DevTest Solutions

Agents

バージョン 8.0

CA への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

目次

第 3 章: DevTest CICS エージェント 109

第 4 章: DevTest LPAR エージェント 171

用語集 187

第 1 章: DevTest Java エージェント

DevTest Java エージェントは、任意の Java プロセス (Java EE コンテナなど) の内部にインストールできるサーバ側テクノロジの 1 つです。エージェントによって、DevTest はサーバ側のアクティビティを制御およびモニタできます。

エージェントは、ほとんどのプロファイラが実行することを実行できます。たとえば、ロードされたクラス/オブジェクト、CPU 使用率、メモリ使用率、スレッドのモニタ、メソッド コールの追跡などです。 ただし、エージェントは複数の JVM にわたって動作し、DevTest と一緒に使用することでテストに独自の機能をもたらします。

たとえば、テストまたはテスト ステップを実行することでバックグラウンドで発生するサーバのアクションを可視化することができます。 この機能は、バグおよびボトルネックを特定するのに役立ちます。 この機能は、CA Continuous Application Insight が実行するものに似ています。 ただし、この機能は、コードまたは設定ファイルをインスツルメントする必要なしに、Java アプリケーションが使用するすべてのプロトコルにわたって動作します。

また、エージェントは CA Service Virtualization をサポートしています。エージェントは、プロトコル全体のトラフィックおよびメソッド コールの記録および再生を可能にします。エージェントは、仮想化するターゲット アプリケーションの領域に対する完全な制御を CA Service Virtualization に与えます。 エージェントは、プロトコルにかかわらずこの機能を実現する統合フレームワークを提供します。

このセクションには、以下のトピックが含まれています。

Java エージェントのアーキテクチャ

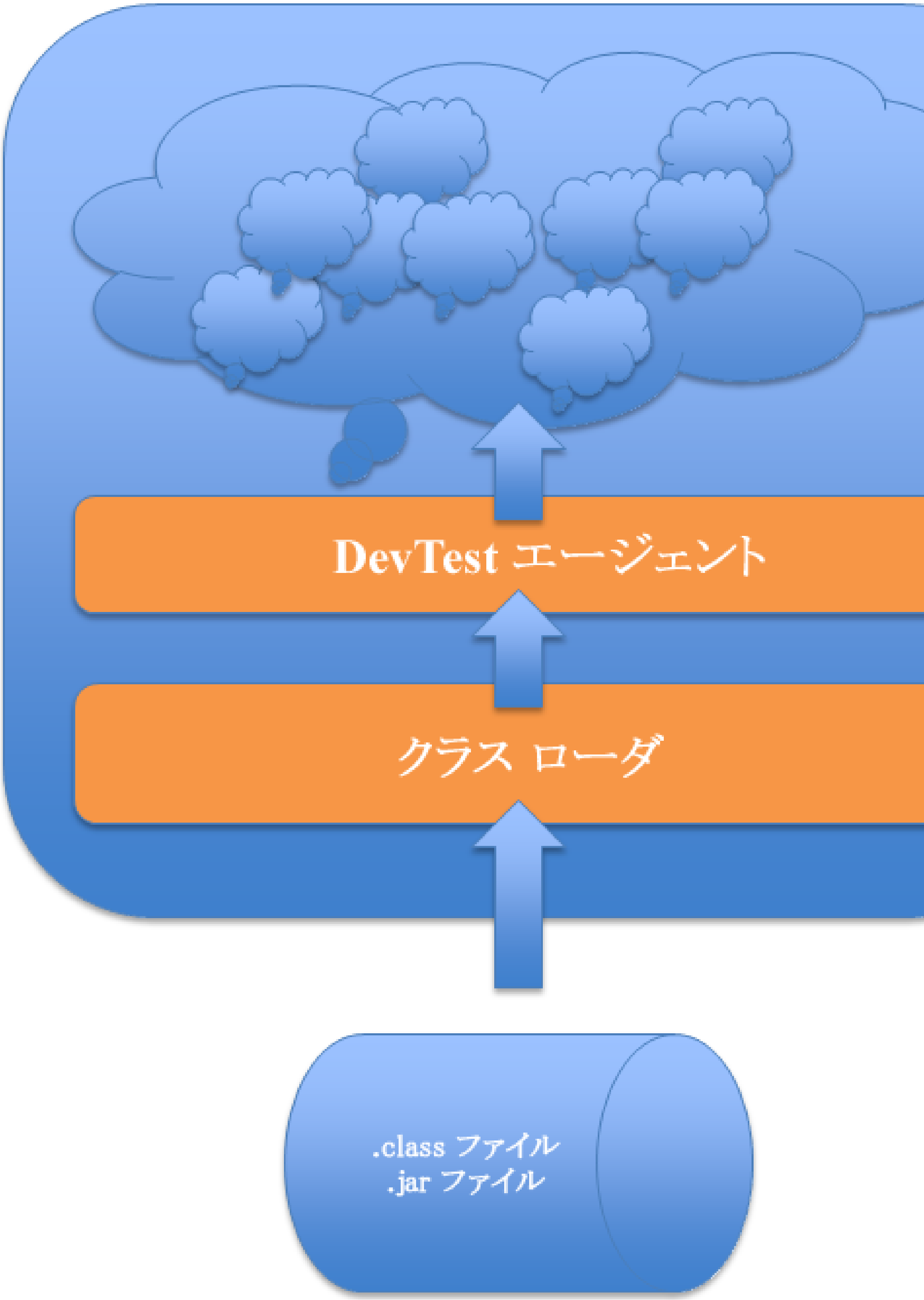
このセクションには、以下のトピックが含まれます。

Java エージェントのテクノロジ

DevTest Java エージェントは Java エージェントの一種です。

Java エージェントは、Java 仮想マシン（JVM）に組み込まれているプログラムです。 これらのプログラムは、実行中のアプリケーションに関する情報の収集、アプリケーションの一部の仮想化などの多くの機能をサポートするように設計できます。

以下の図の大きな青い枠は、JVM を表しています。JVM にはエージェントおよびクラス ローダが含まれます。クラス ローダは実行時に Java クラス ファイルをロードします。 雲は、Java アプリケーションがロードするリソースを表しています。


DevTest エージェント
クラス ローダ
.class ファイル
.jar ファイル

エージェントは JAR ファイルにパッケージ化されています。 JVM は JAR ファイルへのパスを使用して設定する必要があります。 Java 1.5 の時点では、パスおよび任意のオプションを指定するために **-javaagent** で始まる文字列を使用します。 以下に例を示します。

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

単一の JVM に複数のエージェントを含めることができます。

Java エージェント コンポーネント

DevTest Java エージェントには、以下のコンポーネントがあります。

- エージェント自体（Java プロセスに組み込まれて実行されます）
- ブローカ
- コンソール（クライアント）
- データベース

ブローカは、エージェント、コンソール、および組み込みのクライアントの間で Java Message Service （JMS）メッセージをディスパッチするハブです。

組み込みのクライアントは、エージェントのセット、それらの開いているキュー、現在のネットワーク接続など、ネットワーク全体/エージェントにまたがるプロパティを追跡します。 そのため、組み込みのクライアントは、部分的なトランザクション データを完全なトランザクションにアセンブルして、コンソールで利用できるようにします。

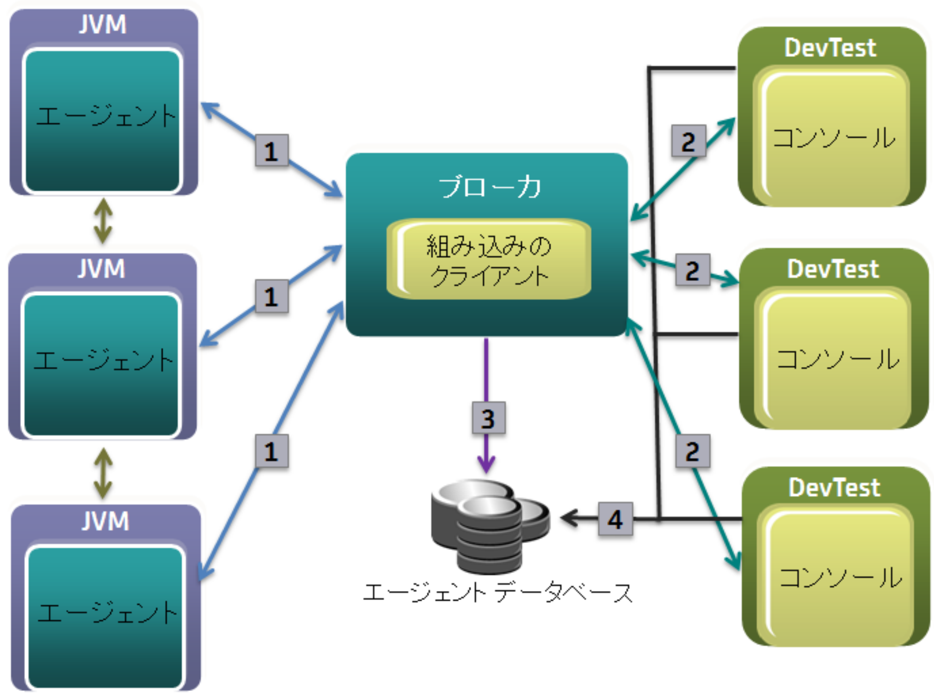
データはアセンブルされた後、コンソールに送信され（短期）、長期保管するためにデータベースに保持されます。

このドキュメントでは、区別せずに、組み込みのクライアントをブローカと呼びます。

コンソールは、表示およびユーザの操作のためにブローカ（最近の場合）およびデータベース（より古い場合）からそれらの最終データを取得します。コンソールは GUI コンポーネントである必要はありません。コンソールには DevTest ワークステーション、VSE、およびシミュレータが含まれます。

Java エージェントのデータ フロー

以下の図は、Java エージェント コンポーネントおよびそれらのインタラクションを示しています。



以下の JMS 宛先が、データのフローを指定します。

- **lisa.agent.info** トピックは接続 1 および 2 上に送信されます。このトピックは、エージェントによって作成され、ブローカおよびコンソールによって使用されます。 このトピックによって、ブローカおよびコンソールは、現在オンラインのエージェントとその基本的なプロパティを確認できます。
- **lisa.agent.port** トピックは接続 1 上に送信されます。このトピックは、エージェントによって作成され、ブローカによって使用されます。 このトピックによって、ブローカは、複数のエージェント間で現在アクティブである接続を確認できます。
- **lisa.agent.api** トピックは接続 1 および 2 上に送信されます。 このトピックは、コンソールによって作成され、エージェントによって使用（およびエージェントに応答）されます。 このトピックによって、コンソールは JMS を介してエージェント API を呼び出すことができます。

