fois ces problèmes réglés, les conditions sont correctes. A partir de là, il s'agit d'améliorer la configuration ou d'écrire des extensions.

Remarques :

- Il peut être difficile de relancer les serveurs pour tenter d'effectuer différentes actions dans l'environnement de l'agent. En général, il est plus simple et judicieux de tester l'exécution de **Java <options d'agent> -version** sur l'ordinateur cible et la machine virtuelle Java. Cet exercice indique si le problème est lié au SE/à la machine virtuelle Java ou au conteneur/à l'application.
- Pour obtenir un utilitaire de ligne de commande qui peut vous aider dans le processus d'installation, consultez la section Utilisation de l'assistant d'installation de l'agent (page 19).

Erreur lors du démarrage : Error occurred during initialization of VM. Could not find agent library in absolute path... (Une erreur s'est produite lors de l'initialisation de machine virtuelle. Bibliothèque de l'agent introuvable dans le chemin d'accès absolu..)

Vérifiez que la bibliothèque est disponible à l'emplacement et utilise la même architecture que Java (32 bits et 64 bits).

Vérifiez également qu'aucune dépendance ne manque dans la bibliothèque. Par exemple, utilisez **depends.exe** sur Win32, **otool -L** sur Mac OS X, ou **ldd -d** sur Linux et UNIX. Si des bibliothèques sont manquantes, vérifiez que la variable **LD_LIBRARY_PATH** est définie de sorte à inclure les répertoires dans lesquelles elles résident. Les conteneurs peuvent remplacer **LD_LIBRARY_PATH** dans leur script de démarrage. Le fait de définir la variable dans un shell plutôt que dans le script ou dans un outil d'administration spécifique au conteneur ne garantit pas qu'elle soit correctement définie.

Si **ldd** n'est pas correctement renvoyé, l'agent ne sera pas correctement exécuté. Par conséquent, la vérification du **ldd** est la première action à effectuer. Si une version de l'agent pour le système d'exploitation est introuvable, envisagez d'utiliser l'agent Java pur.

L'agent sort immédiatement, ou peu de temps après le démarrage du processus.

Si le message **LISA AGENT: VM terminated** (Agent pour LISA : machine virtuelle terminée) s'affiche, il est probable que le processus se soit terminé normalement. Plusieurs conteneurs disposent de processus de lanceur, il est donc normal qu'ils sortent rapidement.

Si ce message ne s'affiche pas ou qu'un vidage sur incident se produit (après le retour correct de **ldd**), un bogue de l'agent peut être justifié. Informez le service de support et tentez d'utiliser l'agent Java pur. Si une panne ou un vidage se produit encore, il peut s'agir d'un bogue de la machine virtuelle Java. Cela se produit avec IBM JVM 1.5 sur certains systèmes d'exploitation, dû à une tentative d'instrumentation de threads. Dans ce cas, essayez de fournir l'argument de machine virtuelle Java de ligne de commande **-Dlisa.debug=true**.

L'agent sort avec le message GetEnv on jvmdi returned -3 (JNI_EVERSION) (GetEnv sur jvmdi renvoyé -3 (JNI_EVERSION)).

Essayez de spécifier l'option de ligne de commande **-Xsov** pour indiquer à la machine virtuelle Java d'utiliser ses bibliothèques activées par débogage. Si cela ne fonctionne pas (par exemple, un message d'erreur d'option non valide s'affiche), ce système d'exploitation n'est pas pris en charge.

L'agent sort avec le message "UTF ERROR" [../../../src/solaris/instrument/EncodingSupport_md.c":66] : ...".

Le message peut varier selon la version exacte du système d'exploitation et/ou de la machine virtuelle Java. En général, le message comprend l'un des éléments suivants : UTF ERROR ["../../../src/solaris/instrument/EncodingSupport_md.c":66]: Failed to complete iconv_open() setup (Impossible de terminer l'installation de iconv_open()).

Ce problème est dû à un bogue de certaines machines virtuelles Java Solaris qui se produit lorsque certains packs linguistiques ne sont pas installés. Pour corriger ce problème, installez le pack linguistique en-US en exécutant l'installation de **pkg SUNWlang-enUS**, puis de **export LANG=en_US.UTF-8**. De même, vous pouvez essayer d'utiliser l'agent natif.

L'agent se bloque ou renvoie de nombreuses exceptions lors du démarrage (LinkageErrors, CircularityErrors, etc.).

L'instrumentation du code d'octet Java à l'aide de Java peut entraîner une légère altération de l'ordre de chargement de précédentes classes (par exemple, **java.***) et se terminer par un interblocage ou des erreurs de vérification de code d'octet.

Toutes les occurrences connues de ces problèmes ont été éliminées pour toutes les combinaisons de machines virtuelles Java et de systèmes d'exploitation. Toutefois, il est possible qu'une combinaison non testée se produise. Informez le service de support de ce problème. S'il s'agit d'un blocage, incluez un fil de discussion à votre problème transmis au service de support. Créez un fil de discussion en appuyant sur Ctrl+Pause sous Windows, CTRL+\ ou kill -3 <pid> sous UNIX/Linux. Vous pouvez également essayer l'agent Java, car l'ordre de chargement des classes est légèrement différent. Si une classe ou un package semble impliqué à chaque blocage de thread, essayez de lui ajouter une directive **exclude** dans le fichier **rules.xml**.

L'agent renvoie l'exception java.lang.VerifyErrors ou l'échange à chaud n'a aucun effet.

La prise en charge de l'échange à chaud de certaines versions de machines virtuelles Java ultérieures à 1.4 contient des bogues (instrumentation des classes après leur chargement).

Dans ce cas, désactivez l'échange à chaud en désactivant la propriété **Enable hot instrumentation** (Activer l'instrumentation à chaud).

Vous pouvez configurer cette propriété à partir de la fenêtre Agents du portail DevTest. La propriété s'affiche dans l'onglet Settings (Paramètres).

Vous devez déterminer à l'avance les classes et/ou les méthodes que vous voulez intercepter ou virtualiser, ainsi qu'ajouter les règles pour ces classes ou méthodes dans le fichier **rules.xml**, puis relancer le serveur. Ce processus est plus fastidieux que sur un serveur dynamique, mais il s'agit de la seule solution connue à ce problème.

Parfois, l'exception **java.lang.VerifyError** n'est pas visible dans les journaux, mais l'agent se comporte comme si la classe désignée n'était pas instrumentée ou renvoie des résultats aléatoires et peut même s'arrêter brutalement.

L'agent démarre, mais les consoles ou l'intermédiaire ne peuvent pas le détecter.

Généralement, cela est dû un problème lié à un pare-feu ou au port entre l'agent et l'intermédiaire.

La partie supérieure du journal de l'agent peut inclure l'avertissement suivant : **Can't connect to broker at tcp://ip:port** (Impossible de se connecter à l'intermédiaire sur tcp://ip:port). Vérifiez que l'adresse IP et le port que l'agent utilise sont corrects. Puis, vérifiez que l'intermédiaire écoute sur le port spécifié dans l'adresse IP spécifiée. **netstat -ano | grep port** doit afficher un port en écoute dans l'adresse IP, ou 0.0.0.0. Enfin, recherchez les problèmes liés au pare-feu en exécutant **telnet ip port** à partir de l'ordinateur de l'agent pour veiller à ce que l'intermédiaire puisse être détecté. Si c'est le cas, l'état de l'intermédiaire est susceptible d'être incorrect. Vérifiez le registre et les journaux de l'intermédiaire (et redémarrez-le si nécessaire).

L'agent entraîne l'expiration de certaines opérations.

Le comportement de certaines machines virtuelles Java (notamment les machines virtuelles Java IBM) est inadéquat après l'instrumentation de certaines de leurs classes de mise en réseau. En conséquence, les appels de mise en réseau peuvent échouer ou expirer sans raison apparente.

Pour éviter l'instrumentation de ces classes, tentez de fournir l'argument de machine virtuelle Java de ligne de commande **-Dlisa.debug=true**.

Si le problème n'est pas résolu, modifiez le fichier **rules.xml** pour exclure les packages de mise en réseau suivants :

```
<exclude class="java.net.**"/>
<exclude class="java.nio.**"/>
<exclude class="sun.nio.**"/>
```

java.lang.NoClassDefFoundError on com/itko/lisa/remote/transactions/TransactionDispatcher.class

Vérifiez que **LisaAgent.jar** est disponible, dispose de droits de lecture et qu'il n'est pas endommagé. Pour cela, la méthode la plus simple consiste à exécuter **java -jar LisaAgent.jar -v**.

Vérifiez également si l'application utilise OSGi. Si c'est le cas (comme pour JBoss 7), ajoutez **com.itko** aux packages système ou d'amorçage. La méthode d'ajout de **com.itko** dépend du conteneur, ce qui complique la fourniture d'instructions spécifiques. Toutefois, il s'agit généralement d'un fichier de configuration contenant une propriété qui spécifie une liste de packages ou un argument de machine virtuelle Java similaire.

Les exceptions liées la sécurité sont levées lorsque l'agent est activé.

Il est possible que les exceptions SecurityExceptions ou PermissionExceptions soient levées par l'application uniquement lorsque l'agent est activé avec la sécurité activée. Ce paramètre est désormais le paramètre par défaut.

La raison et la solution sont expliquées dans la section Sécurité de l'agent Java (page 81).

Utilisation de ressources anormale (UC, mémoire, descripteurs de fichier...)

Si l'utilisation de l'UC est anormalement élevée ou indique des pics réguliers, prenez note de la période des pics, car cela permettra de déterminer le ou les threads défectueux. Essayez également de désactiver CAI et le VSE.

Si vous obtenez des erreurs OutOfMemory, surveillez l'utilisation de segment de mémoire Java. Si elle dépasse la limite **-Xmx,** augmentez cette limite. Toutefois, évitez d'augmenter la limite si elle est déjà élevée et excessive par rapport à l'utilisation d'application normale sans l'agent. Dans ce cas, générez une image mémoire du segment de mémoire. Vous pouvez utiliser l'utilitaire WAS HeapDump pour WebSphere ou l'outil Eclipse MAT gratuit pour les versions antérieures. Si l'utilisation de la mémoire est inférieure à la limite de **-Xmx** au moment où l'erreur s'est produite, il s'agit probablement d'une fuite dans le code natif. Dans ce cas, informez le service de support.

Si vous obtenez des exceptions IOExceptions sans raison (par exemple, un nombre excessif de descripteurs de fichier), notamment sur UNIX ou Linux, vérifiez la valeur ulimit dans la zone (**ulimit -n -H** et **ulimit -n -S**). Si cette valeur est faible, demandez à l'administrateur de la zone de l'augmenter (une valeur inférieure à 4096 est considérée faible pour des applications J2EE modernes). N'oubliez pas de redémarrer ensuite.

L'agent tente de conserver le nombre de descripteurs de fichier sous cette limite en déclenchant un nettoyage de la mémoire lorsque la limite est proche. Si l'agent ne peut pas lire la limite lors du démarrage, il utilise une valeur par défaut de 1024. L'utilisation de cette valeur par défaut peut entraîner un nettoyage de la mémoire excessif et des pics d'UC semi-aléatoires et fréquents. Cette situation est décrite par les messages suivants dans les journaux :

```
Max (or preferred) handles limit approaching - triggering GC... (Limite de
descripteurs maximum (ou préférée) proche - déclenchement du nettoyage de la
mémoire...)
```

Dans ce cas, augmentez la valeur de la propriété **Maximum number of handles** (Nombre maximum descripteurs).

Vous pouvez configurer cette propriété à partir de la fenêtre Agents du portail DevTest. La propriété s'affiche dans l'onglet Settings (Paramètres).

Je peux voir que l'agent a démarré, mais les données de CAI sont manquantes ou incomplètes.

Le processus du cycle de vie de données comprend les étapes suivantes :

- Capture de transaction dans l'agent
- Transfert à l'intermédiaire
- Assemblage de transactions partielles dans l'intermédiaire
- Transfert aux consoles
- Conservation dans la base de données
- Récupération à partir de la base de données

Les données manquantes ou incomplètes peuvent être le résultat d'un problème de l'une de ces étapes.

Consultez le journal de l'agent et recherchez des exceptions de capture. Consultez les journaux de l'intermédiaire et de la console et recherchez des exceptions de transfert ou de persistance.

Si aucune exception n'est présente, mais que les données manquantes sont toujours introuvables, activez le débogage ou la journalisation de développement dans les agents. Si aucune instruction similaire à Sent partial transaction (Transaction partielle envoyée) n'y figure, CAI est probablement désactivé.

Si aucune de ces étapes ne fournit de résultats concluants, contactez le service de support.

J'ai activé l'**enregistrement ou la lecture** de Java pour le VSE, mais le VSE **ne reçoit aucune demande de l'agent.**

En premier lieu, vérifiez que l'agent est en mode d'enregistrement de lecture de VSE Ouvrez le journal d'agent et recherchez "Starting VSE record/playback...

Puis, recherchez des exceptions dans les journaux de l'agent et du VSE. S'ils sont corrects, il est possible que la classe que vous considériez virtualisée ne le soit pas. Recherchez une instruction similaire à Virtualized com.xxx... dans le journal d'agent. Si elle n'y figure pas, il est possible que l'application n'ait pas encore chargé la classe. Il est également possible que des versions antérieures à Java 1.4 ne prennent pas en charge l'échange à chaud. Le VSE Java utilise l'échange à chaud par défaut. Dans ce cas, spécifiez les classes virtualisées dans le fichier **rules.xml** de l'agent et redémarrez-le.

Autres problèmes liés au VSE Java

Si vous rencontrez des problèmes (utilisation de ressource fonctionnelle ou anormale) lors de l'utilisation de VSE Java, désactivez CAI côté agent.

Vous pouvez désactiver CAI en désactivant la propriété **Auto-start** (Démarrage automatique). Puis, redémarrez ensuite la machine virtuelle Java.

Vous pouvez configurer cette propriété à partir de la fenêtre Agents du portail DevTest. La propriété s'affiche dans l'onglet Settings (Paramètres).

Ne désactivez pas CAI côté intermédiaire, sans quoi le VSE Java arrêtera de fonctionner complètement.

La fonctionnalité de scénario ne fonctionne pas.

En premier lieu, vérifiez que l'agent est connecté à un intermédiaire.

Si c'est le cas, examinez la source HTML de la page problématique. Vérifiez que le bas de la page contient un bloc JavaScript facilement reconnaissable aux noms de variables utilisés (par exemple : **com_itko_pathfinder_defectcapture_xxx**). Si ce bloc est manquant, examinez le journal d'agent et recherchez d'éventuelles exceptions. L'absence du bloc peut également être expliquée par l'une des raisons suivantes :

- La page HTML est inhabituelle. Par exemple, des balises HTML sont manquantes.
- L'agent rencontre des problèmes pour le capturer complètement, ce qui peut se produire avec des conteneurs non testés ou des pages statiques.

Si le bloc est présent, vérifiez si un serveur Web natif (par exemple, Apache) ou un équilibreur de charge est présent en face du conteneur Java. Si c'est le cas, configurez-le de sorte à envoyer la demande de JavaScript CAI et fichiers de ressources au conteneur Java. Ces fichiers contiendront le terme **defectcapture** dans leur URL. En général, les administrateurs informatiques connaissent la procédure de cette tâche.

DevTest est configuré pour utiliser une base de données DB2 IBM. Dans le portail DevTest, j'ai sélectionné un noeud JDBC dans un graphique de chemin. Les colonnes Statements et Connection Url sont vides.