- **lisa.broker.api** トピックは接続 2 上に送信されます。このトピックは、コンソールによって作成され、ブローカによって使用（およびブローカに応答）されます。 このトピックによって、コンソールは JMS を介してブローカ API を呼び出すことができます。
- **lisa.stats** トピックは接続 1 および 2 上に送信されます。このトピックは、エージェントによって作成され、ブローカおよびコンソールによって使用されます。 このトピックによって、エージェントの現在のロード タイプがコンソールに通知されます。また、このトピックによってブローカはそのデータをデータベースに保持できます。
- **lisa.vse** トピックは接続 1 および 2 上に送信されます。このトピックは、エージェントによって作成され、コンソールによって使用されます。VSE が有効になっているとき、コンソールは VSE フレームを受信します（再生モードでは、それらに応答します）。
- **lisa.tx.partial** キューは接続 1 上に送信されます。このキューは、エージェントによって作成され、ブローカによって使用されます。 エージェントは部分的なトランザクション（その JVM で発生するすべてのフレーム）をキャプチャすると、それをアセンブリのためにブローカに送信します。
- **lisa.tx.full** トピックは接続 2 上に送信されます。このトピックは、ブローカによって作成され、コンソールによって使用されます。 ブローカは **lisa.tx.partial** を介して受信した部分的なトランザクションのアセンブリを完了すると、コンソールに完全なトランザクションを送信します。
- **lisa.tx.incomplete** トピックは接続 2 上に送信されます。このトピックは、ブローカによって作成され、コンソールによって使用されます。このトピックは **lisa.tx.full** に似ていますが、許可されているタイムアウト内に完了できないトランザクションに使用されます。
- **JDBC** 接続 3 は、ブローカが **StatsFrame** オブジェクトを保存するか、**TransactionFrame** オブジェクトを完全にアセンブルした場合に使用されます。
- コンソールは、メモリ内に保持されなくなったトランザクションまたは統計のクエリを実行するために **JDBC** 接続 4 を使用します。

すべての通信は非同期で、自動的に再確立できるため、エージェント、ブローカ、およびコンソールの起動順序は問題にはなりません。 この概念は、ブローカがダウンしていることが原因で発生するパフォーマンスの問題をエージェントが回避するためなどに特に重要です。

エージェントはオンラインになると、一定の短い間隔で **lisa.agent.info** トピックを介して情報を送り始めます。

ブローカが使用可能でない場合、エージェントは接続が確立または再確立されるまでその他のリスナに何も通知しようとしません。

ブローカが使用可能な場合、すべての通知先にオンラインのエージェントが迅速に通知されます。ブローカおよびコンソールは、それらのエージェントの実行リストを保持します。 情報の送信を停止するとエージェントは期限切れになり、しばらくしてリストから削除されます。

Java エージェントのデータベース スキーマ

ブローカは、DevTest Java エージェントのデータベース スキーマを自動的に作成します。

スキーマを手動で作成するには、以下のコマンドを実行して DDL ステートメントを取得します。

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

LISA_HOME¥database ディレクトリ内の以下のファイルから DDL ステートメントを取得することもできます。

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

通常は、セキュリティまたはプロセスに関する理由、あるいはマイグレーションまたはドキュメントに関する懸念からスキーマを手動で作成します。

Java エージェントのデータ カテゴリ

DevTest Java エージェントは、以下のカテゴリのデータをキャプチャできます。

- クライアント
- EJB
- GUI （AWT、Swing、SWT）
- JCA
- JDBC
- JMS
- ログ
- REST
- RMI
- SAP
- スレッド： スレッド全体のステッチングを示す合成フレーム
- スローアブル
- TIBCO ActiveMatrix BusinessWorks
- webMethods Integration Server
- WebSphere MQ
- Web （HTTP/S）
- Web サービス

Java エージェントのインストール

このセクションでは、DevTest Java エージェントをインストールする方法について説明します。

DevTest Java エージェントには 2 つのバージョンがあります。

- Pure Java エージェント
- ネイティブ エージェント

Pure Java エージェントには、プラットフォーム依存のコードはありません。

ネイティブ エージェントは、プラットフォーム依存のライブラリ モジュールを必要とします。

重要: TIBCO BusinessWorks ベースラインを作成する必要がない場合は、Pure Java エージェントをインストールします。 この機能はネイティブ エージェントを必要とします。

このセクションには、以下のトピックが含まれます。

Pure Java エージェントのインストール

このトピックでは、DevTest Java エージェントの Pure Java バージョンをインストールする方法について説明します。

注: ターゲット環境を事前に調査し、それがテストされていることを確認するか、または自分自身でテストしてください。Java エージェントの問題のほとんどは、アプリケーションではなく、OS または JVM に起因しています。 このドキュメントの指示に従うようにしてください。 それでも解決できない場合は、サポートに問い合わせる前に「Java エージェントのトラブルシューティング (P. 92)」を確認してください。

次の手順に従ってください:

1. **LISA_HOME¥agent** ディレクトリから以下のファイルを取得します。
   - **LisaAgent.jar**
   - **InsightAgent.jar**（Oracle WebLogic Server 以外のすべてのプラットフォームで）
   - **LisaAgent2.jar**（Oracle WebLogic Server で）
2. Java アプリケーションからの読み取り権限があり、自動的にアプリケーション クラスパス（**¥WEB-INF¥lib** ディレクトリなど）でロードされないディスク上の任意の場所にファイルを配置します。
3. プラットフォーム固有のエージェント インストール (P. 25) セクションに移動し、使用しているプラットフォーム用の手順に従います。

   Oracle WebLogic Server 以外のすべてのプラットフォームについては、以下の形式でエージェント パラメータ文字列を指定します。

   ```
   -javaagent:<path_to_InsightAgent.jar>[=url=<broker_url>][,name=<agent_name>]
   ```

   Oracle WebLogic Server については、以下の形式でエージェント パラメータ文字列を指定します。

   ```
   -javaagent:<path_to_LisaAgent2.jar>[=url=<broker_url>][,name=<agent_name>]
   ```

4. Java アプリケーションを起動します。

ネイティブ エージェントのインストール

このトピックでは、DevTest Java エージェントのネイティブ バージョンをインストールする方法について説明します。

注: ターゲット環境を事前に調査し、それがテストされていることを確認するか、または自分自身でテストしてください。Java エージェントの問題のほとんどは、アプリケーションではなく、OS または JVM に起因しています。 このドキュメントの指示に従うようにしてください。 それでも解決できない場合は、サポートに問い合わせる前に「Java エージェントのトラブルシューティング (P. 92)」を確認してください。

TIBCO BusinessWorks ベースラインを作成する場合、ネイティブ エージェントをインストールする必要があります。 また、エージェント パラメータ文字列に **heap** オプションを追加し、値を true に設定します。

以下の表に、ネイティブ エージェント用のプラットフォーム依存のライブラリ モジュールのファイル名を示します。

モジュール	ファイル名
Windows（32 ビット JVM）	**JavaBinder.dll**
Windows（64 ビット JVM）	**JavaBinder.amd64.dll**
Mac OS X（x86/x64）32/64 ビット	**libJavaBinder.jnilib**
Linux（x86）32 ビット	**libJavaBinder.x86.so**
Linux（x64）64 ビット	**libJavaBinder.x64.so**
Solaris（x86）32 ビット	**libJavaBinder.solaris.x86.so**
Solaris（x64）64 ビット	**libJavaBinder.solaris.x64.so**
Solaris（Sparc）32 ビット	**libJavaBinder.solaris.sparc32.so**
Solaris（Sparc）64 ビット	**libJavaBinder.solaris.sparc64.so**
HP-UX（RISC）32 ビット	**libJavaBinder.hp-ux.sl**

次の手順に従ってください：

1. **LISA_HOME¥agent** ディレクトリから以下のファイルを取得します。
   - **LisaAgent.jar**
   - ターゲット環境用のプラットフォーム依存のライブラリ モジュール。
2. Java アプリケーションからの読み取り権限があり、自動的にアプリケーション クラスパス（**¥WEB-INF¥lib** ディレクトリなど）でロードされないディスク上の任意の場所にファイルを配置します。
3. プラットフォーム固有のエージェント インストール（P. 25）セクションに移動し、使用しているプラットフォーム用の手順に従います。 以下の形式でエージェント パラメータ文字列を指定します。

   ```
   -agentpath:<path_to_JavaBinder_file>=jar=file:<path_to_LisaAgent.jar>[,url=<broker_url>][,name=<agent_name>]
   ```

4. Java アプリケーションを起動します。

エージェント パラメータ文字列のオプション

DevTest Java エージェントをインストールするときに、エージェント パラメータ文字列を指定します。 このトピックでは、文字列に含めることができるオプションについて説明します。

重要: **name** オプションは必須です。

url

エージェントが接続するブローカを定義します。以下の形式を使用します。

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

broker-host の部分は、ブローカが展開されているコンピュータの名前または IP アドレスに設定します。

broker-port の部分は、ブローカがリスンしている TCP ポートに設定します。 デフォルト値は 2009 です。

name

エージェントの名前を定義します。 以下の形式を使用します。

```
name=agent-name
```

わかりやすい名前を使用します。

複数のエージェントに同じ名前を使用しないようにしてください。

token

エージェントへのアクセスを保護します。 以下の形式を使用します。

```
token=admin-token:user-token
```

詳細については、「Java エージェントのセキュリティ (P. 86)」を参照してください。

heap

heap オプションはネイティブ エージェントでのみ使用可能です。

ヒープ ウォーキング API （TIBCO BusinessWorks ベースラインで必要）を有効にするかどうかを指定します。 有効な値は true および false です。 デフォルト値は false です。

jar

jar オプションはネイティブ エージェントでのみ使用可能です。

LisaAgent.jar ファイルのパスを定義します。

エージェント インストール アシスタント

エージェント インストール アシスタントは、エージェント パラメータ文字列の値を決定するのに役立つ、コマンド ライン ユーティリティです。

エージェント インストール アシスタントは、エージェントがインストールされた JVM が正しく動作していることの確認も行います。

注: TIBCO BusinessWorks ベースラインを作成する場合は、エージェント パラメータ文字列に **heap** オプションを追加して、値を true に設定します。

自動設定

エージェント インストール アシスタントは、**JavaAgentInstaller.xml** ファイル内のユーザ値に基づいて、プラットフォーム上の Java エージェントを自動的に設定するオプションを提供します。 この XML ファイルは、DevTest でサポートされているプラットフォームおよびバージョン情報が含まれているテンプレートです。**JavaAgentInstaller.xml** ファイルは、**LisaAgent.jar** ファイルと同じディレクトリにあります。

Java エージェントを自動的に設定するオプションを使用するには、以下の前提条件手順を実行する必要があります。

次の手順に従ってください:

1. **JavaAgentInstaller.xml** ファイルにリストされているプラットフォーム名およびオペレーティング システム タイプに基づいてプラットフォームを検索します。
2. 該当する場合、環境変数を定義します。

   環境変数は ${} で囲まれています（例： ${JBOSS_HOME}）。
3. 該当する場合、任意のカスタマイズされた値を置換します。

   カスタマイズされた値は {} で囲まれています。以下に例を示します。

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

   この値をユーザのプロジェクト名に置換します。

   ```
   <ConfigFileName>${ORACLE_HOME}¥user_projects¥domains¥lisa_domain¥bin¥startWebLogic.cmd</ConfigFileName>
   ```
4. 事前定義済みの値をすべて確認し、ユーザのシステムに基づいて値を編集します。

エージェント インストール アシスタントの実行

以下の手順では、**java** 実行可能ファイル、ブローカ URL、およびエージェント名のパスを個別に入力します。 ただし、2 番目の手順でこれらの値を指定することもできます。

次の手順に従ってください:

1. エージェントをインストールするコンピュータに移動します。
2. **-ia** オプションを指定して **LisaAgent.jar** ファイルを実行します。
   - Pure Java エージェントをインストールする場合は、**-java** オプションを追加します。
   - ネイティブ エージェントをインストールする場合は、**-native** オプションを追加します。
3. プロンプトが表示されたら、**java** 実行可能ファイルのパスを入力します。 デフォルト値は実行している **java** 実行可能ファイルのパスです。
4. プロンプトが表示されたら、ブローカ URL を入力します。 デフォルト値は **tcp://localhost:2009** です。
5. プロンプトが表示されたら、エージェント名を入力します。 デフォルト値は **{{x}}** です。これは、一意の名前が実行時に生成されることを意味します。
6. 出力を確認します。
7. （オプション）エージェントを自動的に設定するには、**Yes** を入力しプロンプトに従います。

詳細については、以下の例を参照してください。

- 例 1： Pure Java エージェント（P. 23）
- 例 2： 自動設定を使用する Pure Java エージェント（P. 24）
- 例 3： ネイティブ エージェント（P. 25）

例 1 - Pure Java エージェント

この例は、Pure Java エージェントに基づいています。 自動設定オプションは使用されません。

```
java -jar LisaAgent.jar -ia -java
Enter the path of the Java executable [C:¥Program Files¥Java¥jre7¥bin¥java]:


DevTest ブローカの URL [tcp://localhost:2009] を入力します。

NOTE: If the DevTest agent name contains the character sequence

{{x}}

it will be replaced with a unique identifier.

Enter the DevTest agent name [{{x}}]:



=====

System verification complete.  You can manually add these options to the java
command-line to start the DevTest agent:


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

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
n


Please manually add these options to the Java command-line to start the DevTest agent
```

例 2: 自動設定を使用する Pure Java エージェント

この例は、自動設定オプションを使用する Windows 上の JBoss 4.2 Pure Java エージェントに基づいています。

```
java -jar LisaAgent.jar -ia -java -broker tcp://localhost:2009 -name TestAgent66
Enter the path of the Java executable [C:¥Program
Files¥Java¥jdk1.7.0_40¥jre¥bin¥java]:
=====
System verification complete. You can manually add these options to the java
command-line to start the DevTest agent:

-javaagent:C:¥Project¥DevTest¥main¥remote¥dist¥agent¥InsightAgent.jar=url=tcp://l
ocalhost:2009,name=TestAgent66
Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
y
Please enter the index in [] to choose the platform:
[1] JBoss
[2] IBM WebSphere App Server
[3] Oracle WebLogic Server
[4] TIBCO BusinessWorks
[5] WebMethods Integration Server Windows Service
[X] Exit
1
Detected version 4.2 installed on your system.
Enter path for Java agent configuration file [C:¥DevTest¥jboss¥bin¥run.bat]:
Automatic configuration of the Java Agent is complete. Please start your service.
The configuration info has been updated. The previous configuration was:
SET
JAVA_TOOL_OPTIONS=-javaagent:C:¥Project¥DevTest¥main¥remote¥dist¥agent¥InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent7 %JAVA_TOOL_OPTIONS%
The previous configuration has been saved to file
'C:¥DevTest¥jboss¥bin¥run.bat.2014-54-25_09-54-22'
The current agent configuration is:
SET
JAVA_TOOL_OPTIONS=-javaagent:C:¥Project¥DevTest¥main¥remote¥dist¥agent¥InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent66 %JAVA_TOOL_OPTIONS%
```

例 3 - ネイティブ エージェント

この例は、Solaris（Sparc）32 ビットのネイティブ エージェントに基づいています。

```
$ java -jar dist/agent/LisaAgent.jar -ia -native tcp://localhost:2009 MyAgent
Enter the path of the 'java' executable [...]: /usr/jdk/instances/jdk1.6.0/bin/java

=====

System verification complete.  Add these options to the java command-line to start
the DevTest agent:


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

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter

n

Please manually add these options to the Java command-line to start the DevTest agent
```

プラットフォーム固有のエージェント インストール

このセクションでは、DevTest Java エージェントのインストールに関する、プラットフォーム固有の情報について説明します。

このセクションでは、以下のトピックについて説明します。

IBM WebSphere Application Server

このトピックは、IBM WebSphere Application Server 7.0 および 8.5 に適用されます。

IBM WebSphere Application Server を DevTest Java エージェントを使用するように設定するには、以下のいずれかの方法を使用します。

- 管理コンソール
- server.xml ファイル
- wsadmin ツール

それぞれの方法では、汎用 JVM 引数としてエージェント パラメータ文字列を指定します。 必要な値を決定するために、エージェント インストール アシスタント (P. 21)を使用できます。 以下の例は、Pure Java エージェントに基づいています。

```
-javaagent:/home/itko/agent/InsightAgent.jar=url=tcp://172.24.255.255:2009,name=w
as70_linux32
```

管理コンソール

この手順では、エージェント パラメータ文字列を指定するために Web ベースの管理コンソールを使用します。

次の手順に従ってください:

1. 管理コンソールに移動します。
2. アプリケーション サーバに移動します。
3. ［Java and Process Management］を展開し、［Configuration］タブの［Process definition］リンクをクリックします。
4. ［Additional Properties］の下の［Java Virtual Machine］をクリックします。
5. ［Generic JVM arguments］フィールドでエージェント パラメータ文字列を指定します。

server.xml ファイル

この手順では、**server.xml** ファイルを使用してエージェント パラメータ文字列を指定します。

次の手順に従ってください:

1. **WAS_HOME/AppServer/profiles/AppSrv01/config/cells/<cell_name>/nodes/<node_name>/servers/server1** ディレクトリに移動します。
2. **server.xml** ファイルを開きます。
3. ファイルの末尾にある **genericJvmArguments** エントリを変更します。

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

wsadmin ツール

wsadmin ツールを使用できます。

以下の例では、エージェント パラメータ文字列は **AdminConfig** オブジェクトの modify コマンドで指定されます。Jacl スクリプト言語が使用されます。

```
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

このトピックは、JBoss 4.2 および 7 に適用されます。

DevTest Java エージェントを使用するように JBoss を設定するには、JBoss の起動スクリプトを編集します。

エージェント インストール アシスタント (P. 21)を使用してエージェント パラメータ文字列の値を決定できます。

次の手順に従ってください:

1. JBoss の **run.bat** または **run.sh** ファイルをテキスト エディタで開きます。
2. Java 実行可能ファイルをコールする前に以下のプロパティを定義します。

   ```
   set JAVA_OPTS=<agent_parameters_string>
   ```

3. ファイルを保存します。

Jetty

このトピックは Jetty 8 および 9.*x* に適用されます。

DevTest Java エージェントを使用するように Jetty を設定するには、**JAVA_TOOL_OPTIONS** 環境変数でエージェント パラメータ文字列を指定します。

エージェント インストール アシスタント (P. 21)を使用してエージェント パラメータ文字列の値を決定できます。

```
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
```

Oracle WebLogic Server

このトピックは Oracle WebLogic Server 10.3 および 12.1.1 に適用されます。

Oracle WebLogic Server を DevTest Java エージェントを使用するように設定するには、以下のいずれかの方法を使用します。

- 管理コンソール
- config.xml ファイル
- 起動スクリプト

それぞれの方法では、サーバ起動引数としてエージェント パラメータ文字列を指定します。必要な値を決定するために、エージェント インストール アシスタント (P. 21)を使用できます。 以下の例は、Pure Java エージェントに基づいています。

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

管理コンソール

JVM に引数を追加するための公式にサポートされている方法に従うには、WebLogic 管理コンソールからエージェント パラメータ文字列を指定します。

次の手順に従ってください:

1. WebLogic 管理コンソールを開きます。
2. ［Domain Structure］パネルで［Environments］ノードを展開し、［Servers］ノードをクリックします。
3. ［Configuration］タブおよび［Server Start］サブタブをクリックします。
4. ［Arguments］フィールドで、エージェント パラメータ文字列を追加します。
5. 変更を保存します。

config.xml ファイル

もう 1 つの方法は、ドメインの中央設定ファイルを編集することです。

次の手順に従ってください:

1. **WEBLOGIC_HOME¥user_projects¥domains¥base_domain¥config** ディレクトリに移動します。
2. **config.xml** ファイルを開きます。
3. エージェント パラメータ文字列をターゲット サーバの **<server>/<server-start>/<arguments>** ノードに追加します。

起動スクリプト

WebLogic 起動スクリプトを編集することもできます。

以下の例では、エージェント パラメータ文字列を設定するために **JAVA_TOOL_OPTIONS** 環境変数が使用されます。 これらの行は、**startWebLogic.sh** ファイルの Java バージョンの確認と実際の WebLogic 呼び出しの間にあります。

```
JAVA_TOOL_OPTIONS=<agent_parameters_string>
export JAVA_TOOL_OPTIONS
```

起動スクリプトがカスタマイズされている場合、**JAVA_TOOL_OPTIONS** 環境変数は使用されません。

TIBCO BusinessWorks

このトピックは、TIBCO BusinessWorks 5.*x* に適用されます。

すべてのアプリケーション用のエージェントを設定するには、以下のファイルを編集します。

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

このファイルを編集しても、すでに作成されているアプリケーションには影響しません。

既存のアプリケーション用のエージェントを設定するには、以下のファイルを編集します。

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

各項目の説明：

- <domain-name> は TIBCO ドメインの名前です。
- <application-name> は展開した TIBCO アプリケーションの名前です。
- <service-name> は展開したサービスの名前です。

以下に例を示します。

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

いずれかのファイルに、以下の行を追加します。

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

エージェント インストール アシスタント (P. 21)を使用して文字列を決定できます。TIBCO BusinessWorks ベースラインを作成する場合は、必ず **heap** オプションを追加して、値を **true** に設定します。

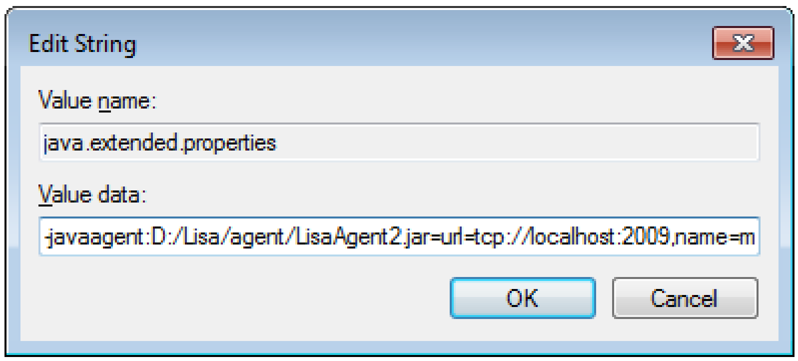
Windows サービス用の設定の違い

TIBCO プロセスが Windows サービスとして実行されている場合、.tra ファイルには **java.extended.properties** 行を追加しません。 代わりに、Windows レジストリに必要な情報を指定します。

Windows レジストリ エディタで、アプリケーション用の TIBCO BusinessWorks サービスのレジストリ キーを見つけます。 通常の名前は、**domain-name.application-name** です。

そのレジストリ キーの下にある Parameters キーを見つけます。Parameters キーの下で、**java.extended.properties** と言う名前の文字列値を作成または編集します。 値データをエージェント パラメータ文字列に設定します。

以下の図は、値データを設定する例を示しています。



webMethods Integration Server

このトピックは webMethods Integration Server 9.0 および 9.5 に適用されます。

server.bat または **server.sh** ファイルでエージェント パラメータ文字列を指定します。 必要な値を決定するために、エージェント インストール アシスタント (P. 21)を使用できます。以下の例は、Pure Java エージェントに基づいています。

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

次の手順に従ってください:

1. webMethods Integration Server インストールで **bin** ディレクトリを見つけます。
2. **server.bat** または **server.sh** ファイルを開きます。
3. **java** 実行可能ファイルをコールする行の前にエージェント パラメータ文字列を追加します。
4. ファイルを保存します。

Windows サービス

webMethods Integration Server が Windows サービスとして定義されている場合、**if "1%1"=="1-service"** スイッチに定義されている **/jvmargs** 値にエージェント パラメータ文字列を挿入することで、**server.bat** ファイルを編集できます。 以下に例を示します。