Désactivez la diffusion en continu progressive en ajoutant la chaîne **progressiveStreaming=2** à l'URL de connexion JDBC. Par exemple :

```
lisadb.pool.common.url=jdbc:db2://myhostname:50000/dbname:progressiveStreaming=2;
```

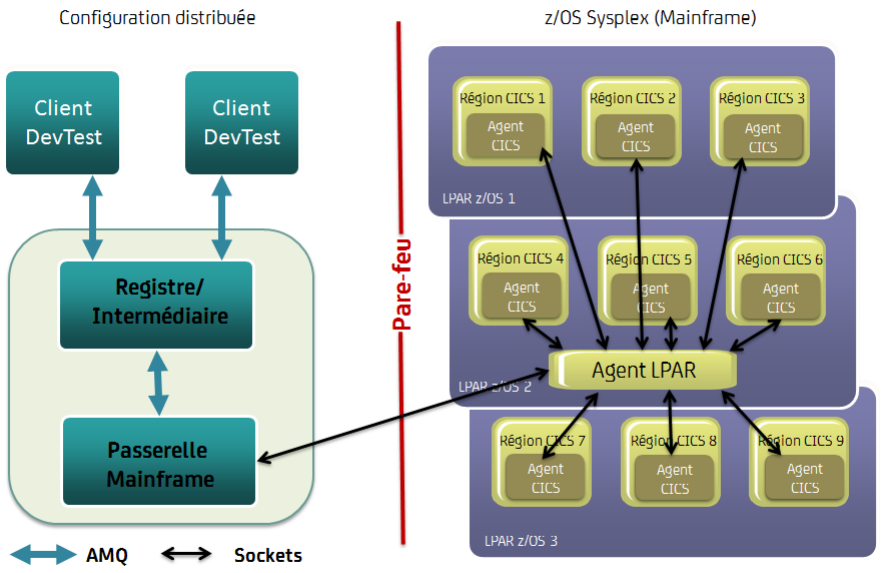
Chapitre 2: Passerelle Mainframe

La passerelle Mainframe est un composant qui permet aux clients DevTest de communiquer avec des agents Mainframe DevTest.

Ce chapitre traite des sujets suivants :

Architecture de la passerelle Mainframe

Le diagramme suivant illustre l'interaction de la passerelle Mainframe avec d'autres composants.



Une fois la connexion établie, le trafic part des clients DevTest vers l'intermédiaire, vers la passerelle Mainframe, vers le système Mainframe réel. Le trafic peut également partir dans l'ordre inverse.

Règles de la passerelle Mainframe

Vous devez configurer les propriétés d'agent suivantes pour la passerelle Mainframe :

Code page (page de code)

Page de code utilisée pour la conversion entre ASCII et EBCDIC

La valeur par défaut est Cp1047.

Initial packet buffer size (Taille initiale de tampon de paquet)

Taille du tampon initial pour la gestion des paquets à partir du Mainframe, exprimée en octets. La taille de tampon augmente si des paquets de taille supérieure sont reçus.

La valeur par défaut est 1024*65.

Packet trace LPAR (Partition logique de suivi de paquet)

Spécifie si des paquets à partir de l'agent de partition logique et vers celui-ci sont suivis.

Par défaut, le suivi est désactivé.

Packet trace client (Client de suivi de paquet)

Spécifie si des paquets à partir de stations de travail et vers celles-ci sont suivis.

Par défaut, le suivi est désactivé.

Client connect timeout (Délai de connexion de client)

Durée (en secondes) de l'attente entre les tentatives de connexion si la passerelle est démarrée en mode client et que la connexion à l'agent de partition logique échoue.

La valeur par défaut est 30.

Vous pouvez configurer ces propriétés dans la fenêtre Agents du portail DevTest. Les propriétés s'affichent dans l'onglet Settings (Paramètres).

Propriétés de la passerelle Mainframe

Les propriétés suivantes doivent être ajoutées au fichier **local.properties** dans le répertoire **LISA_HOME.**

lisa.mainframe.bridge.enabled

Spécifie si la passerelle Mainframe est lancée lors du démarrage du registre. La passerelle Mainframe est exécutée dans le registre et non comme processus externe.

Valeur par défaut : false

lisa.mainframe.bridge.mode

Spécifie le mode d'exécution de la passerelle Mainframe. Le mode définit la relation de parité avec les agents de partition logique. Vous pouvez exécuter la passerelle Mainframe en mode client ou de serveur :

- En mode client, l'agent de partition logique doit être configuré pour être exécuté en mode serveur. La passerelle initialise la connexion à l'agent de partition logique.
- En mode serveur, l'agent de partition logique doit être configuré pour une exécution en mode client. La passerelle attend l'établissement de connexions à partir de l'agent de partition logique.

Valeurs :

- client
- serveur

Valeur par défaut : server

lisa.mainframe.bridge.port

En mode serveur, cette propriété spécifie le port connu sur lequel la passerelle écoute des connexions à partir de la connexion de partition logique. Cette propriété est ignorée en mode client.

Valeurs : un numéro de port valide

Valeur par défaut : 61617

lisa.mainframe.bridge.server.host

En mode client, cette propriété définit l'hôte vers lequel la passerelle initialise une connexion. Cette propriété est ignorée en mode serveur.

Valeurs : un numéro de port valide

Valeur par défaut : localhost

lisa.mainframe.bridge.server.port

En mode client, cette propriété définit le port vers lequel la passerelle initialise une connexion vers l'hôte spécifié. Cette propriété est ignorée en mode serveur.

Valeurs : un numéro de port valide

Valeur par défaut : 3997

lisa.mainframe.bridge.connid

Cette propriété est réservée à une utilisation ultérieure.

Console d'agent z/OS pour DevTest

La passerelle Mainframe fournit une console, disponible via la console de serveur, qui offre une visibilité et une gestion limitées de l'agent CICS pour DevTest. Cela permet à un utilisateur ou à administrateur sans accès à z/OS d'effectuer une gestion limitée de l'agent CICS. Un programmeur de systèmes CICS ou z/OS doit effectuer une gestion complète de l'agent CICS.

La console d'agent z/OS fournit les fonctionnalités suivantes pour une visibilité limitée sur l'agent CICS et permettre de le gérer :

- Vérification de l'activation d'un agent CICS pour DevTest.
- Affichage de certains attributs de l'agent CICS
- Affichage de certains attributs de la région du CICS
- Liste des activités d'enregistrement et de lecture de la virtualisation de l'agent CICS
- Annulation/réinitialisation des activités d'enregistrement et de lecture de la virtualisation d'un agent CICS

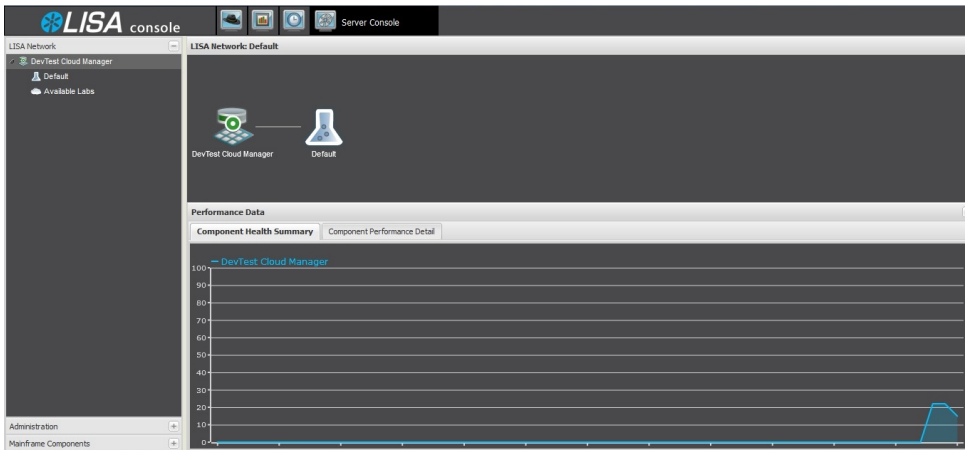
La console d'agent z/OS n'est pas conçue pour remplacer un programmeur de systèmes z/OS ou CICS effectuant des opérations d'agent.

La console ne permet pas :

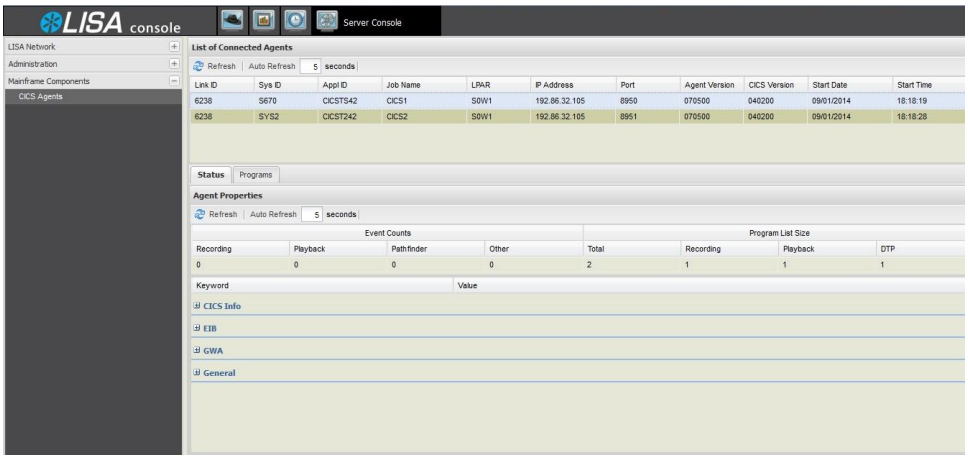
- Le démarrage, l'arrêt ou le redémarrage des agents z/OS pour DevTest
- Toute modification de la configuration d'agent z/OS pour DevTest

Lancement de la console DevTest

Dans la console DevTest, une arborescence Mainframe Components (Composants Mainframe) existe si la passerelle Mainframe est activée.



Pour développer l'arborescence Mainframe Components, cliquez sur +. Les agents CICS et z/OS connectés sont affichés.



Sélectionnez un agent dans le volet supérieur et affichez ou gérez l'agent dans le volet inférieur.

Le volet inférieur comporte deux onglets : Status (Statut) et Programs.

Sous les onglets, vous trouverez quelques nombres de transactions d'agent. Une transaction dans ce cas correspond à un événement CICS unique, comme CICS LINK ou DTP ALLOCATE, pas une transaction CICS entière.

Nombres d'événement

Enregistrement

Affiche le nombre de transactions enregistrées depuis l'initialisation de l'agent.

Playback (Lecture)

Affiche le nombre de transactions virtualisées depuis l'initialisation de l'agent.

Pathfinder

Affiche le nombre de transactions signalées à CAI depuis l'initialisation de l'agent.

Autre

Affiche le nombre de transactions d'autres types depuis l'initialisation de l'agent.

Taille de liste de programme

Total

Nombre total d'entrées dans la liste de programmes. La liste de programmes contient une entrée pour chaque transaction enregistrée ou virtualisée.

Enregistrement

Nombre de transactions en mode d'enregistrement dans la liste de programmes.

Playback (Lecture)

Nombre de transactions en mode de lecture (virtualisation) dans la liste de programmes.

DTP

Nombre de transactions de processus de transaction distribué dans la liste de programmes (en mode d'enregistrement et de lecture).

Onglet Etat

L'onglet Status (Statut) affiche les attributs de l'agent.

L'arborescence CICS Info (Informations CICS) contient certaines informations de configuration de CICS provenant de la région CICS dans laquelle l'agent est exécuté.

L'arborescence EIB contient l'interface de bloc d'exécution de l'agent CICS (transaction CICS).

L'arborescence GWA contient la zone de travail global de l'agent CICS. Cette zone est utilisée pour communiquer avec les diverses sorties CICS utilisées par l'agent CICS.

L'arborescence General (Général) contient la version de l'agent, l'ID de transaction, etc.

Onglet Program (Programme)

L'onglet Program indique toutes les transactions (CICS LINK, DTP, etc.) déployées vers l'agent CICS.

L'exemple illustre deux programmes :

1. PROGRAM1 est en mode d'enregistrement pour les liens CICS LINK.
2. PROGRAM2 est en mode de lecture pour les commandes de traitement de transaction distribué.

Si l'un de ces programmes est orphelin dans l'agent, vous pouvez le supprimer en désélectionnant la case à cocher correspondante et en cliquant sur Reset Progams (Réinitialiser les programmes). Cette action le supprime de l'agent CICS.

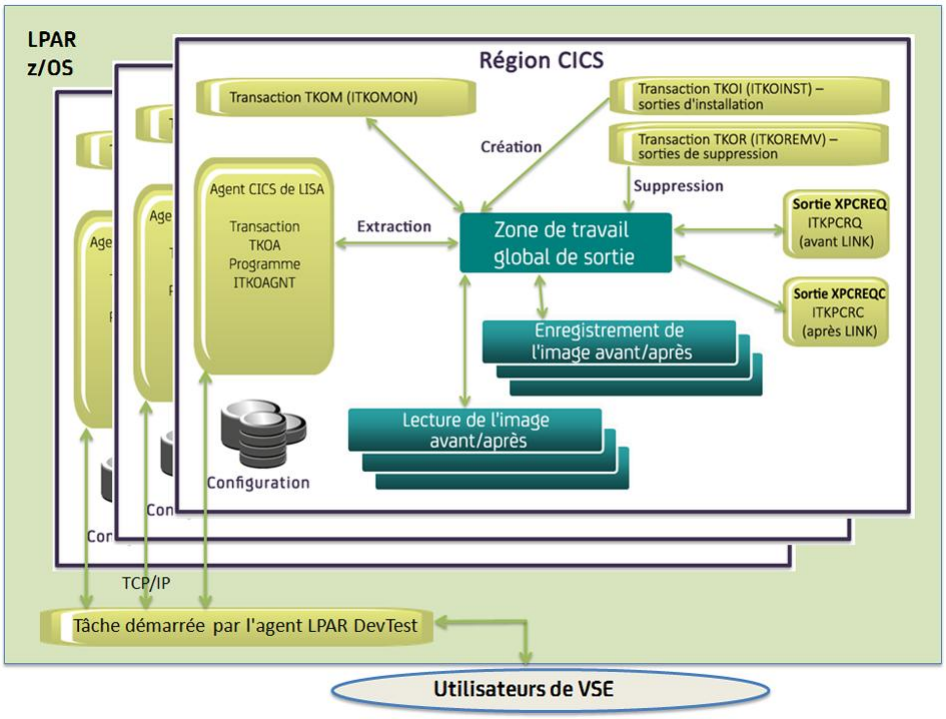
Chapitre 3: Agent CICS pour DevTest

L'agent CICS pour DevTest est un agent CICS résidant pour z/OS qui fournit des fonctions pour la prise en charge du VSE. L'agent est implémenté sous forme de combinaison de transaction TKOA de tâche de longue durée et des sorties CICS, XPCREQ, XPCREQC, XEIIN et XEIOUT. Vous pouvez activer l'agent automatiquement via la PLT (Program load table, table de chargement de programmes) de CICS, ou manuellement via la transaction TKOI à installer et la transaction TKOR à supprimer. La seule condition requise est l'emplacement de l'agent de partition logique pour DevTest.

Ce chapitre traite des sujets suivants :

Présentation de l'agent CICS

Le diagramme suivant illustre les composants de l'agent de partition logique pour DevTest et l'agent CICS pour DevTest.



Procédure d'installation de l'agent CICS

Le package de l'agent CICS pour DevTest comprend les fichiers suivants :

- iTKOCICSAgentDoc.pdf
- iTKOCICSAgentLoadCICS32
- iTKOCICSAgentLoadCICS41
- iTKOCICSAgentLoadCICS42
- iTKOCICSAgentLoadCICS51
- iTKOCICSAgentLoadCICS52
- iTKOCICSAgentCntl

Voici la procédure d'installation :

1. Envoi des fichiers via un site FTP vers le système z/OS cible (page 157)
2. Restauration des ensembles de données partitionnés (PDS) (page 105)
3. Vérification de l'installation de l'interface TCP/IP et de son démarrage dans CICS (page 105)
4. Création du fichier de configuration de l'agent CICS (page 106)
5. Définition des ressources de l'agent CICS pour CICS (page 106)
6. Ajout de l'activation/la désactivation de sortie à la PLT de CICS (page 108)
7. Modification de la JCL de démarrage de CICS (page 109)

Envoi des fichiers via un site FTP vers le système z/OS cible

Utilisez la bibliothèque de chargement qui correspond à votre version de CICS :

- Pour la version 3.2 de CICS, utilisez iTKOCICSAgentLoadCICS32.
- Pour la version 4.1 de CICS, utilisez iTKOCICSAgentLoadCICS41.
- Pour la Version 4.2 de CICS, utilisez iTKOCICSAgentLoadCICS42.
- Pour la version 5.1 de CICS, utilisez iTKOCICSAgentLoadCICS51.
- Pour la version 5.2 de CICS, utilisez iTKOCICSAgentLoadCICS52.

Pour déterminer votre version de CICS, exécutez la transaction suivante et vérifiez la valeur de CICSTslevel. Le niveau de version/maintenance de CICSTslevel est au format vvrrmm : CECI INQUIRE SYSTEM CICSTslevel.

Les fichiers iTKOCICSAgentLoadCICSvvv et iTKOCICSAgentCntl sont envoyés via un site FTP au système z/OS cible en mode binaire avec les attributs z/OS indiqués dans l'exemple qui suit.

L'exemple suivant de client FTP de ligne de commande indique le transfert approprié vers z/OS des ensembles de données de fichiers de distribution **iTKOCICSAgentCntl** et de **iTKOCICSAgentLoadCICS42** vers des ensembles de données ITKO.CICSAGNT.CNTL.UNLOAD et ITKO.CICSAGNT.LOAD.UNLOAD de z/OS . Vous pouvez changer les noms de fichier /OS pour vos conventions d'attribution de nom d'ensemble de données.

```
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put iTKOCICSAgentLoadCICS42 'ITKO.CICSAGNT.LOAD.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.LOAD.UNLOAD
250 Transfer completed successfully.
ftp> put iTKOCICSAgentCntl 'ITKO.CICSAGNT.CNTL.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.CNTL.UNLOAD
250 Transfer completed successfully.
```

Restauration des ensembles de données partitionnés (PDS)

Les PDS de la bibliothèque de chargement et la bibliothèque de contrôle (JCL) de l'agent CICS sont restaurés à l'aide de la commande TSO RECEIVE, généralement exécutée dans un job de traitement par lots.

Dans l'exemple de JCL suivant, les PDS sont restaurés à partir des ensembles de données de fichiers **ITKO.CICSAGNT.LOAD** et **ITKO.CICSAGNT.CNTL** envoyés via un site FTP sur le volume VOL001. Vous pouvez changer les noms d'ensemble de données pour vos conventions d'attribution de nom d'ensemble de données de site.

```
//job
//*
//* Unload the LOAD Library with TSO RECEIVE
//*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('ITKO.CICSAGNT.LOAD.UNLOAD')
   DATASET('ITKO.CICSAGNT.LOAD') VOLUME(VOL001)
RECEIVE INDSNAME('ITKO.CICSAGNT.CNTL.UNLOAD')
   DATASET('ITKO.CICSAGNT.CNTL') VOLUME(VOL001)

/*
```

Vérification de l'installation de l'interface TCP/IP

L'agent CICS requiert l'installation et le lancement de l'interface TCP/IP de CICS. Pour vérifier l'installation de l'interface TCP/IP, utilisez la transaction suivante : EZAO, INQUIRE, CICS.

Création du fichier de configuration de l'agent CICS

L'agent CICS communique avec l'agent de partition logique via TCP/IP. Le fichier de configuration de l'agent CICS identifie l'adresse IP et le numéro de port TCP de l'agent de partition logique. Ce fichier est accessible en tant que file d'attente de données transitoires lors du démarrage de l'agent. Cet ensemble de données possède les attributs suivants : DSORG=PS, RECFM=FB, LRECL=80.

Dans le fichier, les lignes de la colonne 1 contenant un astérisque (*) sont ignorées. Le premier enregistrement sans astérisque dans la colonne 1 contient l'adresse IP et le numéro de port de l'agent de partition logique. Le format de cet enregistrement doit être le suivant :

Octets 1 à 15	Octet 16	Octets 17 à 21	Octets 22 à 80
Adresse IP de l'agent de partition logique	vide	Numéro de port de l'agent de partition logique	inutilisés

La configuration de membre des PDS de contrôle contient l'exemple suivant :

```
* Configuration de l'agent CICS pour CA ITKO
* Les colonnes 1 à 15 contiennent l'adresse IP de l'agent de partition logique
(15 caractères).
* Les colonnes 17 à 21 contiennent le numéro de port de l'agent de partition logique
(5 caractères).
192.168.0.100    2998
```

Définition des ressources de l'agent CICS pour CICS

La table suivante répertorie les ressources qui doivent être définies pour CICS :

Nom	Type	Description
TKOC	File d'attente de données transitoires	File d'attente de données transitoires de configuration
TKOI	Transaction	Transaction permettant d'installer des sorties CICS et de démarrer la transaction d'agent (TKOA), exécutée en général lors du traitement de PLTPI.
TKOR	Transaction	Transaction permettant de supprimer des sorties CICS et d'arrêter la transaction d'agent (TKOA), exécutée en général lors du traitement de PLTSD.
TKOA	Transaction	Transaction d'agent, généralement démarrée par TKOI et arrêtée par TKOR, bien que la transaction de moniteur (TKOM) puisse également effectuer ces actions.
TKOM	Transaction	Transaction de surveillance, utilisée pour démarrer et arrêter manuellement des sorties et surveiller le traitement d'agent.

ITKOAGNT	Programme	Programme d'agent, démarrré par TKOA.
ITKOINST	Programme	Programme d'installation de sortie, démarrré par TKOI.
ITKOREMV	Programme	Programme de suppression de sortie, démarrré par TKOR.
ITKPCRQ	Programme	XPCREQ de CICS (avant la sortie de CICS LINK), activé par TKOI, désactivé par TKOR.
ITKPCRC	Programme	XPCREQC de CICS (après la sortie de CICS LINK). activé par TKOI, désactivé par TKOR.
ITKOXEIN	Programme	Sortie XEIIN de CICS (avant la sortie de commande CICS), activée par TKOI, désactivée par TKOR.
ITKOXEIO	Programme	Sortie XEIOUT de CICS (après la sortie de commande CICS), activée par TKOI, désactivée par TKOR.
ITKOTABN	Programme	Sortie XPCABND de CICS (sortie d'interruption de tâche), activée par TKOI, désactivés par TKOR.
ITKOMON	Programme	Programme de surveillance

La JCL d'un job pour installer ces ressources via DFHCSDUP est dans le PDS de contrôle nommé DFHCSDUP.

Ajout de l'activation/la désactivation de sortie à la PLT de CICS

Vous pouvez démarrer l'agent CICS et les sorties manuellement à l'aide de la transaction TKOI.

Vous pouvez démarrer l'agent CICS et les sorties manuellement à l'aide de la transaction TKOR.

Une méthode simple pour activer les sorties CICS et démarrer l'exécution de l'agent consiste à placer des entrées dans PLTSI (démarrage de CICS). Voici un exemple de PLTPI. Si SIP contient PLTPI=SI, la table suivante sera assemblée en tant que DFHPLTSI. L'interface TCP/IP de CICS est lancée avant la fermeture de DevTest et avant l'agent.

```
DFHPLT TYPE=INITIAL,SUFFIX=SI
*
* Les programmes suivants sont exécutés lors du deuxième passage de
PLTPI.
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* Les programmes suivants sont exécutés lors du troisième passage de
PLTPI.
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOINST
         DFHPLT TYPE=FINAL
         END
```

Pour désactiver les sorties de DevTest et arrêter l'agent lorsqu'il se ferme, incluez une entrée dans le PLTSD (arrêt de CICS). Voici un exemple de PLTSD. Si SIP contient PLTSD=SD, la table suivante sera assemblée en tant que DFHPLTSD. Les sorties de DevTest et l'agent sont arrêtés avant l'interface TCP/IP de CICS.

```
DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* Les programmes suivants sont exécutés lors du premier passage de
PLTSD.
*
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOREMV
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* Les programmes suivants sont exécutés lors du deuxième passage de
PLTSD.
*
         DFHPLT TYPE=FINAL
         END
```

Modification de la bibliothèque JCL de démarrage de CICS

Pour ajouter les modules de chargement d'agent CICS pour DevTest à CICS, copiez-les dans une bibliothèque de chargement DFHRPL existante. De même, vous pouvez ajouter la bibliothèque de chargement de l'agent CICS pour DevTest à la concaténation de DFHRPL.

La JCL doit également être ajoutée pour définir le fichier de configuration de l'agent (étape 5 (page 106)). Si l'ensemble de données est nommé ITKO.AGENT.CONFIG et que la transaction TKOC de file d'attente de données transitoires a été définie avec DDNAME(ITKOACFG), l'instruction de JCL suivante peut être utilisée :

```
//ITKOACFG DD DSN=ITKO.AGENT.CONFIG,DISP=SHR
```

Remarques sur le stockage de l'agent CICS

L'agent CICS pour DevTest est une tâche de longue durée. L'agent requiert environ 16 k d'espace de stockage de programme sur ESDSA et environ 1 k de stockage dynamique sur ECDSA.

Les sorties de CICS pour DevTest requièrent environ 34 k d'espace de stockage de programme sur ERDSA et environ 3 k d'espace de stockage dynamique (par transaction) sur ECDSA.

Les images enregistrées d'un lien CICS LINK sont créées sur ECDSA lors de l'utilisation de l'enregistreur de VSE.

Les sorties créent des images enregistrées et les transmettent à l'agent CICS. L'agent CICS les envoie à l'agent de partition logique, puis les libère ; leur durée de vie est donc courte.

La taille d'une image enregistrée est d'environ 330 + 2 * ((longueur de COMMAREA ou somme de toutes les longueurs de CONTAINER) + longueur d'INPUTMSG).

La quantité d'ECDSA requise pour accueillir les images enregistrées dépend de la taille moyenne calculée ci-dessus, ainsi que du taux de transactions.

Les images de lecture d'un lien CICS LINK sont également créées dans ECDSA lorsqu'un service virtuel est déployé.

Au début d'un lien CICS LINK virtualisé, les sorties créent une demande de lecture et la transfèrent à l'agent CICS. L'agent CICS l'envoie à l'agent de partition logique.

Lorsque l'agent CICS reçoit la réponse de lecture, la réponse est mise en correspondance avec la demande d'origine. La demande et la réponse sont transférées aux sorties, qui les libèrent et terminent le lien CICS LINK.

La taille d'une demande de lecture et la réponse est d'environ 300 + (longueur de COMMAREA ou somme de toutes les longueurs de CONTAINER).

La quantité d'ECDSA requise pour accueillir les images de lecture dépend de la taille moyenne calculée ci-dessus, ainsi que du taux de transactions.

Remarques sur la sortie CICS

L'agent CICS pour DevTest utilise les points de sortie CICS suivants :

- XPCREQ (avant la sortie de lien CICS LINK)
- XPCREQC (après la sortie de lien CICS LINK)
- XEIIN (avant la sortie de commande CICS)
- XEIOUT (après la sortie de commande CICS)
- XPCABND (transaction ABEND)

XPCREQ et XPCREQC sont utilisés pour la virtualisation de liens CICS LINK.

XEIIN et XEIOUT sont utilisés pour la virtualisation du traitement des transactions distribuées (DTP) de CICS.

XPCABND (et les autres sorties) est utilisé pour CA Continuous Application Insight.

XPCREQ et XPCREQC sont requis. Si la virtualisation du DTP ou CAI ne sont pas requis, vous pouvez supprimer XEIIN et XEIOUT. Pour les supprimer, supprimez ITKOXEIN et ITKOXEIO de la bibliothèque de charge (ou renommez-les).

Si CAI n'est pas requis, vous pouvez supprimer XPCABND. Pour le supprimer, supprimez ITKOTABN de la bibliothèque de chargement (ou renommez-la).

Coexistence avec d'autres sorties

Si une sortie de type XPCREQ, XPCREQC, XEIIN, XEIOUT, ou XPCABND est déjà installée, les sorties de DevTest peuvent coexister avec elles.

CICS appelle les sorties par ordre d'activation. Vous pouvez contrôler l'ordre à l'aide du positionnement de l'entrée ITKOINST dans la PLT.

Récapitulatif du comportement de sortie de l'agent CICS pour DevTest

- Vous pouvez saisir les sorties de l'agent CICS pour DevTest avec un code de retour précédent (résolu par UEPCRCA) autre que zéro. Dans ce cas, la sortie est renvoyée avec le code de retour (R15) défini sur le code de retour précédent (défini par une autre sortie). Les sorties de DevTest peuvent coexister avec d'autres sorties s'il ne s'agit pas de la première sortie appelée par CICS.
- Les sorties de lien CICS LINK pour DevTest (XPCREQ et XPCREQC) utilisent le jeton (UEPPCTOK) transféré par CICS aux sorties pour transmettre l'adresse d'un bloc de contrôle de DevTest. Si cette adresse est endommagée par une autre XPCREQ ou XPCREQC, un arrêt anormal peut se produire, ou la sortie XPCREQC de DevTest peut écrire un message unique : **ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit (Jeton endommagé dans la sortie ITKPCRC)**.
- Si l'agent CICS pour DevTest **avant** les sorties (XPCREQ et XEIIN) doit omettre une commande pour la virtualiser, R15 renvoie le code de retour **bypass** (UERCBYP). Si CICS appelle une autre sortie du même type après la sortie de DevTest, la sortie doit propager le code de retour d'omission vers CICS. Sinon, la commande ne sera pas ignorée et la virtualisation échouera.
- Si l'indicateur de **récursivité** (résolu par UEPRECUR) est défini sur une valeur autre que zéro, les sorties XPCREQC et XPCREQ de DevTest seront immédiatement renvoyées. Cela permet d'éviter des problèmes qui peuvent survenir lorsqu'une autre sortie de ce type utilise une commande CICS LINK.
- La sortie XPCABND (transaction ABEND) est toujours renvoyée avec UEPCRCA dans R15, elle doit donc coexister avec d'autres sorties XPCABND. La seule exception à ce comportement concerne l'événement d'une fonction XPI renvoyant un code de **tâche purgée** (consultez le paragraphe suivant).
- Si l'une des sorties de l'agent CICS pour DevTest reçoit un code de retour de **tâche purgée** à partir de la fonction XPI de CICS, elle sera renvoyée avec un code de **purge** (UERCPURG) dans R15. La sortie remplace le code de retour précédent (résolu par UEPCRCA) par UERCPURG. Si CICS appelle une autre sortie du même type après la sortie de DevTest, la sortie doit propager le code de retour de purge vers CICS.

Sorties d'utilisateur de l'agent CICS

ITKOUEX1 : sortie d'initialisation de l'agent CICS pour DevTest

Cette sortie est appelée pendant l'initialisation de l'agent CICS pour DevTest. Vous pouvez utiliser la sortie pour effectuer les opérations suivantes :

- Indiquer le groupe d'utilisateurs dont cette région CICS est membre
- Indiquer le CICSPlex dont cette région CICS est membre
- Allouer un espace de stockage et transmettre l'adresse/la longueur à d'autres sorties à l'aide du paramètre de jeton
- Effectuer une autre initialisation requise par d'autres sorties d'utilisateur

Cette sortie est appelée via un lien CICS LINK, de sorte à pouvoir l'écrire dans une langue prise en charge par CICS. Pendant l'initialisation, une tentative de lien CICS LINK vers ITKOUEX1 est effectuée.

Si cette sortie n'est pas nécessaire, effectuez l'une des actions suivantes :

- N'incluez pas de module nommé ITKOUEX1 dans la concaténation DFHRPL.
- Désactivez la sortie dans PPT.

Les paramètres sont transférés dans COMMAREA.

Le tableau ci-dessous décrit ces paramètres.

Parameter (Paramètre)	Longueur en octets	Entrée ou sortie	Type de données	Description
Jeton	8	Entrée/sortie	Unknown (Inconnu)	Huit octets qui peuvent être utilisés par les sorties d'utilisateur. Conservés d'un appel de sortie à l'autre.
Group (Groupe)	8	Sortie	Caractère	Défini par la sortie. Identifie un groupe d'utilisateurs dont CICS est membre.
CICSPlex	8	Sortie	Caractère	Défini par la sortie. Identifie un CICSPlex dont CICS est membre.
Sysid	4	Entrée	Caractère	SYSID de CICSPlex issu de CICS ASSIGN SYSID().
Applid	8	Entrée	Caractère	APPLID de CICSPlex issu de CICS ASSIGN APPLID().
Jobname	8	Entrée	Caractère	Nom du job CICS à partir de CICS INQUIRE SYSTEM JOBNAME().
LPAR	8	Entrée	Caractère	Nom de la partition logique à partir de CVTSNAME dans laquelle CICS réside.

Sysplex	8	Entrée	Caractère	Nom de la partition logique à partir de ECVTSPLX dans laquelle CICS réside.