```
"%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%#%AP
PENDCLASSES_SWITCH%#%APPENDCLASSES%#%ENV_CLASSPATH_SWITCH%#%ENV_CLASSPATH%#%3#%4#
%5#%6#%7#%8#%9
```

Wily Introscope エージェント

DevTest Java エージェントは、Wily Introscope エージェントと共存できます。

Wily Introscope エージェントは Pure Java エージェントで、通常、**-javaagent** 構文で設定されます。

両方のエージェントで **-javaagent** 構文を使用している場合は、Wily Introscope エージェントの*前*に DevTest Java エージェントを指定します。

通常のアプリケーション サーバまたはコンテナに対しては、以下の例が適切です。

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

また、以下の例も適切です。

```
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
```

一方、この例は正しくありません。

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

その行と一緒に、どこかで必ず以下を実行してください。

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

Java エージェントの使用方法

このセクションには、以下のトピックが含まれます。

ブローカの起動

ブローカが Windows サービスとしてインストールされていない場合は、ブローカを手動で起動します。

エンタープライズ ダッシュボードおよび DevTest ポータルでエージェントを表示するには、ブローカが実行中である必要があります。

次の手順に従ってください:

1. レジストリが実行されていることを確認します。
2. 以下のいずれかの操作を実行します。
   - コマンド プロンプトを開き、**LISA_HOME¥bin** ディレクトリに移動し、**Broker** 実行可能ファイルを実行します。
   - インストールにスタート メニュー フォルダがある場合は、［スタート］メニュー - ［すべてのプログラム］ - ［DevTest Solutions］ - ［ブローカ］をクリックします。

rules.xml ファイル

DevTest Java エージェントの設定ファイルの名前は、**rules.xml** です。

rules.xml ファイルは、ブローカが実行されているコンピュータの **LISA_HOME** ディレクトリにあります。 ブローカが初めて起動されるときに、**rules.xml** ファイルが生成されます。ブローカに接続するすべてのエージェントがこの設定を使用します。

デフォルトでは、**rules.xml** ファイルには、設定プロパティのサンプル セットのみが含まれています。 プロパティおよびそれらのデフォルト値は、すべて内部に保持されます。

rules.xml.sample という名前のファイルでそれらのプロパティを確認できます。 ブローカが初めて起動されるときに、**rules.xml.sample** ファイルが生成されます。**rules.xml.sample** ファイルは、**rules.xml** ファイルと同じディレクトリにあります。

プロパティにはそれぞれコメント、名前および値が含まれます。 以下に例を示します。

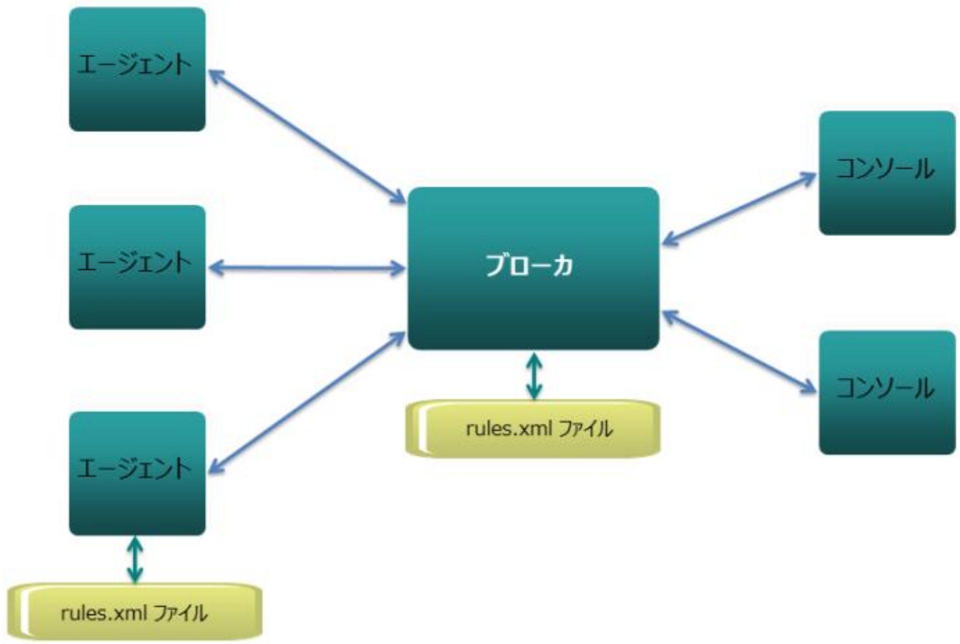
```
<property comment="Keep track of file descriptors"
key="lisa.agent.tracking.disabled" value="false"/>
```

以下の XML は、**rules.xml** ファイルの一般的な形式を示しています。 **agent** エレメントには、エージェントのプロパティが含まれます。 **broker** エレメントには、ブローカのプロパティが含まれます。 **console** エレメントには、コンソールのプロパティが含まれます。

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

rules.xml ファイルは、エージェント側またはコンソール側に配置することもできます。 これらのファイル内の設定は、ブローカ側のエージェントまたはコンソールの情報より優先されます。

以下の図は、設定の例を示しています。 ブローカは、同じコンピュータにある **rules.xml** ファイルにアクセスします。 複数のエージェントおよびコンソールがブローカに接続されています。エージェントのうちの 1 つには、独自の **rules.xml** ファイルがあります。



注: エージェントはブローカへの接続を行う前に以下のプロパティを読み取るため、以下のプロパティはエージェント側でのみ設定できます。

- 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

特定の機能を実行するために、**rules.xml** ファイルに*ディレクティブ*を追加できます。 ディレクティブの詳細については、以下のトピックを参照してください。

- **category** ディレクティブ： カテゴリ設定 (P. 47)

- **database** ディレクティブ： データベースのシンクの設定（P. 46）
- **exclude** ディレクティブ： インターセプトおよび仮想化からの除外（P. 51）
- **feature** ディレクティブ： プロトコルの重み付けの設定（P. 41）
- **intercept** ディレクティブ：インターセプトするメソッドの追加（P. 49）
- **loadbalancer** ディレクティブ： ロード バランサおよびネイティブ Web サーバ（P. 85）
- **response** ディレクティブ： Java エージェントの自動応答生成（P. 53）
- **track** ディレクティブ： トラッキング用のクラスの追加（P. 48）
- **virtualize** ディレクティブ：VSE のインスツルメンテーション ルール（P. 44）

グループ内のエージェントの管理

2 つ以上のエージェントを含む*グループ*を作成できます。

グループを作成した後はグループ レベルで設定を変更でき、同じ変更をエージェントごとに個別に加える必要がなくなります。

注: 1 つのエージェントが複数のグループに属することはできません。

グループに属さないエージェントは*スタンドアロン エージェント*と呼ばれます。

以下の XML では、グループを持つ **rules.xml** ファイルの一般的な形式を表しています。**group** エレメントには、グループのプロパティが含まれます。**agent** エレメントには、エージェントのプロパティが含まれます。 エージェントがグループの一部である場合は、**agent** エレメントが **group** エレメント内に現れます。 **broker** エレメントには、ブローカのプロパティが含まれます。 **console** エレメントには、コンソールのプロパティが含まれます。

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

group エレメント内の **agent** エレメントに 1 つ以上のプロパティがある場合、そのプロパティはグループの設定をオーバーライドします。

group エレメントの内部に **broker** エレメントまたは **console** エレメントを配置しないでください。

DevTest ポータルからエージェント プロパティを更新すると、設定は **rules.xml** ファイルのスタンドアロン エージェントのセクションに保存されます。

注: DevTest ポータルの詳細情報については、「*CA Continuous Application Insight の使用*」を参照してください。

例: rules.xml ファイルでのグループの設定

以下の XML は、1 つのグループが含まれる **rules.xml** ファイルを示しています。このグループには､3 つのエージェントが含まれています。このファイルには、スタンドアロン エージェントも含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

プロトコルの重み付けの設定

feature ディレクティブでは、DevTest Java エージェントがキャプチャできる各プロトコルのキャプチャ レベルを変更できます。

注: WebSphere MQ、JMS など、キューベースのクライアント/サーバ通信では、異なるキャプチャ レベルの設定はサポートされていません。

DevTest ポータルからキャプチャ レベルを変更することもできます。詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

feature ディレクティブの形式は、以下のとおりです。