Un exemple de COBOL ITKOUEX1 est disponible dans ITKO.CICSAGNT.CNTL(ITKOUEX1). Cet exemple définit le groupe d'utilisateurs sur les quatre premiers caractères du nom du job CICS.

Le copybook de COBOL COMMAREA est disponible dans ITKO.CICSAGNT.CNTL(ITKOEX1C).

Le copybook d'assembleur COMMAREA est disponible dans ITKO.CICSAGNT.CNTL(ITKOEX1A).

La bibliothèque JCL de compilation COBOL est disponible dans ITKO.CICSAGNT.CNTL(COMPUEX1).

ITKOUEX2 : sortie d'arrêt de l'agent CICS pour DevTest

Cette sortie est appelée pendant l'arrêt de l'agent CICS pour DevTest. Vous pouvez l'utiliser pour effectuer un nettoyage requis par les sorties d'utilisateur (par exemple, libération d'espace de stockage, fermeture de fichiers).

Cette sortie est appelée via un lien CICS LINK, de sorte à pouvoir l'écrire dans une langue prise en charge par CICS. Pendant l'arrêt de l'agent, une tentative de lien CICS LINK à ITKOUEX2 est effectuée. Si cette sortie n'est pas nécessaire, effectuez l'une des actions suivantes :

- N'incluez pas de module nommé ITKOUEX2 dans la concaténation DFHRPL.
- Désactivez la sortie dans PPT.

Les paramètres sont transférés dans COMMAREA.

Le tableau ci-dessous décrit ces paramètres.

Parameter (Paramètre)	**Longueur en octets**	**Entrée ou sortie**	**Type de données**	**Description**
Jeton	8	Entrée/sortie	Unknown (Inconnu)	Huit octets qui peuvent être utilisés par les sorties d'utilisateur. Conservés d'un appel de sortie à l'autre.

Un exemple de COBOL ITKOUEX2 est disponible dans ITKO.CICSAGNT.CNTL(ITKOUEX2).

Le copybook de COBOL COMMAREA est disponible dans ITKO.CICSAGNT.CNTL(ITKOEX2C).

Le copybook d'assembleur COMMAREA est disponible dans ITKO.CICSAGNT.CNTL(ITKOEX2A).

La bibliothèque JCL de compilation COBOL est disponible dans ITKO.CICSAGNT.CNTL(COMPUEX2).

ITKOUEX3 : sortie de consolidation CICS COMMAREA pour LISA

Cette sortie est utilisée pour gérer une COMMAREA qui contient des adresses liées à des emplacements de stockage hors de la COMMAREA. Cette sortie consolide les fragments de stockage avec la COMMAREA pour créer une COMMAREA contiguë unique.

Cette sortie est appelée :

- Pendant l'enregistrement de CICS LINK, pour les images antérieures et ultérieures, afin de créer une COMMAREA de remplacement sans adresses liées à des données hors de la COMMAREA. La sortie est chargée d'obtenir la nouvelle zone de stockage à l'aide de la commande GETMAIN. L'agent CICS pour DevTest libérera la zone à l'aide de la commande FREEMAIN. Les adresses dans la COMMAREA qui pointe vers des emplacements dans la COMMAREA sont également problématiques. Dans ce cas, la sortie doit convertir les adresses en décalages relatifs pendant l'enregistrement, puis de nouveau en adresses réelles pendant la lecture.
- Pendant la lecture de CICS LINK (virtualisation) pour restaurer la COMMAREA d'origine à partir de la réponse de DevTest
- Pendant l'initialisation de l'agent CICS (après la sortie d'utilisateur 1) de sorte que la sortie puisse effectuer toute tâche d'initialisation supplémentaire.
- Pendant l'arrêt de l'agent CICS (avant la sortie d'utilisateur 2) de sorte que la sortie puisse effectuer toute tâche d'arrêt supplémentaire.

Cette sortie est appelée via un lien CICS LINK, de sorte à pouvoir l'écrire dans une langue prise en charge par CICS. Toutefois, le langage de l'assembleur est le plus adapté pour résoudre la manipulation et le déplacement des données de longueur variable. Pendant l'initialisation, une tentative de lien CICS LINK vers ITKOUEX3 est effectuée.

Si cette sortie n'est pas nécessaire, effectuez l'une des actions suivantes :

- N'incluez pas de module nommé ITKOUEX3 dans la concaténation DFHRPL.
- Désactivez la sortie dans PPT.

Les paramètres sont transférés dans COMMAREA.

Le tableau ci-dessous décrit ces paramètres.

Parameter (Paramètre)	Longueur en octets	Entrée ou sortie	Type de données	Description
Jeton	8	Entrée/sortie	Unknown (Inconnu)	Huit octets qui peuvent être utilisés par les sorties d'utilisateur. Conservés d'un appel de sortie à l'autre.
Type d'appel	1	Entrée	Binaire	12 indique une lecture. 16 indique un arrêt.
Type d'image	1	Entrée	Binaire	4 indique une image antérieure. 8 indique une image ultérieure.
Adresse COMMAREA	4	Entrée	Address	Adresse COMMAREA transférée dans l'application CICS LINK
Longueur COMMAREA	2	Entrée	Binaire	Longueur de la COMMAREA transférée dans l'application CICS LINK
Programme appelant	8	Entrée	Caractère	Nom du programme qui émet la commande CICS LINK
Programme cible	8	Entrée	Caractère	Nom de programme lié à l'aide de la commande CICS LINK.
Nouvelle adresse COMMAREA	4	Entrée/sortie	Address	Pour l'enregistrement : l'adresse de la nouvelle COMMAREA consolidée est placée ici par cette sortie. Cette sortie obtient généralement la zone à l'aide de la commande GETMAIN. Pour la lecture : l'adresse de la nouvelle COMMAREA consolidée à partir de la réponse du VSE est placée ici. Cette sortie copie le contenu dans l'application COMMAREA.
Nouvelle longueur de COMMAREA	2	Entrée/sortie	Binaire	Pour l'enregistrement : la longueur de la nouvelle COMMAREA consolidée est placée ici par cette sortie. Pour la lecture : la longueur de la nouvelle COMMAREA consolidée à partir de la réponse du VSE est placée ici pour être utilisée par cette sortie.

Code de retour	1	Sortie	Binaire	0 indique que la COMMAREA a été consolidée. L'agent CICS pour DevTest utilisera les nouvelles **adresse COMMAREA** et **longueur COMMAREA** pour créer l'image enregistrée. 4 indique que cette sortie a abouti, mais qu'aucune COMMAREA n'a été créée. L'agent CICS pour DevTest utilise la COMMAREA à partir de CICS LINK pour créer l'image enregistrée. 8 indique que cette sortie a échoué et qu'il n'existe aucune nouvelle COMMAREA. L'agent CICS pour DevTest est renvoyé vers l'application à partir CICS LINK avec les valeurs EIBRCODE, EIBRESP et EIBRESP2 comme indiqué dans les paramètres ci-dessous. 12 indique l'ensemble du traitement du code de retour 8 et que cette sortie doit être supprimée de l'enregistrement et de la lecture ultérieurs de CICS LINK pour DevTest.
EIBRCODE	6	Sortie	Binaire	Valeur EIBRCODE à utiliser lorsque le **code de retour** est 8 ou 12
EIBRESP	4	Sortie	Binaire	Valeur EIBRRESP à utiliser lorsque le **code de retour** est 8 ou 12
EIBRESP2	4	Sortie	Binaire	Valeur EIBRRESP2 à utiliser lorsque le **code de retour** est 8 ou 12

Un exemple de langage d'assembleur ITKOUEX3 est disponible dans ITKO.CICSAGNT.CNTL(ITKOUEX3).

Dans cet exemple, une COMMAREA est gérée au format produit par le langage Natural de Software AG lors de l'appel de COBOL avec le paramètre Natural CALLRPL=(ALL,3) ou CALLRPL=(ALL,4). Avec ce format de COMMAREA, la longueur de COMMAREA est de 12 (lorsque CALLRPL=(ALL,3) est utilisé) et de 16 (lorsque CALLRPL=(ALL,4) est utilisé). Le premier mot entier dans la COMMAREA est l'adresse d'une liste de paramètres. Le bit le plus significatif défini sur 1 dans l'adresse indique le dernier paramètre. Le troisième mot entier est l'adresse d'une liste de longueurs de paramètre.

Le copybook d'assembleur COMMAREA est disponible dans ITKO.CICSAGNT.CNTL(ITKOEX3A).

La bibliothèque JCL d'assemblage est disponible dans ITKO.CICSAGNT.CNTL(ASMBUEX3).

Opération d'agent CICS

Démarrage de l'agent CICS et de sorties pour DevTest

L'agent CICS et les sorties sont démarrés pendant le traitement de la PLT de démarrage de CICS. Vous pouvez également démarrer l'agent et les sorties manuellement en exécutant la transaction TKOI ou avec la transaction de surveillance TKOM. La transaction TKOI installe les sorties et démarre la transaction de l'agent, TKOA.

Arrêt de l'agent CICS et de sorties pour DevTest

L'agent CICS et les sorties sont arrêtés pendant le traitement de la PLT de fermeture de CICS. Vous pouvez également arrêter l'agent et les sorties manuellement en exécutant la transaction TKOR ou avec la transaction de surveillance TKOM.

Dans de rares cas, la fin de la transaction de l'agent CICS, TKOA, peut échouer. Si vous pensez que c'est le cas, utilisez la commande CEMT I TAS pour confirmer qu'elle est encore en cours d'exécution. Si c'est le cas, utilisez la commande CEMT SET TAS(nnn)PURGE ou CEMT SET TAS(nnn) FORCEPURGE comme il convient pour arrêter TKOA.

Impact sur les commandes LINK de CICS virtualisées

Dans de rares cas, la virtualisation de CICS LINK peut échouer. Dans cet événement, CICS LINK est renvoyé vers l'application avec la condition Invalid Request (INVREQ): EIBRCODE=x'E00000000000' and EIBRESP=16 (Demande non valide). En général, cette situation résulte d'un arrêt anormal de transaction, sauf si l'application dispose du traitement de condition INVREQ. Vous pouvez utiliser EIBRESP2 pour déterminer la cause exacte de l'échec de la virtualisation :

500 / x'1F4'

Aucune réponse de lecture n'a été reçue à partir du VSE. Cela est probablement dû à une expiration de délai. L'intervalle d'expiration est défini sur 10 secondes. Si la valeur d'expiration doit être changée, contactez le service de support clientèle.

501 / x'1F5'

L'agent CICS pour DevTest est inactif. La sortie After CICS LINK (après la sortie de CICS LINK) a détecté que l'agent CICS pour DevTest est inactif. Aucune réponse de lecture ne peut être attendue.

502 / x'1F6'

Echec de la virtualisation. Une réponse de lecture a été reçue à partir de Service Virtualization et la mise à jour de la COMMAREA de CICS LINK ou de CHANNEL/CONTAINERs a échoué. Consultez le journal de job CICS et recherchez un message d'échec de la commande CICS.

503 / x'1F7'

L'agent de partition logique de DevTest a été arrêté, ce qui a annulé toutes les demandes de lecture en attente.

Impact sur les commandes DTP/MRO de CICS virtualisées

Dans de rares cas, la virtualisation d'une commande DTP/MRO de CICS (ALLOCATE, FREE, SEND, RECEIVE, CONVERSE, or EXTRACT ATTRIBUTES) peut échouer. Dans cet événement, la commande CICS est renvoyée vers l'application avec la condition Invalid Request (INVREQ): EIBRCODE=x'E00000000000' et EIBRESP=500 (Demande non valide). En général, cette situation résulte d'un arrêt anormal de transaction, sauf si l'application dispose du traitement de condition INVREQ. Vous pouvez utiliser EIBRESP2 pour déterminer la cause exacte de l'échec de la virtualisation :

501 / x'1F5'

L'agent CICS pour DevTest est inactif. La sortie After CICS LINK (après la sortie de CICS LINK) a détecté que l'agent CICS pour DevTest est inactif. Aucune réponse de lecture ne peut être attendue.

502 / x'1F6'

Aucune réponse n'a été reçue à partir de Service Virtualization.

503 / x'1F7'

Une réponse non valide a été reçue à partir de Service Virtualization. Consultez le journal de job CICS et recherchez un message d'échec de la commande CICS.

504 / x'1F8'

La table de session est complète.

505 / x’1F9’

Une tentative d'ajout de la demande de lecture à la file d'attente de lecture a échoué.

506 / x’1FA’

L'intervalle d'expiration XEIIN WAIT a expiré avant qu'une réponse ait été reçue à partir de Service Virtualization.

507 / x’1FB’

Une commande GETMAIN pour la demande ou la réponse a échoué.

Transaction de surveillance (TKOM)

La transaction de surveillance (TKOM) permet d'effectuer les opérations suivantes :

- Vérification du statut de l'agent et des sorties
- Affichage de la Global Work Area (Zone de travail globale, GWA) au format hexadécimal
- Affichage de certains champs formatés/décodés de la GWA
- Vérification des nombres d'enregistrements et de lectures
- Vérification du nombre de tables de programme
- Affichage de la table de programme
- Démarrage et arrêt de l'agent
- Démarrage et arrêt des sorties
- Vérification de la profondeur actuelle (nombre) et des seuils supérieurs (Max) des trois files d'attente principales :

  File d'attente d'enregistrement (RecQ) :

  - Définit la file d'attente d'images enregistrées.

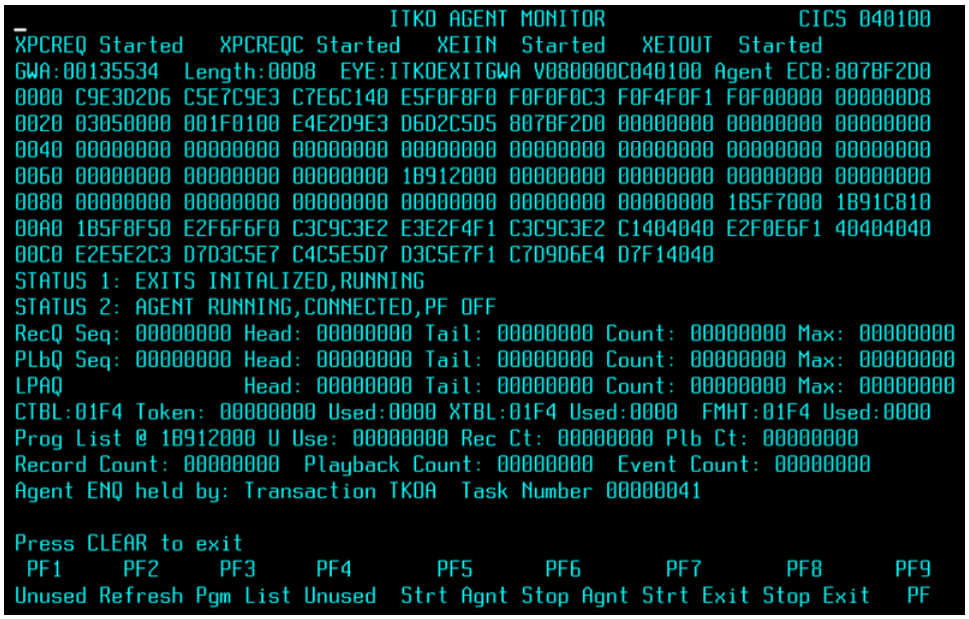
  File d'attente de lecture (PlbQ) :

  - Définit la file d'attente d'images de lecture à envoyer à l'agent de partition logique pour DevTest.

  File d'attente de l'agent de partition logique (LPAQ) :

  - Définit la file d'attente d'images de lecture en attente de réponse de l'agent de partition logique pour DevTest.
- Evaluation du nombre de tâches de la transaction d'agent

La fenêtre de surveillance des transactions s'affiche comme suit :



Messages de l'agent CICS

Les messages sont écrits dans la console d'opérateur et dans le journal de job CICS par l'agent CICS et les sorties pour DevTest.

Le programme d'installation de sortie DevTest (Transaction TKOI, programme ITKOINST) écrit les messages suivants :

- ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started (Sortie iTKO activée et démarrée)
- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started (Sortie iTKO activée, mais non démarrée)
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999 (Erreur de GWA de la sortie iTKO. Adresse : xxxxxxxx Longueur : 9999)

Le programme d'installation de sortie DevTest (Transaction TKOR, programme ITKOREMV) écrit les messages suivants :

- ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled (Sortie iTKO désactivée)

La sortie XPCREQ CICS (avant CICS LINK) (module ITKPCRQ) ou la sortie XPCREQC CICS (après CICS LINK) (module ITKPCRC) écrit les messages suivants :

- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit (GWA manquant ou trop limité pour la sortie)
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (Erreur dans le nom du programme)
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx (Enregistrement abandonné du programme d'appel LINK pour la tâche 9999999 de transaction xxxx de programme appelé Term xxxx)
- ITKO0010 – [Recorded Image | Playback Image | LPAR Agent] PLO failed in [ITKPCRQ | ITKPCRC] Exit ([Image enregistrée | Image de lecture | Agent de partition logique] Echec du OLP dans la sortie [ITKPCRQ | ITKPCRC])
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx (Echec de la création d'image antérieure/ultérieure avec de code du motif xxx)

- ITKO0012 - Image update failed with reason code 999 (Echec de la mise à jour de l'image avec le code du motif 999)
- ITKO0013 - PDU Parsing Error in ITKPCRC Exit (Erreur d'analyse du PDU dans la sortie ITKPCRC)
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit (Erreur : jeton endommagé dans la sortie ITKPCRC)

L'agent CICS pout DevTest (transaction TKOA, programme ITKOAGNT) écrit les messages suivants :

- ITKO0101 - Agent waiting for exits (Agent en attente de sorties)
- ITKO0102 - Agent initialization complete (Initialisation de l'agent terminée)
- ITKO0103 - Agent shutting down (Arrêt de l'agent)
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent ([Image Enregistrée | Image de lecture | Image de l'agent de partition logique] Echec du PLO dans l'agent)
- ITKO0105 - iTKO Agent started while another agent is running (L'agent iTKO a démarré pendant l'exécution d'un autre agent)
- ITKO0106 - iTKO Agent connected to LPAR Agent (Agent iTKO connecté à l'agent de partition logique)
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999 (Adresse IP de l'agent de partition logique de configuration d'agent : nnn.nnn.nnn.nnn port 99999)
- ITKO0108 - iTKO GWA error - Address is xxxxxxxx Length is 9999 (Erreur GWA d'iTKO. Adresse : xxxxxxxx Longueur : 9999)
- ITKO0109 - Socket func xxxxxxxx failed. (Echec de la fonction de socket xxxxxxxx) RETCODE=xXXXXXXXX ERRNO=999999
- ITKO0110 - Agent CICS cmd failed EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0111 - iTKO Agent disconnected from LPAR Agent (Agent iTKO déconnecté de l'agent de partition logique)
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de l'image enregistrée à l'agent de partition logique)
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de l'image de lecture à l'agent de partition logique)
- ITKO0114 - iTKO Agent error sending response to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de la réponse à l'agent de partition logique)
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent (Erreur de l'agent iTKO lors de la réception de la demande à partir de l'agent de partition logique)
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address (Erreur de l'agent iTKO : adresse IP de l'agent de partition logique non valide)

- ITKO0117 - iTKO Agent released nnnnnn recorded images (L'agent iTKO a émis nnnnnn images enregistrées)
- ITKO0118 - iTKO Agent canceled nnnnnn playback images (L'agent iTKO a annulé nnnnnn images de lecture)
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images (L'agent iTKO a annulé nnnnnn images de lecture en attente)
- ITKO0120 - ITKO Agent released playback image: token xxxxxxxxxxxxxxxx Reason xxx (L'agent iTKO a émis nnnnnn images de lecture : jeton xxxxxxxxxxxxxxxx motif xxx)
- ITKO0121 - iTKO Agent error reading configuration (Erreur de l'agent iTKO lors de la lecture de la configuration)
- ITKO0122 - Agent TCP/IP API Timeout (Expiration du délai de l'API TCP/IP de l'agent)
- ITKO0123 - Agent waiting for TCP/IP API (Agent en attente de l'API TCP/IP)
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxx (Expiration du délai de réponse de l'agent ITKO pour le jeton xxxxxxxxxxxxxxx)
- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 (Echec de l'appel à ITKOUEX1/ITKOUEX2/ITKOUEX3)
- ITKO0126 - Handshake error encountered Shutting down Agent (Une erreur d'établissement de liaison s'est produite. Arrêt de l'agent)

La sortie XEIIN de CICS (avant la commande CICS) (module ITKOXEIN) ou XEIOUT de CICS (après la commande CICS, module ITKOXEIO) écrit les messages suivants :

- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed (Echec de ITKOXEIN XPI INQ_APPLICATION_DATA)
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed (Echec du stockage dynamique avec la commande Getmain)
- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed (Echec du stockage dynamique avec la commande Freemain)
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed (Echec du stockage d'images avec la commande Getmain)
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed (Echec du stockage d'images avec la commande Freemain)
- ITKO205 - ITKOXEIN Task table full - discarding image (Table de tâche ITKOXEIN complète - rejet de l'image)
- ITKO206 - ITKOXEIN XPI WAIT Failed (Echec de ITKOXEIN XPI WAIT)
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed (Echec de demande/réponse de lecture avec la commande Freemain)
- ITKO208 - ITKOXEIO Add to recording queue failed (Echec de l'ajout à la file d'attente d'enregistrement)

- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed (Echec du stockage d'images SET() avec la commande Freemain)
- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed (Echec du stockage d'images SET() avec la commande Getmain)

Informations complémentaires :

Descriptions de message de l'agent CICS pour DevTest (page 124)

Descriptions de message de l'agent CICS pour DevTest

Pour obtenir des descriptions de chaque message, cliquez sur les liens suivants :

ITKO0001

ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

Une commande CICS a échoué et ITKOINST a été interrompu. La valeur hexadécimale EIBFN identifie la commande de CICS qui a échoué. Commande en échec :

EIBFN	CICS Command
x0C02	GETMAIN
x1008	START
x2202	ENABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

Action :

Les codes EIBRCDE, EIBRESP et EIBRESP2 permettent de déterminer la cause de l'échec et de la corriger. Causes d'échec possibles :

1. L'utilisateur qui exécute la transaction TKOI ne dispose pas des droits d'exécution de commandes de l'interface de programmation de système CICS (interface SPI) (par exemple : EXTRACT EXIT).
2. Une commande START (Démarrer) pour la transaction TKOA de l'agent a échoué, car TKOA n'est pas définie ou est définie incorrectement.

ITKO0002

ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started (Sortie iTKO activée et démarrée)

Type de message :

Informational (Information)

Description :

La sortie indiquée (XPCREQ ou XPCREQC) a été installée à l'adresse indiquée avec une zone de travail globale (GWA) à l'adresse indiquée et avec la longueur indiquée. La sortie est activée et a démarré.

Généralement, deux de ces messages s'affichent : l'un pour XPCREQ et l'autre pour XPCREQC. Les deux ont les mêmes adresse et longueur GWA.

Action :

Aucun

ITKO0003

ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started (Sortie iTKO activée, mais non démarrée)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

La sortie indiquée (XPCREQ ou XPCREQC) a été activée ; toutefois, la sortie n'a pas pu démarrer. La fonctionnalité d'enregistrement et de lecture d'ITKO ne fonctionne pas correctement.

Le message ITKO0001 accompagne généralement ce message.

Action :

Diagnostiquez le message ITKO0001 associé et corrigez le problème.

Supprimez les sorties avec TKOR et réinstallez-les avec TKOI.

ITKO0004

ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999 (Erreur de GWA de la sortie iTKO. Adresse : xxxxxxxx Longueur : 9999)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

Une zone de travail globale trop réduite a été allouée aux sorties ITKO.

Ce message peut être accompagné d'un message ITKO0001.

Action :

Signalez l'erreur au service de support ITKO.

ITKO0005

ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

Une commande CICS a échoué et ITKOREMV a été interrompu. La valeur hexadécimale EIBFN identifie la commande de CICS qui a échoué. Commande en échec :

EIBFN	CICS Command
x0C02	FREEMAIN
x1004	DELAY
x2204	DISABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

Action :

Les codes EIBRCDE, EIBRESP et EIBRESP2 permettent de déterminer la cause de l'échec et de la corriger. L'échec peut être expliqué par le fait que l'utilisateur exécutant la transaction TKOR ne dispose pas des droits d'exécution des commandes (telles que EXTRACT EXIT) de l'interface de programmation de système CICS (interface SPI) .

ITKO0006

ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled (Sortie iTKO désactivée)

Type de message :

Information

Description :

La sortie indiquée (XPCREQ ou XPCREQC) a été désactivée.

Action :

Aucun

ITKO0007

ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit (GWA manquant ou trop limité pour la sortie)

Type de message :

Erreur irrécupérable

Description :

La zone de travail globale pour les sorties n'existe pas ou est trop réduite.

Action :

Signalez l'erreur au service de support ITKO.

ITKO0008

ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999 (Erreur dans le nom du programme)

Type de message :

Erreur irrécupérable

Description :

Une commande CICS a échoué dans le programme de sortie indiqué. La valeur hexadécimale EIBFN identifie la commande de CICS qui a échoué. Commande en échec :

EIBFN	CICS Command
0202	ADDRESS
0208	ASSIGN INVOKINGPROG
0C02	GETMAIN
0C02	FREEMAIN
3414	GET CONTAINER
3416	PUT CONTAINER
6C02	WRITE OPERATOR
9626	STARTBROWSE CONTAINER
9628	GETNEXT CONTAINER

Action :

Les codes EIBRCDE, EIBRESP et EIBRESP2 permettent de déterminer la cause de l'échec et de la corriger. L'échec peut s'expliquer par le fait qu'une commande GETMAIN a échoué dû aux faibles conditions de stockage de CICS.

ITKO0009

ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx (Enregistrement abandonné du programme d'appel LINK pour la tâche 9999999 de transaction xxxx de programme appelé Term xxxx)

Type de message :

Warning (Avertissement)

Description :

Lors de l'enregistrement de CICS LINK du programme appelant vers le programme appelé pour la transaction, la tâche et le terminal indiqués, aucune ressource n'était disponible pour gérer l'image enregistrée. Un message ITKO0008 associé peut identifier la ressource (par exemple, un échec GETMAIN pour l'image enregistrée). L'agent de partition logique pour DevTest a pu se déconnecter de l'agent CICS pour DevTest.

Action :

Examinez la condition entraînant l'omission de l'image enregistrée.

ITKO0010

ITKO0010 – [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit ([Image enregistrée | Image de lecture | Liste de l'agent de partition logique] Echec de la commande OLP dans la sortie [ITKPCRQ | ITKPCRC])

Type de message :

Warning (Avertissement)

Description :

La commande PLO (Perform Locked Operation z/OS) a échoué.

La commande PLO est utilisée pour sérialiser la manipulation de trois files d'attente/listes utilisées pour communiquer entre les sorties et l'agent. La file d'attente Recorded Images (Images enregistrées) contient des images enregistrées que l'agent doit traiter. La file d'attente Playback Image (Image de lecture) contient des demandes de lecture pour l'agent doit traiter. La liste LPAR Agent (Agents de partition logique) contient les réponses de lecture que l'agent reçoit pour les transférer vers les sorties.

Pour une entrée de file d'attente Recorded Images (Images enregistrées), la sortie rejette l'image enregistrée.

Pour une entrée de file d'attente Playback Image (Image de lecture), l'image est rejetée et la transaction continue sans lecture (virtualisation).

Pour une entrée de liste LPAR Agent (Agents de partition logique) (réponse de lecture), l'image devient orpheline dans la liste d'agents de partition logique par la sortie.

Action :

Signalement de l'erreur au service de support

ITKO0011

ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx (Echec de la création d'image antérieure/ultérieure avec de code du motif xxx)

Type de message :

Erreur

Description :

Dans la sortie XPCREQ (module ITKPCRQ) ou XPCREQC (module ITKOPCRC), la création d'une image antérieure et ultérieure a échoué avec le code du motif indiqué.

Les codes du motif sont répertoriés ici à des fins d'information uniquement :

8 - Aucune image n'a été transmise à la routine de création.

12- Aucune image antérieure ou ultérieure n'est présente dans la routine de création.

16- longueur de PDU dépassée lors de l'insertion de INPUTMSG dans la routine de création

20- longueur de PDU dépassée lors de l'insertion de COMMAREA dans la routine de création

24- longueur de PDU dépassée lors de l'insertion de conteneur dans la routine de création

28- longueur de PDU dépassée lors de l'insertion de données d'origine dans la routine de création

32- erreur de logique MVCL dans la routine de création

36- Aucun TKOCLNK dans l'image n'a été transmis à la routine de création.

40- La valeur de CPDUVARN est de zéro dans la routine de création.

44- longueur de PDU dépassée lors de l'insertion de EIB dans la routine de création

Action :

Signalez l'erreur au service de support ITKO.

ITKO0012

ITKO0012 - Image update failed with reason code 999 (Echec de la mise à jour de l'image avec le code du motif 999)

Type de message :

Erreur

Description :

Dans la sortie XPCREQ (module ITKPCRQ) ou XPCREQC (module ITKOPCRC), la mise à jour d'une image antérieure et ultérieure a échoué avec le code du motif indiqué.

Les codes du motif sont répertoriés ici à des fins d'information uniquement :

4 - Une erreur de commande CICS s'est produite.

5 - Aucune image n'a été transmise à la routine de mise à jour.

6 - Aucun EIB ultérieur dans l'image n'a été transmis à la routine de mise à jour.

7 - Aucun TKOCLNK dans l'image n'a été transmis à la routine de mise à jour.

8 - CVARTYP au lieu de CVARLINK dans l'image transmise à la routine de mise à jour

9 - CLNKEYE au lieu de CLNK dans l'image transmise à la routine de mise à jour

10 - La valeur de CVARLEN est de zéro dans l'image transmise à la routine de mise à jour.

11- CTNREYE au lieu de CTNR dans l'image transmise à la routine de mise à jour

Action :

Signalement de l'erreur au service de support

ITKO0013

ITKO0013 - PDU Parsing Error in ITKPCRC Exit (Erreur d'analyse du PDU dans la sortie ITKPCRC)

Type de message :

Erreur

Description :

Dans la sortie XPCREQC (module ITKPCRC), la mise à jour d'une COMMAREA développée dans la sortie d'utilisateur 3 a échoué.

Action :

Signalement de l'erreur au service de support

ITKO0014

ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit (Erreur : jeton endommagé dans la sortie ITKPCRC)

Type de message :

Error (Erreur)

Description :

Dans la sortie ultérieure XPCREQC de CICS LINK (module ITKPCRC), un jeton (UEPPCTOK) qui ne traite pas le bloc de contrôle de l'agent CICS pour DevTest approprié a été transféré. Cela est probablement dû à une sortie XPCREQ ou XPCREQC conflictuelle. Ce message est uniquement écrit une seule fois.

Action :

Examinez les sorties XPCREQ et XPCREQC installées dans la région CICS.

ITKO0101

ITKO0101 - Agent waiting for exits (Agent en attente de sorties)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest a démarré avant l'activation des sorties. L'agent patiente pendant l'activation des sorties avant de continuer.

Action :

Aucune

ITKO0102

ITKO0102 - Agent initialization complete (Initialisation de l'agent terminée)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest a terminé l'initialisation et commence la connexion à l'agent de partition logique pour DevTest.

Action :

Aucune

ITKO0103

ITKO0103 - Agent shutting down (Arrêt de l'agent)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest se ferme.

Action :

Aucune

ITKO0104

ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent ([Image Enregistrée | Image de lecture | Image de l'agent de partition logique] Echec du PLO dans l'agent)

Type de message :

Avertissement

Description :

La commande PLO (Perform Locked Operation z/OS) a échoué.

La commande PLO est utilisée pour sérialiser la manipulation de trois files d'attente/listes utilisées pour communiquer entre les sorties et l'agent. La file d'attente Recorded Image (Images enregistrées) contient des images enregistrées que l'agent doit traiter. La file d'attente Playback Image (Image de lecture) contient des demandes de lecture pour l'agent doit traiter. La liste LPAR Agent (Agents de partition logique) contient les réponses de lecture que l'agent reçoit pour les transférer vers les sorties.