```
<feature name="プロトコル名" weight="重み"/>
```

feature ディレクティブは、**rules.xml** ファイルの **group** エレメントまたは **agent** エレメント内に配置することができます。

weight 属性を 0、4、または 8 に設定します。 値 0 は「**カウント**」レベルに対応します。値 4 は「**カウント数およびパス**」レベルに対応します。値 8 は「**全データ**」レベルに対応します。

以下の例は、JDBC プロトコルを「**全データ**」レベルに設定します。

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

シグネチャの指定

rules.xml ファイル内の以下のディレクティブには、シグネチャの指定が含まれます。

- exclude
- intercept
- respond
- virtualize

シグネチャの指定は、Java メソッドのシグネチャの特性を説明するために使用します。

指定の最初の部分は **signature** という語で、その後に等号と引用符が続きます。

指定の 2 番目の部分には、丸かっこで囲まれた引数が含まれます。メソッドに引数がない場合でも、丸かっこを含める必要があります。

指定の 3 番目の部分には、戻り値型が含まれ、その後に引用符が続きます。戻り値型が void の場合は、V という文字を使用します。

指定には、スペースを使用しないでください。

引数または戻り値型でプリミティブ型を指定するには、以下のいずれかの文字を使用します。

文字	プリミティブ型
Z	boolean
B	byte
C	char
D	double
F	float
I	int
J	long
S	short

完全修飾クラスを指定するには、以下の手順に従います。

- 先頭に文字 L を追加します。
- 区切り文字として、ドットではなくスラッシュを使用します。
- 末尾にセミコロンを追加します。

以下に例を示します。

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

例: 1 つの引数、void を返す

javax.jms.MessageListener インターフェースの **onMessage()** メソッドをインターセプトすると仮定します。 このメソッドは、以下のシグネチャを持ちます。

- 引数は **javax.jms.Message** オブジェクトです。
- 戻り値型は void です。

インターセプト ルールのシグネチャの指定は、以下のようになります。

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

例: 引数なし、プリミティブ型を返す

javax.jms.MessageProducer インターフェースの **getPriority()** メソッドをインターセプトすると仮定します。 このメソッドは、以下のシグネチャを持ちます。

- このメソッドには引数がありません。
- 戻り値型は整数です。

インターセプト ルールのシグネチャの指定は、以下のようになります。

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

VSE のインスツルメンテーション ルール

エージェントの **rules.xml** ファイルに **virtualize** ディレクティブを追加することで、VSE 機能をカスタマイズできます。

virtualize ディレクティブは、**rules.xml** ファイルの **group** エレメントまたは **agent** エレメント内に配置することができます。

仮想化用のクラスの追加（レコーディング モードおよび再生モード）

```
<virtualize class="クラス名"/>
```

例：

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

特定のプロトコルのセッションの構成要素を確認する方法のエージェントへの指示

セッション識別子を返すコード スニペットを指定することにより、このタスクを実行します。

```
<virtualize>
   <track class="クラス名" method="メソッド名" signature="シグネチャ"
push="true|false">
      <code><![CDATA[" および "]]></code>
   </track>
</virtualize>
```

シグネチャの形式については、「シグネチャの指定 (P. 42)」を参照してください。

例：

```
<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>
```

これらの例はエージェント内にハードコードされているため、**rules.xml** ファイル内には必要ありません。 ただし、これらの行では、エージェントを再コンパイルせずに HTTP および EJB 以外の任意の数のプロトコルに対してプロセスを実装できるように、プロセスの動作を示しています。

class 属性の値は､そこからセッションへのアクセスを取得するクラスです｡

method および **signature** 属性の値は、呼び出されたときにセッション識別子を計算するメソッドを決定します。 この計算では、**code** 属性の値を使用します。**$0** はソース オブジェクトを表します。**$1**、**$2** などはメソッドの引数です。

push 属性は､VSE フレームによって後で使用するためにそのセッションを格納する方法を指定します。

- **push="true"** は、コンテナがセッションを設定し、セッションへのオブジェクトのマッピングを維持することを指定します。
- **push="false"** は、セッションはスレッドにスコープされ、スレッドのローカル変数に格納することを指定します。

最も内側のセッション（識別子）は、VSE によって **com.itko.lisa.remote.vse.VSEFrame** の **getSessionId()** メソッドで返されます。詳細については､**LISA_HOME¥doc** ディレクトリで､JavaDocs を参照してください。

データベースのシンクの設定

database ディレクティブは、**rules.xml** ファイルの **broker** エレメントに追加できます。

この設定は、エージェントのライフスパンにおいて常に使用されます。

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

例：

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

カテゴリ設定

DevTest Java エージェントは、各トランザクション フレームにカテゴリを割り当てます。 エージェントがカテゴリを決定できない場合、デフォルトのカテゴリが割り当てられます。

デフォルト以外のカテゴリの 1 つとインターセプトされたクラスまたはインターフェースを関連付けるようにエージェントを設定できます。 エージェントの **rules.xml** ファイルに **category** ディレクティブを追加します。 クラスまたはインターフェース名、およびカテゴリ番号を指定します。

category ディレクティブの形式は、以下のとおりです。

```
<category class="クラスまたはインターフェース名" value="カテゴリ番号"/>
```

category ディレクティブは、**broker** エレメント内に配置する必要があります。

有効なカテゴリ番号は以下のとおりです。

- 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

例：

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

注: 可能な限り、エージェントによってデフォルト以外のカテゴリがすでに割り当てられているトランザクション フレームのカテゴリを変更しないことをお勧めします。

トラッキング用のクラスの追加

track ディレクティブは、後でインスタンスをヒープから容易に取得できるようにします。

track ディレクティブは、**rules.xml** ファイルの **group** エレメントまたは **agent** エレメント内に配置することができます。

```
<track class="クラス名"/>
```

例：

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

インターセプトするメソッドの追加

エージェントの **rules.xml** ファイルに **intercept** ディレクティブを追加することで、DevTest Java エージェントでインターセプトするメソッドを追加できます。

intercept ディレクティブの形式は、以下のとおりです。

```
<intercept class="クラス名" method="メソッド名" signature="シグネチャ"/>
```

intercept ディレクティブは、**rules.xml** ファイルの **group** エレメントまたは **agent** エレメント内に配置することができます。

シグネチャの形式については、「シグネチャの指定 (P. 42)」を参照してください。

クラス階層が考慮されます。 クラス B がクラス A を拡張し、両方のクラスがメソッド M を定義すると仮定します。クラス A のメソッド M をインターセプトする場合、クラス B のメソッド M もキャプチャされます。

デフォルトでは、CA Continuous Application Insight はゲッター メソッドおよびセッター メソッドをキャプチャしません。 ゲッター メソッドおよびセッター メソッドを追加するには、値を true に設定した **delay** 属性を含めます。

例

以下の例は、**com.itko.examples.entity.Account** クラスの **setName()** メソッドを追加します。 このメソッドは、引数として **java.lang.String** オブジェクトを受け取ります。 このメソッドは void を返します。

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

以下の例は、**com.itko.examples.entity.Account** クラスの **getTransactions()** メソッドを追加します。このメソッドは引数を受け取りません。このメソッドは、**java.util.Collection** オブジェクトを返します。

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

以下の例は、**com.itko.examples.airline.ws.jaxws.Request** クラスの **getRequestTypes()** メソッドを追加します。このメソッドは引数を受け取りません。このメソッドは、**java.lang.String** オブジェクトの配列を返します。

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

インターセプトおよび仮想化からの除外

エージェントの **rules.xml** ファイルに **exclude** ディレクティブを追加することで、DevTest Java エージェントがメソッド、クラス、またはパッケージをインターセプトまたは仮想化できないようにすることができます。

exclude ディレクティブの形式は、以下のとおりです。

```
<exclude class="クラス名" method="メソッド名" signature="シグネチャ"/>
```

exclude ディレクティブは、**rules.xml** ファイルの **group** エレメントまたは **agent** エレメント内に配置することができます。

シグネチャの形式については、「シグネチャの指定 (P. 42)」を参照してください。

クラス階層は考慮されません。 クラス B がクラス A を拡張し、両方のクラスがメソッド M を定義すると仮定します。 クラス A を除外する場合、クラス A のメソッド M はキャプチャされません。 ただし、クラス B のメソッド M はキャプチャされます。

メソッドまたはシグネチャに関係なくクラスまたはパッケージを除外する場合、以下のいずれかの方法で実行できます。

- 「**method="*" signature="*"**」の指定
- **method** および **signature** 属性の省略

exclude ディレクティブを追加または削除する場合は、エージェントを再起動する必要があります。

デフォルトでは、CA Continuous Application Insight はゲッター メソッドおよびセッター メソッドをキャプチャしません。 ゲッター メソッドおよびセッター メソッドを除外する必要はありません。

例

以下の例は、**com.itko.examples.ejb3.OrdinaryBean** クラスの **serviceComposition()** メソッドを除外します。 このメソッドは引数を受け取りません。 このメソッドは、**java.lang.String** オブジェクトを返します。

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

以下の例は、**com.itko.examples.ejb3.OrdinaryBean** クラスを除外します。

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

以下の例は、**com.itko.examples.ejb** パッケージを除外します。 ワイルドカード文字が 1 つ使用されていることに注目してください。

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

以下の例は、**com.itko.examples** パッケージおよびそのすべてのサブパッケージを除外します。ワイルドカード文字が 2 つ使用されていることに注目してください。

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

Java エージェントの自動応答生成

DevTest Java エージェントを使用して、バックエンドが使用可能ではない場合でも、トランザクションを記録することができます。

この機能を設定するには、**respond** ディレクティブを **rules.xml** ファイルに追加します。 **respond** ディレクティブの形式は、以下のとおりです。

```
<respond class="クラスまたはインターフェース名" method="メソッド名"
signature="シグネチャ" source="文字列値" args="文字列値" return="クラス名
"/>
```

respond ディレクティブは、**rules.xml** ファイルの **agent** エレメント内に配置する必要があります。

必須の属性は、**class** 属性のみです。

エージェントがインターセプトするメソッド（複数可）を指定するには、以下の属性を使用します。

- **class**： メソッド（複数可）を定義するクラスまたはインターフェースの名前を定義します。 インターフェースを指定する場合、そのインターフェースを実装するすべてのクラスも含まれます。
- **method**： メソッドの名前を定義します。 この属性が指定されていない場合、クラスまたはインターフェースのすべてのメソッドが含まれます。
- **signature**: メソッドのシグネチャを定義します。この形式については、「シグネチャの指定（P. 42）」で説明されています。
- **source**: オブジェクトに対する **toString()** メソッドのコールにこの文字列値が含まれる場合、そのメソッドをインターセプトします。
- **args**: 引数に対する **toString()** メソッドのコールにこの文字列値が含まれる場合、そのメソッドをインターセプトします。

エージェントは、指定に一致するメソッド コールをインターセプトする際に、以下のアクションを実行します。

- どのタイプのオブジェクトを返す必要があるかについて、経験に基づいて推測します。
- 汎用のオブジェクトを返します。 オブジェクトにはランダムな値が含まれます。

return 属性を使用してオブジェクト タイプを指定すると、最初のアクションをオーバーライドします。

値に小なり記号（<）が含まれる場合、小なり記号を以下のテキストに置換します。

```
&lt;
```

値に大なり記号（>）が含まれる場合、大なり記号を以下のテキストに置換します。

```
&gt;
```

例: EJB コール用の自動応答生成

以下の Java メソッドは、JNDI コンテキスト オブジェクトを作成し、EJB を実行できるサーバをコールし、API を呼び出します。

```
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");

        /* 最初のリモート コール (オプション) */
        Context ctx = new InitialContext(props);

        /* 2 番目のリモート コール */
        EJB3UserControlBeanRemote remote = (EJB3UserControlBeanRemote)
ctx.lookup("EJB3UserControlBean/remote");

        /* 3 番目のリモート コール */
        User u = remote.getUser("lisa_simpson");

        /* ローカル コール ... */
        System.out.println(u.getEmail());
    }
```

サーバが使用可能ではないと仮定します。 このメソッドを実行すると、接続を取得できないため、このコードは成功しません。

以下の **response** ディレクティブを使用して、このコードを強制的に成功させることができます。最初のディレクティブは、**InitialContext** オブジェクトの作成をインターセプトします。2 番目のディレクティブは、**Context** オブジェクトの **lookup()** メソッドをインターセプトします。3 番目のディレクティブは、**EJB3UserControlBeanRemote** オブジェクトのすべてのメソッドをインターセプトします。

```
<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"/>
```

Java エージェントに対する開発

クライアント側からの DevTest Java エージェントへの主なインターフェースは、**com.itko.lisa.remote.client.AgentClient** です。 このクラスには、エージェントまたはブローカ上の API の呼び出し、エージェントの検出、およびそれらのステータス変更（オンライン/オフライン）の通知の受け取りを行うメソッドがあります。

また、このクラスにより、機能の主な領域でエージェントのインタラクションを担当するクラスにアクセスできます。

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

注: エージェントの JavaDocs で、これらのクラスに関する詳細情報を表示できます。 JavaDocs は、**LISA_HOME¥doc** ディレクトリにあります。

このセクションには、以下のトピックが含まれます。

エージェントの一般的な API

注: このトピックで説明するインターフェースおよびクラスに関する詳細情報は、エージェントの JavaDocs で表示できます。 JavaDocs は、**LISA_HOME¥doc** ディレクトリにあります。

ほとんどのクライアント API では、ターゲットとしてエージェントを指定する必要があります。 この指定は、**com.itko.lisa.remote.IAgentInfo** タイプのパラメータを渡すことで実行されます。 このパラメータは、エージェントおよびその基本情報の一部を一意に識別するオブジェクトを表します。

```
/** このエージェントまたはコンソールの一意の識別子。 指定した場合、GUID は VM のライフスパンの間、永続します */
 public long getGuid();
 public void setGuid(long guid);

 /** 手動で指定するか、システム プロパティから生成される、このエージェントを識別するための読みやすい名前 */
 public String getName();
 public void setName(String name);

 /** このオブジェクトが生成されたマシンの名前（使用可能な場合） */
 public String getMachine();
 public void setMachine(String machine);

 /** このオブジェクトが生成されたマシンの IP （使用可能な場合） */
 public String getIp();
 public void setIp(String ip);

 /** このオブジェクトが実行されている JVM の作業ディレクトリ */
 public String getWorkingDir();
 public void setWorkingDir(String workingDir);

 /** java.class.path プロパティによって返されたクラスパス */
 public String getClasspath();
 public void setClasspath(String classpath);

 /** java.library.path プロパティによって返されたライブラリ パス */
 public String getLibpath();
 public void setLibpath(String libpath);

 /** エージェントの場合、呼び出された main メソッドを含む java クラスを返します */
 public String getMainClass();
 public void setMainClass(String mainClass);

 /** コマンドラインの省略版（表示用のため正確でない可能性あり） */
 public String getCommandLine();
```

```
/** このオブジェクトが送受信された時間を追跡するための一時的なフィールド */
public Date getGenerationTime();
public void setGenerationTime(Date time);

/** このエージェント上の API を JMS 経由で呼び出すために必要なパスワードを追跡するため
の一時的なフィールド */
public String getToken();
public void setToken(String token);
```

API の一部は、**com.itko.lisa.remote.client.AgentClient** から離れて直接確認することができます。 このリストは、すべてを網羅していませんが、ほとんどのクライアントの大半のニーズを満たしています。

```
/** 特定のエージェントのすべてのディスカバリと情報を担当するクラスを取得します */
public DiscoveryClient getDiscoveryClient();

/** トランザクション関連のすべての操作を担当するクラスを取得します */
public TransactionsClient getTransactionClient();

/** VSE 関連のすべての操作を担当するクラスを取得します */
public VSEClient getVSEClient();

/**
 * 検出されたすべてのエージェントを取得します。これは、その他すべての API コールで使用さ
れる IAgentInfos を取得するメイン API です
 * @param tokens： エージェントの GUID または名前とトークンのマップ、トークンでセキュ
リティが有効にされたエージェントがない場合は null
 * @return： 現在オンラインのエージェントを表す IAgentInfo オブジェクトのセット
 */
 public Set getRemoteAgentInfos(Map tokens);

/**
 * エージェントのオンライン情報を登録されているリスナに転送します
 * @param info： オンラインになったことが検出されたエージェント
 */
 public void onAgentOnline(IAgentInfo info);

/**
 * エージェントのオフライン情報を登録されているリスナに転送します
 * @param info： オフラインになったことが検出されたエージェント
 */
 public void onAgentOffline(IAgentInfo info);

/**
 * 指定したエージェントの任意のコードを評価します
 * @param info： このコードが評価されるエージェント
 * @param input： 実行するコード。 変数 '_agent' はエージェント インスタンスを表しま
す。
```

```
 * @return： コードによって返される値
 * @throws JMSInvocationExceptionthrown： コードがエージェントでスローした場合
 */
public Object eval(IAgentInfo info, String input) throws JMSInvocationException
```

エージェント ディスカバリ API

com.itko.lisa.remote.client.DiscoveryClient クラスは、エージェント上のデータの検出に関連するメソッドを提供します。 **AgentClient.getInstance().getDiscoveryClient()** コールで **DiscoveryClient** クラスを取得できます。

注: エージェントの JavaDocs で、このクラスに関する詳細情報を表示できます。 JavaDocs は、**LISA_HOME¥doc** ディレクトリにあります。

```
/**
  * 指定したエージェントのシステム プロパティ
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Map getVMProperties(IAgentInfo info) throws JMSInvocationException;

 /**
  * from と to の日付の間に、指定したエージェントに対して記録された StatsFrames のリストを返します。
  * @param agentInfo： 統計を取得するエージェント
  * @param from： フィルタを開始する日付
  * @param to： フィルタを終了する日付
  * @return： 新しいほうから（to の日付から開始）並べられた目的の StatsFrames のリスト
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to);

 /**
  * from と to の日付の間に、指定したエージェントに対して記録された StatsFrames のリストを返します。
  * @param agentInfo： 統計を取得するエージェント
  * @param from： フィルタを開始する日付
  * @param to： フィルタを終了する日付
  * @param interval： 結果を集約する方法（秒単位）。 10 の場合、10 秒ごとの結果が平均されます。
  * @param limit： 結果の最大数
  * @return： 新しいほうから（to の日付から開始）並べられた目的の StatsFrames のリスト
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to, int interval, int limit);

 /**
  * TODO： （再）インプリメント。現在はスローします
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Topology getTopology(IAgentInfo info) throws JMSInvocationException;
```

```
 /**
  * 出口点は、スタックの下でネットワーク コールを行うクラス/メソッドをキャプチャする
MethodInfo です
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Set getExitPoints(IAgentInfo info) throws JMSInvocationException;

 /**
  * J2EE コンテナ (J2EE コンテナでない場合は java) の名前を返します
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public String getServerInfo(IAgentInfo info) throws JMSInvocationException;

 /**
  * 指定した J2EE コンテナに展開されている Web アプリケーションを返します
  * @param info
  * @return
  */
 public WebApplication[] getWebApps(IAgentInfo info) throws
JMSInvocationException;

 /**
  * ClassNode ツリーで表された、指定したエージェントの JNDI 階層を返します
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getJNDIRoot(IAgentInfo info) throws JMSInvocationException;

 /**
  * エージェント VM の現在のスレッド
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ThreadInfo[] getThreadInfos(IAgentInfo info) throws
JMSInvocationException;

 /**
  * エージェント VM の現在のスレッド スタック
  * @param info
  * @return
  */
 public String[] dumpThreads(IAgentInfo info) throws JMSInvocationException;
```

```
 /**
  * 指定したエージェントのクラスパスにあるすべてのファイル
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Set getClasspath(IAgentInfo info) throws JMSInvocationException;

 /**
  * 指定したパスの下のクラス階層を返します
  * @param info
  * @param fromPath
  * @return
  */
 public ClassNode getClassNodes(IAgentInfo info, String fromPath) throws
JMSInvocationException;

 /**
  * 指定した URL のアーカイブで見つかったクラス階層
  * @param info
  * @param url
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getArchiveNodes(IAgentInfo info, URL url) throws
JMSInvocationException;

 /**
  * クラスに関するデータ (フィールド/メソッド/ソース) を含むセット
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public Set getClassInfo(IAgentInfo info, String className) throws
JMSInvocationException;

 /**
  * ソースを逆コンパイルし、クラスに返します
  * @param info
  * @param clazz
  * @param loc: クライアントまたはエージェントで逆コンパイルします
  * @return
  * @throws JMSInvocationException
  */
 public String getClassSrc(IAgentInfo info, String clazz, boolean loc) throws
JMSInvocationException, IOException;
```

```
 /**
  * このクラスが属する階層（すべての祖先とすべてのエクステンダ/インプリメンタ）を返します
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode[] getClassHierarchy(IAgentInfo info, String className) throws
JMSInvocationException;

/**
  * 指定したクラスのヒープに存在するすべてのオブジェクト（への参照）を返します。使用には注
意してください
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getInstancesView(IAgentInfo info, String className) throws
JMSInvocationException;

 /**
  * エージェントによって追跡されたヒープに存在するすべてのオブジェクト（への参照）を返しま
す
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getTrackedObjects(IAgentInfo info) throws
JMSInvocationException;

 /**
  * オブジェクト（再帰的に計算されたフィールド）の未加工のグラフを表示
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getObjectGraph(IAgentInfo info, String clazz, int hashCode)
throws JMSInvocationException;

 /**
  * オブジェクトから GC ルートへのパスを取得します
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
```

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

 /**
  * エージェント ファイル システムにあるファイルを取得し、それを一時的な場所でクライアン
トにダウンロードし、ハンドルをクライアントに返します
  * @param info
  * @param file
  * @return
  * @throws JMSInvocationException
  */
 public File getFile(IAgentInfo info, String file) throws JMSInvocationException,
IOException;
```

エージェント トランザクション API

注: エージェントの JavaDocs で、これらのクラスに関する詳細情報を表示できます。 JavaDocs は、**LISA_HOME¥doc** ディレクトリにあります。

トランザクションは、クライアント要求の結果として 1 つ以上のサーバによって実行されるコード パスです。 トランザクションは、要求を開始したクライアントをルートとするツリー構造によって表されます。 ツリーのノードは、**com.itko.lisa.remote.transactions.TransactionFrame** オブジェクトです。 これらのオブジェクトは、サーバ処理の一部として呼び出されたクラス、メソッド、および引数に関する情報をカプセル化します。 また、フレームには、期間、実行時間、およびそれが発生したスレッドなどの付随情報が含まれます。

トランザクションは、メソッド コール スタックと見なすことができます。 違いは、トランザクションがスレッド、プロセス、またはコンピュータの境界を越えるという点です。 また、スタックにはスレッドのコード実行に関係するメソッドがすべて含まれるのに対し、トランザクションはいくつかのレベルをスキップし、選択されたメソッドのフレームのみを含みます。 このようなメソッドは、*インターセプトされたメソッド*と呼ばれます。