Pour les entrées de file d'attente Recorded and Playback Image (Images enregistrées et Images de lecture), l'image est ignorée et une nouvelle tentative de PLO est effectuée lors de la transmission suivante des files d'attente d'images enregistrées/de lecture.

Pour une entrée de liste LPAR Agent (Agents de partition logique) (réponse de lecture), l'image est rejetée et la transaction d'utilisateur qui émet le CICS LINK reçoit une réponse INVREQ (EIBRCODE=x'E00000000000', EIBRESP=16 et EIBRESP2=500).

Action :

Signalement de l'erreur au service de support

ITKO0105

ITKO0105 - iTKO Agent started while another agent is running (L'agent iTKO a démarré pendant l'exécution d'un autre agent)

Type de message :

Warning (Avertissement)

Description :

Un agent CICS pour DevTest (transaction TKOA) a démarré pendant l'exécution d'une autre instance de l'agent. Un seul agent peut être exécuté dans une région CICS, de sorte que le deuxième agent puisse terminer.

Action :

Aucune

ITKO0106

ITKO0106 - iTKO Agent connected to LPAR Agent (Agent iTKO connecté à l'agent de partition logique)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest a établi une connexion TCP/IP à l'agent de partition logique pour DevTest.

Action :

Aucune

ITKO0107

ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999 (Adresse IP de l'agent de partition logique de configuration d'agent : nnn.nnn.nnn.nnn port 99999)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest a lu la configuration (TKOC de file d'attente de données transitoires) et utilise l'adresse IP et le numéro de port indiqués pour se connecter à l'agent de partition logique pour DevTest.

Action :

Aucun

ITKO0108

ITKO0108 - iTKO GWA error - Address is xxxxxxxx Length is 9999 (Erreur GWA d'iTKO. Adresse : xxxxxxxx Longueur : 9999)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

La zone de travail global de sortie (GWA) n'existe pas ou a une longueur incorrecte. L'agent système CICS pour DevTest s'arrête.

Action :

Signalement de l'erreur au service de support

ITKO0109

ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed (Echec de la fonction de socket [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI]) RETCODE=xXXXXXXXX ERRNO=999999

Type de message :

Warning (Avertissement)

Description :

Une erreur s'est produite au niveau de la connexion TCP/IP à l'agent de partition logique.

Le message **Socket func RECV failed. RETCODE=x00000000 ERRNO=000000** (Echec de la fonction de socket RECV) indique généralement une déconnexion de l'agent de partition logique. Le message ITKO0111 accompagne ce message. Une erreur dans la fonction SEND (envoyer) peut également indiquer une déconnexion de l'agent de partition logique.

Les valeurs ERRNO sont expliquées dans le Manuel IBM publication z/OS Communications Server IP CICS Sockets Guide.

Action :

Déterminez si l'erreur est due à une déconnexion de l'agent de partition logique pour DevTest ou à un autre problème lié à l'environnement et corrigez ce problème. Sinon, signalez l'erreur au service de support.

ITKO0110

ITKO0110 - Agent CICS cmd failed EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

Une commande CICS a échoué dans l'agent CICS pour DevTest. La valeur hexadécimale EIBFN identifie la commande de CICS qui a échoué. Commande en échec :

EIBFN	CICS Command
0202	ADDRESS
0208	ASSIGN
0C02	GETMAIN/ FREEMAIN
0E02	LINK
1204	ENQ
1206	DEQ
2206	EXTRACT EXIT
4E02	INQUIRE PROGRAM
5402	INQUIRE SYSTEM
6C02	WRITE OPERATOR
8802	INQUIRE EXITPROGRAM

Action :

Les codes EIBRCDE, EIBRESP et EIBRESP2 permettent de déterminer la cause de l'échec et de la corriger si possible. L'échec peut être expliqué par le fait que l'ID d'utilisateur exécutant la transaction TKOA ne dispose pas des droits d'exécution des commandes (telles que EXTRACT EXIT) de l'interface de programmation de système CICS (interface SPI) .

ITKO0111

ITKO0111 - iTKO Agent disconnected from LPAR Agent (Agent iTKO déconnecté de l'agent de partition logique)

Type de message :

Informational (Information)

Description :

L'agent CICS pour DevTest est déconnecté de l'agent de partition logique pour DevTest.

Action :

Aucune

ITKO0112

ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de l'image enregistrée à l'agent de partition logique)

Type de message :

Error (Erreur)

Description :

L'agent CICS pour DevTest procédait à l'envoi d'une image enregistrée et la connexion TCP/IP avec l'agent de partition logique pour DevTest a été interrompue. L'image enregistrée est omise.

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0113

ITKO0113 - iTKO Agent error sending playback image to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de l'image de lecture à l'agent de partition logique)

Type de message :

Error (Erreur)

Description :

L'agent CICS pour DevTest procédait à l'envoi d'une image de lecture et la connexion TCP/IP avec l'agent de partition logique a été interrompue. L'image enregistrée est omise et le CICS LINK en cours se ferme avec une réponse INVREQ (EIBRCODE=x'E00000000000', EIBRESP=16 et EIBRESP2=500).

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0114

ITKO0114 - iTKO Agent error sending response to LPAR Agent (Erreur de l'agent iTKO lors de l'envoi de la réponse à l'agent de partition logique)

Type de message :

Error (Erreur)

Description :

L'agent CICS pour DevTest procédait à l'envoi d'une réponse à l'agent de partition logique pour DevTest et la connexion TCP/IP a été interrompue.

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique CICS.

ITKO0115

ITKO0115 - iTKO Agent error receiving request from LPAR Agent (Erreur de l'agent iTKO lors de la réception de la demande à partir de l'agent de partition logique)

Type de message :

Error (Erreur)

Description :

L'agent CICS pour DevTest allait recevoir une réponse de l'agent de partition logique pour DevTest et la connexion TCP/IP a été interrompue.

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0116

ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address (Erreur de l'agent iTKO : adresse IP de l'agent de partition logique non valide)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

L'adresse IP de l'agent de partition logique pour DevTest dans le fichier de configuration d'agent CICS pour DevTest (TKOC de file d'attente de données transitoires) n'est pas une adresse IPv4 de notation décimale séparée par des points valide.

Action :

Corrigez l'adresse IP configurée.

ITKO0117

ITKO0117 - iTKO Agent released nnnnnn recorded images (L'agent iTKO a émis nnnnnn images enregistrées)

Type de message :

Warning (Avertissement)

Description :

L'agent de partition logique pour DevTest est déconnecté de l'agent CICS pour DevTest pendant que les images enregistrées étaient placées dans la file d'attente. L'agent CICS pour DevTest supprime des images enregistrées en attente de la file d'attente et les libère.

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0118

ITKO0118 - iTKO Agent canceled nnnnnn playback images (L'agent iTKO a annulé nnnnnn images de lecture)

Type de message :

Warning (Avertissement)

Description :

L'agent de partition logique pour DevTest est déconnecté de l'agent CICS pour DevTest pendant que les images de lecture étaient placées dans la file d'attente. L'agent CICS pour DevTest supprime des images de lecture en attente de la file d'attente et les libère. Chaque image de demande de lecture représente un CICS LINK à virtualiser. Ces commandes de CICS LINK se terminent avec une réponse INVREQ (EIBRCODE=x'E00000000000', EIBRESP=16 et EIBRESP2=500).

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0119

ITKO0119 - iTKO Agent canceled nnnnnn pending playback images (L'agent iTKO a annulé nnnnnn images de lecture en attente)

Type de message :

Warning (Avertissement)

Description :

L'agent de partition logique pour DevTest est déconnecté de l'agent CICS pour DevTest pendant que des images de réponse de lecture étaient en attente étaient attendues. L'agent CICS pour DevTest supprime des images de réponse de lecture en attente de la file d'attente et les libère. Chaque image de réponse de lecture représente un CICS LINK à virtualiser. Ces commandes de CICS LINK se terminent avec une réponse INVREQ (EIBRCODE=x'E00000000000', EIBRESP=16 et EIBRESP2=500).

Action :

Recherchez la cause de la déconnexion de l'agent de partition logique pour DevTest.

ITKO0120

ITKO0120 - ITKO Agent released playback image with token xxxxxxxxxxxxxxxx (L'agent iTKO a émis une image de lecture avec le jeton xxxxxxxxxxxxxxxx.)

Type de message :

Warning (Avertissement)

Description :

L'agent CICS de DevTest a reçu une réponse de lecture de l'agent de partition logique pour DevTest qui ne correspond pas à une demande de lecture en attente. La transaction CICS en attente dans la virtualisation de CICS LINK n'a pas pu être identifiée. L'image de lecture est libérée (rejeté).

Les codes de motif suivants sont possibles. Les codes 5 et 6 indiquent qu'une réponse de lecture a été reçue à partir de l'environnement de services virtuels après une expiration de 30 secondes de l'agent CICS. Le message ITKO0124 accompagne généralement ce message.

1 - Valeur du PDU trop faible

2 - Erreur de format de TKOCPDU

3 - Le préfixe du jeton TKOCPDU n'est pas ITKO.

4 - Le suffixe du jeton TKOCPDU est x'00000000'.

5 - File d'attente de l'agent de partition logique vide - la demande a peut-être expiré.

6 - Introuvable dans la file d'attente de l'agent de partition logique - la demande a peut-être expiré.

7 - Erreur de format de CVAR

8 - En-tête de commande CICS manquant

9 - EIB d'image ultérieure manquant

10 - Erreur CPDU de la demande

11 - Correspondance d'image dans la file d'attente de l'agent de partition logique déjà publiée

12 - CMALEYE différent de CMAL

13 - CMFREYE non CMFR

14 - CMSDEYE non CMSD

15 - CMRCEYE non CMRC

16 - CMCVEYE non CMCV

17 - CMEXEYE non CMEX

18 - La commande de réception n'a reçu aucun bloc de données.

19 - La commande de conversion n'a reçu aucun bloc de données.

20 - Bloc de données de la commande de réception trop limité

21 - Bloc de données de la commande de conversion trop limité

Action :

Pour les codes de motif 5 et 6, examinez le délai d'expiration. Pour tous les autres codes, signalez l'erreur au service de support.

ITKO0121

ITKO0121 - iTKO Agent error reading configuration (Erreur de l'agent iTKO lors de la lecture de la configuration)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

Une erreur s'est produite lors de la lecture de TKOC dans la file d'attente de données transitoires de configuration.

Action :

Vérifiez la définition et le contenu de TKOC de la file d'attente de données transitoires.

ITKO0122

ITKO0122 - Agent TCP/IP API Timeout (Expiration du délai de l'API TCP/IP de l'agent)

Type de message :

Fatal Error (Erreur irrécupérable)

Description :

L'agent CICS pour DevTest attend l'expiration de l'interface TCP/IP de CICS et l'arrêt de l'agent CICS pour DevTest.

Action :

Vérifiez que l'interface TCP/IP de systèmes de contrôle des informations client a démarré et redémarrez l'agent.

ITKO0123

ITKO0123 - Agent waiting for TCP/IP API (Agent en attente de l'API TCP/IP)

Type de message :

Warning (Avertissement)

Description :

L'agent CICS pour DevTest patiente pendant l'initialisation de l'interface TCP/IP de CICS. L'agent CICS pour DevTest effectue 30 tentatives par intervalles de 10 secondes (5 minutes). Si l'interface TCP/IP de CICS ne s'est pas initialisée avant le délai d'expiration, l'agent CICS pour DevTest se ferme.

Action :

Aucune action n'est requise.

ITKO0124

ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxx (Expiration du délai de réponse de l'agent ITKO pour le jeton xxxxxxxxxxxxxxx)

Type de message :

Warning (Avertissement)

Description :

Une image de lecture a été envoyée à l'environnement de services virtuels et aucune réponse n'a été reçue avant le délai d'expiration de 30 secondes. La demande de lecture est purgée et la commande CICS se termine avec une réponse INVREQ.

Action :

Vérifiez le VSE et recherchez des erreurs.

ITKO0125

ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 (Echec de l'appel à ITKOUEX1/ITKOUEX2/ITKOUEX3)

Type de message :

Warning (Avertissement)

Description :

Echec du lien CICS LINK vers ITKOUEX1, ITKOUEX2, ou ITKOUEX3. Consultez le message ITKO0110 associé.

Action :

Corrigez le problème identifié par le message ITKO0110.

ITKO0126

ITKO0126 - Handshake error encountered Shutting down Agent (Une erreur d'établissement de liaison s'est produite. Arrêt de l'agent)

Type de message :

Error (Erreur)

Description :

L'établissement d'une liaison entre l'agent CICS pour DevTest et l'agent de partition logique pour DevTest a échoué. Cet échec est généralement dû à un numéro de port incorrectement configuré pour l'agent de partition logique pour DevTest dans la configuration de l'agent CICS pour DevTest. L'erreur entraîne la connexion de l'agent CICS pour DevTest à un serveur qui n'est pas un agent de partition logique pour DevTest.

Action :

Corrigez la configuration de l'agent CICS pour DevTest.

ITKO0200

ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed (Echec de ITKOXEIN XPI INQ_APPLICATION_DATA)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) a reçu une erreur dans la commande XPI INQ_APPLICATION_DATA.

Action :

Signalement de l'erreur au service de support

ITKO0201

ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed (Echec du stockage dynamique avec la commande Getmain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) ou XEIOUT (module ITKOXEIO) a reçu une erreur dans la commande XPI GETMAIN lors de l'allocation de stockage dynamique.

Action :

Examinez les faibles conditions de stockage de CICS.

ITKO0202

ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed (Echec du stockage dynamique avec la commande Freemain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) ou XEIOUT (module ITKOXEIO) a reçu une erreur dans la commande XPI FREEMAIN lors de la libération de stockage dynamique.

Action :

Signalement de l'erreur au service de support

ITKO0203

ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed (Echec du stockage d'image avec la commande Getmain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) ou XEIOUT (module ITKOXEIO) a reçu une erreur dans la commande XPI GETMAIN lors de l'allocation d'une image enregistrée.

Action :

Examinez les faibles conditions de stockage de CICS.

ITKO0204

ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed (Echec du stockage d'image avec la commande Freemain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) ou XEIOUT (module ITKOXEIO) a reçu une erreur dans la commande XPI FREEMAIN lors de la libération de stockage d'image.

Action :

Signalement de l'erreur au service de support

ITKO0205

ITKOXEIN Task table full - discarding image (Table de tâche ITKOXEIN complète - rejet de l'image)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) a tenté d'allouer une entrée dans la table de tâche alors que la table est complète.

Action :

Signalement de l'erreur au service de support

ITKO0206

ITKOXEIN XPI WAIT Failed (Echec de ITKOXEIN XPI WAIT)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) a reçu une erreur dans la commande XPI WAIT pendant l'attente d'une réponse de lecture.

Action :

Signalement de l'erreur au service de support

ITKO0207

ITKOXEIN XPI Freemain plbk req/rsp failed (Echec de demande/réponse de lecture avec la commande Freemain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) a reçu une erreur dans la commande XPI FREEMAIN pendant la libération d'une réponse ou d'une demande lecture.

Action :

Examinez les faibles conditions de stockage de CICS.

ITKO0208

ITKOXEIO Add to recording queue failed (Echec de l'ajout à la file d'attente d'enregistrement)

Type de message :

Error (Erreur)

Description :

La sortie XEIOUT (module ITKOXEIO) a tenté d'ajouter une image enregistrée à la file d'attente d'enregistrement et la tentative a échoué.

Action :

Signalement de l'erreur au service de support

ITKO0209

ITKOXEIN/ITKOXEIO XPI Freemain SET() Failed (Echec du stockage SET() avec la commande Freemain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) ou la sortie XEIOUT (module ITKOXEIO) a reçu une erreur dans la commande XPI FREEMAIN pendant la libération du stockage alloué pour le paramètre SET dans une commande de réception ou de conversion.

Action :

Signalement de l'erreur au service de support

ITKO0210

ITKOXEIN XPI Getmain SET() Failed (Echec du stockage SET() avec la commande Getmain)

Type de message :

Error (Erreur)

Description :

La sortie XEIIN (module ITKOXEIN) a reçu une erreur dans la commande XPI GETMAIN pendant la libération du stockage alloué pour le paramètre SET dans une commande de réception ou de conversion.

Action :

Signalement de l'erreur au service de support

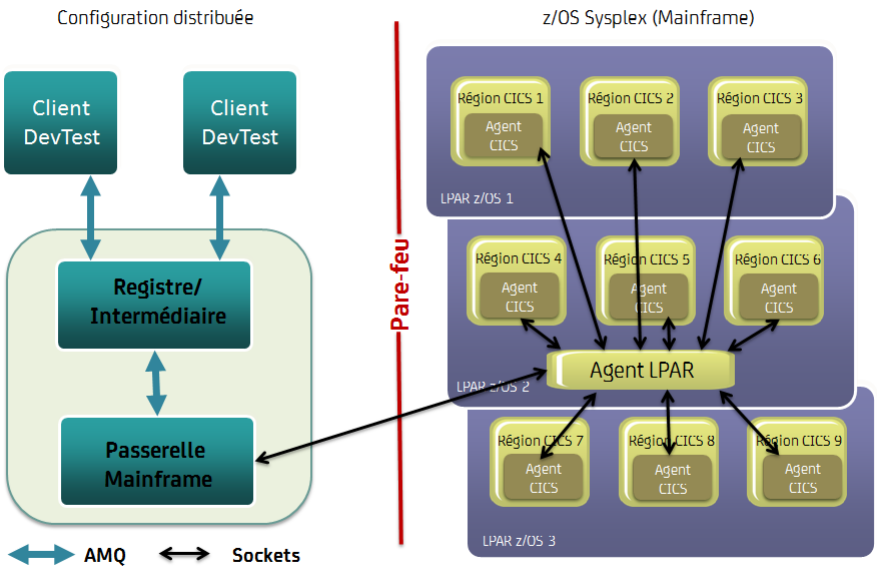
Chapitre 4: Agent de partition logique pour DevTest

L'agent de partition logique pour DevTest est un agent résidant sur z/OS exécuté comme tâche démarrée. Vous pouvez installer l'agent sur une partition logique dans le Sysplex de z/OS. L'agent fournit un point de contact unique au registre de DevTest (et par conséquent aux clients DevTest) pour communiquer avec d'autres agents z/OS pour DevTest. L'agent permet à DevTest de connaître les agents CICS z/OS en cours d'exécution et certaines informations de configuration de base sur ces agents. L'agent de partition logique permet également de grouper des agents CICS z/OS pour DevTest si plusieurs agents de partition logique de DevTest sont définis vers le Sysplex z/OS. L'agent peut fonctionner en mode client (il établit la connexion au registre de DevTest). L'agent peut également fonctionner en mode serveur (il accepte des connexions à partir de registres de DevTest).

Ce chapitre traite des sujets suivants :

Présentation de l'agent LPAR

Le diagramme suivant illustre les composants de l'agent de partition logique pour DevTest et l'agent CICS pour DevTest.



Procédure d'installation de l'agent LPAR

Le package de l'agent de partition logique comprend trois fichiers :

- iTKOLPARAgentDoc.pdf
- iTKOLPARAgentLoad
- iTKOLPARAgentCntl

Procédure de l'installation :

1. Envoi des fichiers via un site FTP vers le système z/OS cible (page 157)
2. Restauration des ensembles de données partitionnés étendus (PDSE) et de l'ensemble de données partitionné (PDS) (page 158)
3. Configuration de l'agent de partition logique pour son exécution comme tâche démarrée (page 159)
4. Création du membre de configuration de l'agent de partition logique (page 161)

Envoi des fichiers via un site FTP vers le système z/OS cible

Les fichiers iTKOLPARAgentLoad et iTKOLPARAgentCntl sont envoyés via un site FTP au système z/OS cible en mode binaire avec les attributs z/OS indiqués dans l'exemple qui suit.

L'exemple suivant de FTP de ligne de commande illustre le transfert approprié vers z/OS des ensembles de données ITKO.LPARAGNT.CNTL.UNLOAD et ITKO.LPARAGNT.LOAD.UNLOAD. Vous pouvez changer les noms de fichier z/OS selon vos conventions d'attribution de nom d'ensemble de données.

```
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=40 SEC=5
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put iTKOLPARAgentLoad 'ITKO.LPARAGNT.LOAD.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.LPARAGNT.LOAD.UNLOAD
250 Transfer completed successfully.
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
200 SITE command was accepted
ftp> put iTKOLPARAgentCntl 'ITKO.LPARAGNT.CNTL.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.LPARAGNT.CNTL.UNLOAD
250 Transfer completed successfully.
```

Restauration des PDSE et PDS

Les PDSE de la bibliothèque de chargement et le PDS de la bibliothèque de contrôle (JCL) sont restaurés à l'aide de la commande TSO RECEIVE, généralement exécutée dans un job de traitement par lots.

Dans l'exemple suivant, la JCL restaure les ensembles de données partitionnés étendus (PDSE) et l'ensemble de données partitionnés (PDS) à partir des ensembles de données envoyés via un site FTP ITKO.LPARAGNT.LOAD et ITKO.LPARAGNT.CNTL sur le volume VOL001. Vous pouvez changer les noms d'ensemble de données pour vos conventions d'attribution de nom d'ensemble de données de site.

```
//job
//*
//* Unload the LOAD Library with TSO RECEIVE
//*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('ITKO.LPARAGNT.LOAD.UNLOAD')
   DATASET('ITKO.LPARAGNT.LOAD') VOLUME(VOL001)
RECEIVE INDSNAME('ITKO.LPARAGNT.CNTL.UNLOAD')
   DATASET('ITKO.LPARAGNT.CNTL') VOLUME(VOL001)
/*
```

Configuration de l'agent de partition logique

Vous pouvez utiliser le membre LPARAGNT dans ITKO.LPARAGNT.CNTL comme modèle pour la procédure qui exécute l'agent de partition logique comme tâche démarrée. Une fois le nouveau membre de procédure créé, il doit être copié dans un PROCLIB à partir duquel les tâches peuvent être démarrées. Quatre instructions de JCL composent la procédure :

PROC :

Instruction qui spécifie le nom de la tâche démarrée

EXEC :

Instruction qui spécifie le nom et les paramètres du programme

STEPLIB :

Bibliothèque de chargement de PDSE qui contient le module de chargement de l'agent de partition logique

config :

Instruction DD qui spécifie un membre PDS dans lequel la configuration de l'agent de partition logique est définie.

La valeur PARM spécifiée dans l'instruction EXEC définit des variables d'environnement. La valeur PARM spécifie également le nom DD de l'instruction qui pointe vers le membre de données de configuration de l'agent de partition logique. Une barre oblique (/) délimite les données PARM. Les valeurs placées avant la barre oblique spécifient des options d'exécution et les valeurs placées après la barre oblique spécifient les arguments de ligne de commande pour l'application. La variable d'environnement est une option d'exécution et les données de configuration de l'agent de partition logique sont un argument de ligne de commande.

Actuellement, la seule variable d'environnement que l'agent de partition logique reconnaît est un fuseau horaire. La variable est facultative et si elle n'est pas spécifiée, elle sera définie par défaut par UTC. Cette variable est utilisée pour calculer des valeurs de date et d'heure dans les messages produits par l'agent de partition logique. Les valeurs valides sont les chaînes UNIX TZ standard ; par exemple : EST5EDT, CST6CDT, MST7MDT, PST8PDT.

L'exemple d'instruction proc suivant figure dans le membre LPARAGNT de ITKO.LPARAGNT.CNTL :

```
//LPARAGNT    PROC

//AGENT       EXEC
PGM=LPARAGNT,PARM='ENVAR("TZ=CST6CDT")/"//DD:CONFIG"'

//STEPLIB     DD DISP=SHR,DSN=ITKO.LPARAGNT.LOAD

//CONFIG      DD DISP=SHR,DSN=ITKO.LPARAGNT.CNTL(CONFIG)
```

```
//SYSUDUMP   DD SYSOUT=*
```

Création du membre de configuration de l'agent de partition logique

Les données de configuration requises par l'agent de partition logique pour DevTest sont spécifiées comme paires de mot clé-valeur en texte brut. Spécifiez le membre PDS dans une instruction DD dans la procédure utilisée pour démarrer l'agent de partition logique. Spécifiez le nom DD dans la valeur PARM de l'instruction EXEC. Le fichier est analysé comme suit :

- Un caractère de hachage (#) est utilisé comme indicateur de commentaire.
- Les lignes vides ou commençant par un caractère de hachage sont ignorées.
- Le signe égal sépare les mots clés et les valeurs.
- Les espaces sont retirés du début et de la fin des mots clés et des valeurs.
- La casse des mots clés n'est pas respectée, tandis que celle de la plupart des valeurs est conservée.
- Il existe une liste de mots clés valides comprise par l'agent de partition logique.

Les mots clés que vous pouvez utiliser pour configurer l'agent de partition logique sont les suivants :

nom

Spécifie le nom de l'agent de partition logique. Ce nom est utilisé pour des actions telles que la journalisation.

Valeur par défaut : DevTest LPAR Agent

Mode

Spécifie si l'agent de partition logique est en mode **serveur** (mot clé : Server) ou **client** (mot clé : Client). En mode serveur, l'agent accepte des connexions à partir du registre de DevTest sur le port *PortClient*. Cette valeur doit être le contraire de la propriété **lisa.mainframe.bridge.mode** dans le fichier **lisa.properties**.

PortClient

Dans le cas d'une exécution en mode serveur, ce mot clé définit la valeur du port connu auquel les clients se connectent. Ce numéro de port doit correspondre à la valeur spécifiée par la propriété **lisa.mainframe.bridge.server.port** dans le fichier **lisa.properties**.

Valeur par défaut : 3997

Serveur

Dans le cas d'une exécution en mode client, ce mot clé spécifie l'adresse IP du serveur auquel se connecter. Cette adresse IP correspond généralement à l'adresse IP de l'hôte de DevTest Server exécuté dans le registre.

ServerPort

Dans le cas d'une exécution en mode client, ce mot clé définit le numéro de port utilisé par le registre du DevTest Server. Ce port doit correspondre à la valeur spécifiée par la propriété **lisa.mainframe.bridge.port** dans le fichier **lisa.properties**.

PortAgent

Définit la valeur du port connu auquel les agents CICS se connectent. Ce numéro de port doit correspondre à la valeur configurée par tous les agents CICS qui se connectent à cet agent de partition logique.

Valeur par défaut : 3998

InitialTrace

Spécifie si le suivi des paquets est initialement activé ou désactivé.

Valeurs :

- **True :** Le suivi est activé pour toutes les connexions.
- **False :** Le suivi est désactivé pour toutes les connexions.

Limites : Ces valeurs ne respectent pas la casse.

Valeur par défaut : False

BufferSize

Taille de tampon initiale allouée. Cette valeur est généralement définie sur deux fois la taille de la charge utile de données attendue la plus élevée. Vous pouvez réallouer la taille de tampon jusqu'à la valeur maximum configurée lorsque des paquets d'une taille supérieure que cette valeur sont reçus.

Valeur par défaut : 65536

MaxBufferSize

Taille de paquet maximum à accepter. Un paquet supérieur à cette valeur sera rejeté et un message sera journalisé. La valeur zéro désactive la vérification de la taille et tous les paquets à partir d'un client seront donc acceptés.

Valeur par défaut : 0

L'exemple de fichier de configuration suivant pour une exécution en mode **serveur** est disponible dans le membre CONFIGSV member de ITKO.LPARAGNT.CNTL :

```
#
# A sample server mode configuration file
#
Name             = DevTest LPAR Agent


#
# Mode can be "Client" or "Server"
#
Mode             = Server
```

```
PortClient       = 3997
#

#
# The port used as a server for the agent connections
#
PortAgent        = 3998

#
BufferSize       = 65536
MaxBufferSize    = 65536
#
```

L'exemple de fichier de configuration suivant pour une exécution en mode **client** est disponible dans le membre CONFIGCL member de ITKO.LPARAGNT.CNTL :

```
#
# A sample client mode configuration file
#
Name              = DevTest LPAR Agent

#
# Mode can be "Client" or "Server"
#
Mode              = Client
Server            = 10.0.0.1
ServerPort        = 61617

#
# The port used as a server for the agent connections
```

```
#
PortAgent          = 3998


#
BufferSize         = 65536
MaxBufferSize      = 65536
#
```

Remarques sur le stockage de l'agent de partition logique

L'agent de partition logique pour DevTest est exécuté en tant qu'espace d'adressage autonome avec une région minimum de 3 Mo. Une région supérieure peut être requise si les images enregistrées sont excessivement grandes ou si le taux d'arrivée est élevé.

Opération d'agent LPAR

Démarrage de l'agent LPAR pour DevTest

Dans une opération ordinaire, l'agent de partition logique est configuré comme tâche démarrée et démarre en même temps que les IPL (procédures de chargement initial) de partition logique. Vous pouvez utiliser la commande d'opérateur standard MVS START (S) pour démarrer l'agent de partition logique de manière explicite.

Arrêt de l'agent LPAR pour DevTest

L'agent de partition logique répond à la commande d'opérateur standard MVS STOP (P). Vous pouvez également arrêter l'agent en émettant le message SHUTDOWN (Arrêt) via la connexion de socket de gestion ou la commande d'opérateur MVS MODIFY (F). L'agent de partition logique se termine de manière identique pour chaque technique.

Surveillance et gestion de l'agent de partition logique

Vous pouvez surveiller l'agent de partition logique et le gérer dans une mesure limitée via des commandes d'opérateur MVS standard.

Pour utiliser des commandes d'opérateur MVS standard, émettez les demandes vers l'agent de partition logique à l'aide de la commande MODIFY (F). La réponse est écrite dans le journal de la console. Les commandes de l'agent de partition logique prises en charge sont les suivantes :

?

Affiche une liste de commandes disponibles.

Syntaxe : F, LPARAGN,?

L

Répertorie les connexions actuelles et les ID de connexion associés.

Syntaxe : F LPARAGNT,L

T

Bascule le suivi de paquet.

Syntaxe : F LPARAGNT,T <id> <state>

<id>

Définit l'ID de connexion présente dans la commande "L".

<state>

(Facultatif) Spécifie l'état du suivie de paquet pour une connexion.

L'émission de la commande T sans l'état renvoie l'état du suivi pour cette connexion.

Valeur par défaut : ON et OFF

SHUTDOWN

Arrête l'Agent de partition logique.

Syntaxe : F LPARAGNT,SHUTDOWN

La sortie suivante est un exemple de commande **L** :

```
F LPARAGNT,L

+ITKO9003 - Command accepted

+ID    Type IP              Port         In       Out  Connected

+3    AT   192,168.0,100   6891        1        0  15/12/11 09:38:44

+4    A    192,168.0,100   6892        1        0  15/12/11 09:38:45
```

Le type est **A** pour agent connection (connexion d'agent) ou **C** pour client connection (connexion cliente). Si un **T** suit le **A** ou **C**, cela signifie que le suivi de paquets est activé pour cette connexion. L'adresse IP et le numéro de port identifient la source de la connexion. In et Out indique le nombre de paquets reçus à partir de et envoyés vers l'homologue, respectivement. Les date et heure Connected indiquent la date et l'heure de la création de la connexion. Les date et heure Connected utilisent la valeur de fuseau horaire spécifiée dans la valeur PARM de l'instruction proc EXEC.

La sortie suivante est un exemple de commande T :

```
F LPARAGNT,T 3

+ITKO9003 - Command accepted

+Tracing for ID 3 is ON

F LPARAGNT,T 3 OFF

+ITKO9003 - Command accepted

+Tracing for ID 3 is now OFF
```

Messages de l'agent LPAR

Les messages sont écrits dans la console d'opérateur et vers des ensembles de données dans le journal JES pour la tâche démarrée.

Les messages suivants peuvent être écrits dans la console d'opérateur. Ces messages sont fournis à titre d'information uniquement.