TransactionFrame オブジェクトを取得および操作する主な方法は、**AgentClient.getInstance().getTransactionsClient()** コールで取得できる **com.itko.lisa.remote.client.TransactionsClient** によって提供される以下の API のいくつかのオーバーロードを使用する方法です。

```
/**
  * トランザクションの記録を開始します
  * @param info： 記録を開始するエージェント
  */
 public void startXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * トランザクションの記録を停止します
  * @param info： 記録を停止するエージェント
  */
 public void stopXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * クライアントとサーバのトランザクションを調整するために使用するクライアントのソケット使用率の追跡を開始します。
  * これは、addClientTransaction で追加するコールよりも前に呼び出す必要があります。
  * @param global： すべてのソケットまたはこのスレッドにのみインストールします
  */
 public void installSocketTracker(boolean global);
```

```
 /**
  * クライアントとサーバのトランザクションを調整するために使用するクライアントのソケット
使用率の追跡を停止します。
  * これは、addClientTransaction で追加するコールの後に呼び出す必要があります。
  * @param global： すべてのソケットまたはこのスレッドでのみアンインストールします
  */
 public void uninstallSocketTracker(boolean global);

 /**
  * クライアントは、このメソッドを呼び出して、指定したパラメータを使用して作成された新しい
クライアント トランザクションをルートとしたトランザクション ツリー
  * を作成できます。
  * 注: ここで追加するクライアント側トランザクションを開始する前に、installSocketTracker
をコールする必要があります
  * また、ネットワーク コールで完了した後に、uninstallSocketTracker をコールすることを
お勧めします。
  * @param stamp： ルート トランザクションを識別するために後で使用できる一意の文字列
  * (たとえば、getTransactions のコール)
  * @param stepInfo： トランザクションの内容をわかりやすく説明する文字列（null にする
こともできます）
  * @param args： クライアントがトランザクションに渡すパラメータ（空にすることもできます）
  * @param result： トランザクションの結果（null にすることもできます）
  * @param duration： クライアント側から見たトランザクションの継続時間
  */
 public void addClientTransaction(String stamp, String stepInfo, Object[] args,
Object result, long duration);

 /** このクライアントから発生したすべてのトランザクションと関連データを削除します。 */
 public void clearTransactions();

 /**
  * 引数として渡したフィルタに一致するすべてのエージェントから受信した
TransactionFrames のフラットなリストを取得します
  * @param offset： オフセット
  * @param limit： 結果の最大数
  * @param category： トランザクション カテゴリでフィルタします
 (TranactionFrame.CATEGORY_XXX を参照。フィルタなしの場合は 0)
  * @param clazz： クラス名でフィルタします（フィルタなしの場合は null または ""）
  * @param method： メソッド名でフィルタします（フィルタなしの場合は null または ""）
  * @param minTime： minTime よりも長いフレーム期間でフィルタします
  * @return： 指定した条件を満たす TransactionFrames を新しいほうから並べたリスト
  */
 public List getTransactions(int offset, int limit, int category, String clazz,
String method, int minTime);

 /**
  * 指定したトランザクションをルートとし、指定した条件を満たすトランザクション ツリーのリ
ストを取得します
```

```
  * @param offset： オフセット
  * @param limit： 結果の最大数
  * @param stamps： ルート クライアントのトランザクション スタンプの配列。これが空の場合、すべてが返されます
  * @param minTime： この期間を下回るトランザクションは、結果から除かれます
  * @return ret： 条件に一致するトランザクション ツリーを新しいほうから並べたリスト
  */
 public List getTransactionsTree(int offset, int limit, String[] stamps, int minTime)
```

トランザクションは複数のエージェントにまたがることができるため、これらの API のパラメータでは **Agent** または **AgentInfo** を指定しません。

これらの API は、**com.itko.lisa.remote.transactions.TransactionFrame** のリスト（ツリー）を返すため、カプセル化されているデータを確認できます。

```
/** このフレームの一意の識別子 */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** このフレームの親フレームのフレーム ID
 public String getParentId();
 public void setParentId(String parentId);

 /** 同じトランザクションに属するすべてのフレームによって共有される識別子（グローバル ルート フレーム ID と同じ） */
 public String getTransactionId();
 public void setTransactionId(String transactionId);

 /** 親 TransactionFrame オブジェクト */
 public TransactionFrame getParent();
 public void setParent(TransactionFrame parent);

 /** 子 TransactionFrame オブジェクトのリスト */
 public List getChildren();
 public void setChildren(List children);

 /** このフレームが記録されているエージェントの一意の識別子 */
 public long getAgentGuid();
 public void setAgentGuid(long agentGuid);

 /** フレームを並べるのに役立つ増加するカウンタ（時間の正確さは十分でない可能性があります）
*/
 public long getOrdinal();
 public void setOrdinal(long ordinal);

 /** ネットワークの受信または送信フレームは、これを設定して、通信元の IP を通知します */
 public String getLocalIP();
 public void setLocalIP(String ip);
```

```
/** ネットワークの受信または送信フレームは、これを設定して、通信元のポートを通知します */
public int getLocalPort();
public void setLocalPort(int port);

/** ネットワークの受信または送信フレームは、これを設定して、通信先の IP を通知します */
public String getRemoteIP();
public void setRemoteIP(String ip);

/** ネットワークの受信または送信フレームは、これを設定して、通信先のポートを通知します */
public int getRemotePort();
public void setRemotePort(int port);

/** このフレームが記録されているスレッドの名前 */
public String getThreadName();
public void setThreadName(String threadName);

/** このフレームが記録されているクラスまたはインターフェースの名前（インターセプトの指定
のとおり） */
public String getClassName();
public void setClassName(String className);

/** このフレームが記録されている実際のオブジェクトのクラスの名前 */
public String getActualClassName();

/** このフレームが記録されているメソッドの名前 */
public String getMethod();
public void setMethod(String method);

/** このフレームが記録されているメソッドのシグネチャ */
public String getSignature();
public void setSignature(String signature);

/** このフレームが記録されているオブジェクトを整形して表示 */
public String getSource();
public void setSource(Object source);

/** このフレームが記録されているメソッドの引数を整形して表示 */
public String[] getArguments();
public void setArguments(Object[] arguments);

/** このフレームが記録されているメソッドの結果を整形して表示 */
public String getResult();
public void setResult(Object result);

/** このフレームがその親の内部で重複した回数 */
public long getHits();
public void setHits(long hits);
```

```
/** フレームが記録されたサーバ時間 */
public long getTime();
public void setTime(long time);

/** このフレームの実行に要した実時間 */
public long getClockDuration();
public void setClockDuration(long clockDuration);

/** このフレームの実行に要した CPU 時間 */
public long getCpuDuration();
public void setCpuDuration(long cpuDuration);

/** このフレームに関連付けられている状態のカスタム表示 */
public String getState();
public void setState(Object state);

/** LEKEncoder クラスによってエンコード/デコードされた、整形された LEK 情報 */
public String getLekInfo();
public void setLekInfo(String lekInfo);

/** このフレームが属する事前に計算されたカテゴリ。TransactionFrame.CATEGORY_XXX を参
照 */
public int getCategory();
public void setCategory(int category);

/** ビット単位 OR 演算が実行された、情報のさまざまな内部要素の組み合わせ。
TransactionFrame.FLAG_XXX を参照 */
public long getFlags();
public void setFlags(long flags);
```

エージェント VSE API

VSE により、十分に定義された境界に沿って、プロセス、サービス、またはそれらの一部をスタブ アウトできます。 これらのプロセスおよびサービスの内部要素は、カスタム ユーザ定義ルールに従って DevTest によって実行されるレイヤと置換できます。 DevTest が実行するこれらのレイヤは、モデルと呼ばれます。 通常、これらのルールのデフォルトの開始点は、ライブ システムのインタラクションの記録から取得されます。

DevTest は、すでに、HTTP プロトコル（Web アプリケーションおよび Web サービスの仮想化を実現可能）、JMS、および JDBC に対して VSE をある程度までサポートしています。 エージェントは、サーバ プロセスの内部から直接仮想化を可能にする API を提供するため、仮想化はプロトコルに依存しません。 仮想化は、HTTP、JMS、JDBC だけでなく、RMI、EJB、または任意のカスタム Java オブジェクトに対しても有効にできます。

DevTest（またはエージェントのその他のクライアント）は、**AgentClient.getInstance().getVSEClient()** によって取得される **com.itko.lisa.remote.client.VSEClient** で定義された以下の API を使用して、この仮想化を実現できます。

注: エージェントの JavaDocs で、このクラスに関する詳細情報を表示できます。 JavaDocs は、**LISA_HOME¥doc** ディレクトリにあります。

```
/**
  * 実装が true の場合、名前が指定した正規表現に一致するすべてのクラス/インターフェースの名前のリスト、または、
  * 名前が指定した正規表現に一致するクラス/インターフェースを拡張/実装するすべてのクラス/インターフェースの名前のリスト
  * を返します。 アノテーションの検索は、以下の構文でサポートされています。
  * 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

 /**
  * すべての仮想化されたメソッドに対してレコーディング モードで onFrameRecord が呼び出され、
  * すべての仮想化されたメソッドに対して再生モードで onFramePlayback メソッドが呼び出される
  * 指定したエージェントに VSE コールバックを登録します。
  * @param info
```

```
 * @param callback
 */
public void registerVSECallback(IAgentInfo info, IVSECallback callback);

/**
 * 指定したエージェントへの VSE コールバックの登録を解除します。
 * @param info
 * @param callback
 */
public void unregisterVSECallback(IAgentInfo info, IVSECallback callback);

/**
 * 指定したエージェントで仮想化記録コールバックのコールを起動します。
 * @param agentInfo
 * @throws RemoteException
 */
public void startVSERecording(IAgentInfo agentInfo) throws
JMSInvocationException

/**
 * 指定したエージェントで仮想化再生コールバックのコールを起動します。
 * @param agentInfo
 * @throws RemoteException
 */
public void startVSEPlayback(IAgentInfo agentInfo) throws
JMSInvocationException

/**
 * 指定したエージェントで仮想化を停止します。
 * @param agentInfo
 * @throws RemoteException
 */
public void stopVSE(IAgentInfo agentInfo) throws JMSInvocationException

/**
 * 指定したエージェントで指定したクラス/インターフェースおよびそのすべての子孫を仮想化し
ます。
 * @param agentInfo
 * @param className
 * @return
 */
public void virtualize(IAgentInfo agentInfo, String className) throws
JMSInvocationException
```