- ITKO9001 - processus de l'agent de partition logique initialisé et en attente de connexions
- ITKO9002 - commande d'opérateur trop longue
- ITKO9003 - commande acceptée
- ITKO9004 - commande Start acceptée
- ITKO9005 - commande Stop acceptée
- ITKO9006 - finalisation du processus de l'agent de partition logique

Exemple de journal JES (DD SYS00001 de la tâche démarrée) :

```
2014:10:09 19:05:38 - Starting via operator command.

2014:10:09 19:05:38 - LPAR Agent started. Version: V800  Compiled: Feb
7 2014 13:08:00

2014:10:09 19:05:38 - Using configuration file: //DD:CONFIG

2014:10:09 19:05:38 - ---: Name             = CA DevTest LPAR Agent

2014:10:09 19:05:38 - ---: InitialTrace     = False

2014:10:09 19:05:38 - ---: Mode             = Server

2014:10:09 19:05:38 - ---: PortClient       = 4997

2014:10:09 19:05:38 - ---: PortAgent        = 3998

2014:10:09 19:05:38 - ---: BufferSize       = 65536

2014:10:09 19:05:38 - ---: MaxBufferSize    = 65536

2014:10:09 19:05:38 - ---: AppKASeconds     = 60

2014:10:09 19:05:38 - ---: AppKALimit       = 5

2014:10:09 19:05:38 - Configuration file processing successful.

2014:10:09 19:05:38 - LPAR Agent named CA DevTest LPAR Agent

2014:10:09 19:05:38 - Agent socket ready.

2014:10:09 19:05:38 - Client socket ready.
```

```
2014:10:09 19:05:38 - Memory for message buffers allocated.

2014:10:09 19:05:38 - Known interfaces: 192,86.32,105

2014:10:09 19:05:38 - Initial configuration complete.

2014:10:09 19:06:45 - ID: 2 - Accepted agent connection from
192.168.0.100.

2014:10:09 19:06:48 - ID: 3 - Accepted agent connection from
192.168.0.100.

2014:10:09 19:06:48 - ID: 4 - Accepted client connection from
192.168.0.101.
```

Glossaire

Archive de modèle (MAR)

Une *archive de modèle (MAR)* est le principal artefact de déploiement dans DevTest. Les fichiers MAR contiennent un actif principal, tous les fichiers secondaires qui sont requis pour exécuter l'actif principal, un fichier d'informations et un fichier d'audit. Pour plus d'informations, consultez la section Utilisation des archives de modèle (MAR) de la rubrique *Utilisation de CA Application Test*.

assertion

Une *assertion* est un élément qui s'exécute après l'exécution d'une étape et de tous ses filtres. Les assertions vérifient que les résultats de l'étape sont conformes aux prévisions. Une assertion est généralement utilisée pour modifier le flux d'un scénario de test ou le modèle de service virtuel. Les assertions globales s'appliquent à chaque étape d'un scénario de test ou d'un modèle de service virtuel. Pour plus d'informations, consultez la section Assertions de la rubrique *Utilisation de CA Application Test*.

Audit document (Document d'audit)

Un *document d'audit* permet de définir des critères de réussite d'un test, ou d'un ensemble de tests dans une suite. Pour plus d'informations, consultez la section Génération de documents d'audit dans la rubrique *Utilisation de CA Application Test.*

companion (compagnon)

Un *Companion (Compagnon)* est un élément exécuté avant et après chaque exécution de scénario de test. Les compagnons sont comparables à des filtres applicables à l'ensemble du scénario de test, par opposition à des étapes de test spécifiques. Les compagnons sont utilisés pour configurer le comportement global dans le scénario de test. Pour plus d'informations, consultez la section Compagnons de la rubrique *Utilisation de CA Application Test*.

Configuration

Une *configuration* est une collection nommée de propriétés qui spécifient en général des valeurs propres à un environnement pour le système testé. La suppression de données d'environnement codées de manière irréversible permet d'exécuter un scénario de test ou un modèle de service virtuel au niveau d'environnements différents en modifiant simplement des configurations. La configuration par défaut dans un projet est appelée project.config. Un projet peut contenir de nombreuses configurations, mais une seule configuration peut être active à la fois. Pour plus d'informations, consultez la section Configurations de la rubrique *Utilisation de CA Application Test*.

Conversation tree (Arborescence des conversations)

Une *arborescence des conversations* est un ensemble de noeuds liés qui représentent des chemins de conversation pour les transactions avec état dans une image de service virtuel. Chaque noeud porte une étiquette de nom d'opération ; par exemple : withdrawMoney. Exemple de chemin de conversation pour un système d'opérations bancaires : getNewToken, getAccount, withdrawMoney, deleteToken. Pour plus d'informations, reportez-vous à la rubrique *Utilisation de CA Service Virtualization.*

coordinator (coordinateur)

Le coordinateur reçoit les informations d'exécution de test sous forme de document et coordonne les tests exécutés sur un ou plusieurs serveurs de simulation. Pour plus d'informations, consultez la section Coordinator Server (Serveur de coordination) de la rubrique *Utilisation de CA Application Test*.

data protocol (protocole de données)

Un *protocole de données* est également appelé gestionnaire de données. Dans CA Service Virtualization, un protocole de données est responsable de la gestion de l'analyse des demandes. Certains protocoles de transport autorisent (ou requièrent) un protocole de données auquel le job de création de demandes est délégué. C'est pourquoi le protocole doit connaître la charge utile de la demande. Pour plus d'informations, consultez la section Utilisation de protocoles de données dans la rubrique *Utilisation de CA Service Virtualization*.

Data set (Ensemble de données)

Un *Data set (Ensemble de données)* est une collection de valeurs que vous pouvez utiliser pour définir des propriétés dans un scénario de test ou un modèle de service virtuel lors de l'exécution. Les ensembles de données fournissent un mécanisme permettant d'introduire des données de test externes dans un scénario de test ou un modèle de service virtuel. Vous pouvez créer des ensembles de données internes à DevTest, ou de manière externe ; par exemple, dans un fichier ou une table de base de données. Pour plus d'informations, consultez la section Ensembles de données de la rubrique *Utilisation de CA Application Test*.

desensitize (désensibiliser)

La *désensibilisation* consiste à convertir des données sensibles par des valeurs de substitution définies par l'utilisateur. Par exemple, les numéros de carte de crédit et les numéros de sécurité sociale sont des données sensibles. Pour plus d'informations, consultez la section Désensibilisation de données de la rubrique *Utilisation de CA Service Virtualization*.

Event (Evénement)

Un *Event (Evénement)* est un message sur une action qui s'est produite. Vous pouvez configurer des événements au niveau d'un scénario de test ou d'un modèle de service virtuel. Pour plus d'informations, consultez la section Introduction aux événements de la rubrique *Utilisation de CA Application Test*.

Filter (Filtre)

Un *filtre* est un élément exécuté avant et après une étape. Les filtres permettent de traiter les données des résultats ou de stocker des valeurs dans des propriétés. Les filtres globaux s'appliquent à chaque étape d'un scénario de test ou d'un modèle de service virtuel. Pour plus d'informations, consultez la section Filtres de la rubrique *Utilisation de CA Application Test*.

Groupe

Un *groupe* ou un *groupe de services virtuels* est une collection de services virtuels portant la même balise de groupe. Cela permet de les surveiller ensemble dans la console VSE.

Interactive Test Run (ITR) (Exécuter un test interactif)

L'utilitaire *Interactive Test Run (ITR)* (Exécuter un test interactif) permet d'exécuter un scénario de test ou un modèle de service virtuel étape par étape. Vous pouvez changer le scénario de test ou le modèle de service virtuel lors de l'exécution et le réexécuter pour vérifier les résultats. Pour plus d'informations, consultez la section Utilisation de l'utilitaire Interactive Test Run (ITR) (Exécuter un test interactif) de la rubrique *Utilisation de CA Application Test.*

Lab (Laboratoire)

Un *laboratoire* est un conteneur logique pour un ou plusieurs membres de laboratoire. Pour plus d'informations, consultez la section Laboratoires et membres de Laboratoire dans la rubrique *Utilisation de CA Application Test.*

magic date (date magique)

Pendant un enregistrement, un analyseur de dates analyse les demandes et les réponses. Une valeur correspondant à une définition étendue de formats de la date est convertie en *date magique*. Les dates magiques permettent de vérifier que le modèle de service virtuel fournit des valeurs de date explicites dans des réponses. Exemple e date magique : {{=doDateDeltaFromCurrent("yyyy-MM-dd","10");/*2012-08-14*/}. Pour plus d'informations, consultez la section Chaînes et dates magiques de la rubrique *Utilisation de CA Service Virtualization.*

Magic string (Chaîne magique)

Une *chaîne magique* est une chaîne générée pendant la création d'une image de service. Une chaîne magique est utilisée pour vérifier que les réponses fournies par le modèle de service virtuel contiennent des valeurs de chaîne explicites. Exemple de chaîne magique : {{=request_fname;/chris/}}. Pour plus d'informations, consultez la section Chaînes et dates magiques de la rubrique *Utilisation de CA Service Virtualization.*

Match tolerance (Tolérance de correspondance)

La tolérance de correspondance est un paramètre qui permet de contrôler la méthode utilisée par CA Service Virtualization pour comparer une demande entrante avec les demandes dans une image de service. Les options disponibles sont EXACT, SIGNATURE et OPERATION. Pour plus d'informations, consultez la section Tolérance de correspondance de la rubrique *Utilisation de CA Service Virtualization*.

Metrics (Mesures)

Les mesures permettent d'appliquer des méthodes et des mesures quantitatives aux performances et aux aspects fonctionnels de vos tests, ainsi qu'au système testé. Pour plus d'informations, consultez la section Génération de mesures de la rubrique *Utilisation de CA Application Test*.

Model Archive (MAR) Info (Fichier d'informations d'archive de modèle (MAR))

Un *Model Archive (MAR) Info (Fichier d'informations d'archive de modèle (MAR))* est un fichier qui contient des informations requises pour la création d'une archive de modèle MAR. Pour plus d'informations, consultez la section Utilisation des archives de modèle (MAR) de la rubrique *Utilisation de CA Application Test*.

navigation tolerance (tolérance de navigation)

La tolérance de navigation est un paramètre qui permet de contrôler le méthode utilisée par CA Service Virtualization pour rechercher la transaction suivante dans une arborescence des conversations. Les options disponibles sont CLOSE, WIDE et LOOSE. Pour plus d'informations, consultez la section Tolérance de navigation de la rubrique *Utilisation de CA Service Virtualization*.

Network graph (Graphique du réseau)

Le graphique de réseau est une zone de la console de serveur qui contient une représentation graphique du composant DevTest Cloud Manager et des laboratoires associés. Pour plus d'informations, consultez la section Démarrage d'un laboratoire de la rubrique *Utilisation de CA Application Test.*

Node (Noeud)

Une étape de test interne à DevTest peut également être appelée *node* (noeud), ce qui explique l'intégration du terme node dans l'ID de certains événements.

Path (Chemin)

Un *chemin* contient des informations sur une transaction capturée par l'agent Java. Pour plus d'informations, reportez-vous à la rubrique *Utilisation de CA Continuous Application Insight*.

Path graph (Graphique de chemin)

Un *graphique de chemin* contient une représentation graphique d'un chemin et ses trames. Pour plus d'informations, consultez la section Graphique de chemin de la rubrique *Utilisation de CA Continuous Application Insight*.

Project (Projet)

Un *projet* est une collection de fichiers DevTest liés. Les fichiers peuvent inclure des scénarios de test, des suites, des modèles de service virtuel, des images de service, des configurations, des documents d'audit, des documents de simulation, des ensembles de données, des moniteurs et des fichiers d'informations MAR. Pour plus d'informations, consultez la section Panneau Project (Projet) de la rubrique *Utilisation de CA Application Test*.

Property (Propriété)

Une *propriété* est une paire clé-valeur que vous pouvez utiliser comme variable d'exécution. Les propriétés peuvent stocker plusieurs types de données différents. Exemple de propriétés communes : LISA_HOME, LISA_PROJ_ROOT et LISA_PROJ_NAME. Une configuration est une collection nommée de propriétés. Pour plus d'informations, consultez la section Propriétés de la rubrique *Utilisation de CA Application Test*.

quick test (test rapide)

La fonctionnalité *Quick test (Test rapide)* permet d'exécuter un scénario de test avec une installation minimale. Pour plus d'informations, consultez la section Simulation d'un test rapide du *Utilisation de CA Application Test.*

Registry (Registre)

Le *registre* fournit un emplacement central pour l'enregistrement de tous les composants de DevTest Server et de DevTest Workstation. Pour plus d'informations, consultez la section Registre de la rubrique *Utilisation de CA Application Test*.

ressource

Un *actif* est un ensemble de propriétés de configuration groupées dans une unité logique. Pour plus d'informations, consultez la section Actifs de la rubrique *Utilisation de CA Application Test*.

service image (image de service)

Une *image de service* est une version normalisée de transactions enregistrées dans CA Service Virtualization. Chaque transaction peut être avec état (conversationnel) ou sans état. Une image de service peut être créée à l'aide de l'enregistreur d'image de service virtuel. Les images de service sont stockées dans un projet. Une image de service est également appelée *virtual service image (image de service virtuel, VSI)*. Pour plus d'informations, consultez la section Images de service de la rubrique *Utilisation de CA Service Virtualization*.

Simulator (Simulateur)

Un *simulateur* exécute les tests sous la surveillance du serveur de coordination. Pour plus d'informations, consultez la section Simulator Server (Serveur de simulation) de la rubrique *Utilisation de CA Application Test*.

Staging document (Document de simulation)

Un *document de simulation* contient les informations sur l'exécution d'un scénario de test. Pour plus d'informations, consultez la section Génération de documents de simulation dans la rubrique *Utilisation de CA Application Test.*

Subprocess (Sous-processus)

Un *sous-processus* est un scénario de test appelé par un autre scénario de test. Pour plus d'informations, consultez la section Génération de sous-processus de la rubrique *Utilisation de CA Application Test.*

Tableau de bord Continuous Service Validation (CVS) (Service de validation en continu)

Le *Continuous Service Validation (CVS) Dashboard (Tableau de bord du service de validation en continu)* permet de planifier l'exécution régulière de scénarios de test et des suites de test, sur une période étendue. Pour plus d'informations, consultez la section Service de validation en continu dans la rubrique *Utilisation de CA Application Test.*

test case (scénario de test)

Un *scénario de test* est une spécification de la procédure de test d'un composant métier dans le système testé. Chaque scénario de test contient une ou plusieurs étapes de test. Pour plus d'informations, consultez la section Génération de scénarios de test dans la rubrique *Utilisation de CA Application Test.*

test step (étape de test)

Une *étape de test* est un élément du flux de travaux de scénario de test qui représente une action de test unique à réaliser. Exemples d'étapes : Services Web, JavaBeans, JDBC et messagerie JMS. Une étape de test peut contenir des éléments de DevTest, tels que des filtres, des assertions et des ensembles de données liés. Pour plus d'informations, consultez la section Génération d'étapes de test de la rubrique *Utilisation de CA Application Test.*

Test suite (Suite de tests)

Une *suite de tests* est un groupe de scénarios de test, d'autres suites de tests, ou les deux, planifiés pour être exécutés l'un après l'autre. Un document de suite de tests spécifie le contenu de la suite, les rapports à générer et les mesures à collecter. Pour plus d'informations, consultez la section Génération de suites de tests de la rubrique *Utilisation de CA Application Test.*

Think time (Délai de réflexion)

Le temps de réflexion est la durée d'attente d'un scénario de test patiente avant d'exécuter une étape de test. Pour plus d'informations, consultez les sections Exemple d'ajout d'étape de test et Editeur de documents de simulation - Onglet Base de la rubrique *Utilisation de CA Application Test.*

transaction frame (Trame de transaction)

Une *trame de transaction* contient des données sur un appel de méthode intercepté par l'agent Java de DevTest ou un agent Light de CAI. Pour plus d'informations, consultez la section Transactions organisationnelles et trames de transaction de la rubrique *Utilisation de CA Continuous Application Insight.*

Virtual Service Environment (Environnement de service virtuel, VSE)

Le *Virtual Service Environment* (Environnement de service virtuel, VSE) est une application de DevTest Server qui permet de déployer et d'exécuter des modèles de service virtuel. VSE est également appelé CA Service Virtualization. Pour plus d'informations, reportez-vous à la rubrique *Utilisation de CA Service Virtualization*.

virtual service model (VSM, modèle de service virtuel)

Un *modèle de service virtuel* reçoit des demandes de service auxquelles il répond en l'absence du fournisseur de services réel. Pour plus d'informations, consultez la section Modèle de service virtuel de la rubrique *Utilisation de CA Service Virtualization*.