コールバック API のインターフェースは、**com.itko.lisa.remote.vse.IVSECallback** によって定義されており、以下のメソッドが定義されています。

```
/**
```

```
  * これは、メソッドの仮想化がコールされたときに、VSE レコーディングがオンになっているエー
ジェント
  * によって呼び出されるメソッドです。VSE フレームには、既存の記録されたフレームと照合し、
その結果（および参照引数によって）
  * すべての情報が格納されます。
  * @param frame
  * @throws RemoteException
  */
 void onFrameRecord(VSEFrame frame) throws RemoteException;

 /**
  * これは、メソッドの仮想化がコールされたときに、VSE 再生がオンになっているエージェント
  * によって呼び出されるメソッドです。VSE フレームには、既存の記録されたフレームと照合し、
その結果（および参照引数によって）
  * を適切に設定できるようにするために必要なすべての情報が
  * 格納されます。
  * @param frame
  * @return
  * @throws RemoteException
  */
 VSEFrame onFramePlayback(VSEFrame frame) throws RemoteException;
```

最後に、**com.itko.lisa.remote.vse.VSEFrame** オブジェクトは、以下のプロパティのゲッターとセッターを備えた POJO です。

```
/** このフレームの一意の識別子を取得/設定します */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** このフレームの送信元のエージェント ID */
 public long getAgentGuid();
 public void setAgentGuid(long agentId);

 /** このフレーム メソッドが呼び出されたスレッド名 */
 public String getThreadName();
 public void setThreadName(String threadName);

 /** このフレーム メソッドが呼び出されたクラスの名前 */
 public String getClassName();
 public void setClassName(String className);

 /** VM のライフスパンに対してオブジェクトを追跡する一意の識別子 */
 public String getSourceId();
 public void setSourceId(String srcId);

 /** このフレームを格納している最も内側のセッションにスコープされたプロトコルのセッション
ID */
 public String getSessionId();
 public void setSessionId(String sessionId);
```

```
/** 呼び出されたメソッドの名前 */
public String getMethod();
public void setMethod(String method);

/** 呼び出されたメソッドの XStream が実行された引数の配列 */
public String[] getArgumentsXML();
public void setArgumentsXML(String[] argumentsXML);

/** 呼び出されたメソッドの XStream が実行された結果 */
public String getResultXML();
public void setResultXML(String resultXML);

/** メソッドが呼び出された（サーバ）時間 */
public long getTime();
public void setTime(long time);

/** メソッドの実行に要した時間 */
public long getClockDuration();
public void setClockDuration(long duration);

/** 再生モードで目的の結果を計算するために、getCode と ResultXML のどちらを使用するの
かを指定します */
public boolean isUseCode();
public void setUseCode(boolean useCode);

/**
 * isUseCode が true の場合にサーバで実行するコード。 オブジェクト（$0）およびメソッド
の引数（$1、$2 など）
 * にアクセスできる任意のコードを指定できます
 */
public String getCode();
public void setCode(String code);
```

エージェント API の例

エージェント API の例 1

以下のコードでは、クライアント コードからトランザクションを生成し、それによって生成されたトランザクション ツリーを取得します。

```
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!");
 }
```

上記のコードでは以下のユーティリティ関数を使用しています。これはエージェントとはまったく関係がありませんが、完全性のために記述されているものです。

```
/** Tomcat がローカルホスト 8080 で実行されていることを前提とします */
 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();
```

```
}
```

エージェント API の例 2

以下のコードでは、VSE ログ コールバックを登録します。

```
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) {}
 });
```

Java エージェント拡張

DevTest Java エージェントの以下のタイプの拡張を作成できます。

- エージェント拡張 (P. 76)
- ブローカの拡張 (P. 82)
- Java VSE 拡張 (P. 84)

エージェントの拡張

com.itko.lisa.remote.transactions.interceptors.IInterceptor インターフェースを実装する Java クラス、または **com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor** クラスを拡張する Java クラスを作成することにより、エージェントの動作をカスタマイズできます。詳細については、インストール ディレクトリの **doc** フォルダで、JavaDocs を参照してください。

```
public interface IInterceptor {
    /**
     * このカスタム インターセプタが現在無効かどうか
     */
    public boolean isDisabled();

    /**
     * インターセプトが戻る（true）か続行（false）するかを返します
     */
    public boolean block(boolean wayIn, Object src, String spec, String clazz,
String method, String signature, Object[] args, Object ret);

    /**
     * メソッドの開始後に呼び出され、インターセプタ ロジックに基づいて現在のフレームを変
更する可能性があります。
     */
    public boolean preProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);

    /**
     * メソッドの終了前に呼び出され、インターセプタ ロジックに基づいて現在のフレームを変
更する可能性があります
     */
    public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);
}
```

データのキャプチャを停止する場合は、**block()** メソッドを上書きします。この方法は **rules.xml** ファイルへの **exclude** ディレクティブの追加に似ていますが、より柔軟性があります。

エージェントがメソッドをキャプチャする直前にロジックを実行するようにする場合は、**preProcess()** メソッドを上書きします。

エージェントがメソッドをキャプチャした直後にロジックを実行するようにする場合は、**postProcess()** メソッドを上書きします。

これらのメソッドは、インターセプトされたか追跡されているすべてのクラスおよびメソッドに対して自動的に呼び出されます。 メソッドをインターセプトするかまたはクラスを追跡するには、**rules.xml** ファイルで前述の構文を使用して指定できます。 また、拡張クラスのコンストラクタでプログラムによってインターセプトを指定できます。 後の方法の利点は、拡張が自己完結型であるということです。

block() メソッドの最初の引数は **boolean wayIn** です。 **block()** メソッドは、1 つのメソッドあたり 2 回コールされます（入口で 1 回、出口で 1 回）。入口でのコール時の **wayIn** 引数の値は true です。出口でのコール時の **wayIn** 引数の値は false です。

以下では、**block()**、**preProcess()**、および **postProcess()** メソッドに共通の引数について説明します。

引数	説明
Object src	メソッドがコールされているオブジェクト。
String spec	API をインスツルメントするように指定されたクラスまたはインターフェース名。
String clazz	インターセプトされたメソッドを定義するクラスの名前。
String method	インターセプトされたメソッドの名前。
String signature	インターセプトされたメソッドのシグネチャ（JVM 形式）。
Object[] args	インターセプトされたメソッドに渡される引数。
Object ret	インターセプトされたメソッドの戻り値。 wayIn 引数が true の場合、戻り値は NULL です。

src、**spec**、および **clazz** 引数の違いについては、以下の例で説明できます。

以下のインターフェースおよびクラス定義があると仮定します。

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

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

public class C extends B {
}
```

ルールで **intercept("A", "m", "*")** を指定し、コードが **C.m()** をコールする場合、以下の情報は true です。

- **spec** は A である
- **clazz** は B である
- **src** は C のインスタンスである

展開

拡張を展開するには、拡張をコンパイルして、以下のエントリを含むマニフェストと共に JAR ファイルにパッケージ化します。

```
Agent-Extension: extension class name
```

エージェントの JAR ディレクトリにこの JAR を配置すると、エージェントは自動的にそれを取得します。 エージェントが起動した後にファイルを追加した場合、ホット ロード メカニズムによってファイルが確実に適用されます。 エージェントの実行中に拡張 JAR を更新した場合、JAR クラスが動的に再ロードされます。 動的な再ロードにより、サーバを再起動せずに、拡張コードをすばやく簡単にテストできます。

拡張 JAR は、拡張クラスで使用されるクラスをすべて参照できるクラスローダによってロードされます。 使用するクラスを定義する **LisaAgent.jar** およびすべてのコンテナ JAR に対して、拡張ソースをコンパイルできます。 拡張にリフレクションを使用する必要はありません。

例

以下の例では、**block()** メソッドを使用して、スレッド名が **Event Sink Thread Pool** で始まるメソッドをエージェントがキャプチャしないようにします。

```
public class MyInterceptor extends AbstractInterceptor {

   /** このコールをインターセプトしない場合は true を返します */
   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);
   }

   ...

}
```

以下の例では、**postProcess()** メソッドを使用して、トランザクション ウィンドウの［応答］行のデータをキャプチャします。 この例では、**com.itko.lisa.remote.transactions.TransactionFrame** クラスの **setResponse()** メソッドをコールしています。 詳細については、インストール ディレクトリの **doc** フォルダで、JavaDocs を参照してください。

```
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);
   }
```

以下の例では、WebMethods の **com.wm.data.IData** オブジェクトが Integration Server のフローで呼び出されたときにそのオブジェクトの XML コンテンツを出力する方法を示します。 エージェントはすでに WebMethods をサポートしているので、そのための拡張は必要ありません。

```
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);

 }
```

```
}
```

この拡張をビルドするには、**LisaAgent.jar**、**wm-isclient.jar**、および **wm-isserver.jar** に対してこのコードをコンパイルし、以下の行を含むマニフェストと共にクラス ファイルを JAR にまとめます。

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

トランザクション フレームへのタグの追加

トランザクション フレームに任意のキー/値ペアを関連付けることができます。 キー/値ペアは、タグと呼ばれます。

TransactionFrame オブジェクトは **setTag()** メソッドを提供します。 このメソッドには、2 つの文字列引数（キーおよび値）があります。

以下のコードは、エージェント拡張の **postProcess()** メソッドの内部からタグを追加する方法を示しています。

```
// postProcess 内のどこか
frame.setTag("タグ名", "タグ値");
```

たとえば、Web サイトへのログイン時に要求パラメータ **username** の値を取得し、**frame.setTag("ユーザ名", value)** をコールする拡張を作成できます。

タグはデータベース内の **FRAME_TAGS** テーブルに格納されます。

CAI コンソールでタグによってフィルタできます。 詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

ブローカの拡張

ブローカ拡張は、トランザクション フラグメントがエージェントから受信された後、ブローカによってロードおよび呼び出されます。

ブローカ拡張によって、以下のタスクを実行できます。

- フレームに含まれているデータを変更する。
- 特定のフレームを追加または除去する。
- ステッチング アルゴリズムをカスタマイズする。 ステッチング アルゴリズムは、トランザクション フラグメントをアセンブルする方法を定義します。

ブローカを拡張するには、**com.itko.lisa.remote.plumbing.IAssemblyExtension** インターフェースを実装します。 このインターフェースは、**onTransactionReceived()** メソッドを定義します。 詳細については、インストール ディレクトリの **doc** フォルダで、JavaDocs を参照してください。

```
public interface IAssemblyExtension {
     /**
      * アセンブリの前に呼び出されるメソッド
      * @param frame： エージェントから受信される部分的なトランザクション ルート
      * @return： 標準のアセンブリをバイパスする（拡張によってアセンブリを処理する）場合
は true
      */
     public boolean onTransactionReceived(TransactionFrame frame);
 }
```

ブローカ拡張を展開するには、拡張をコンパイルして、以下のエントリを含むマニフェストと共に JAR ファイルにパッケージ化します。

```
Broker-Extension: extension class name
```

ブローカ（レジストリ）ディレクトリにこの JAR を配置すると、ブローカは自動的にそれを取得します。 ブローカ（レジストリ）が起動した後にファイルを追加した場合、ホット ロード メカニズムによってファイルが確実に適用されます。ブローカ（レジストリ）の実行中に拡張 JAR を更新した場合、JAR クラスが動的に再ロードされます。動的な再ロードにより、ブローカ（レジストリ）を再起動せずに、拡張コードをすばやく簡単にテストできます。

例

典型的な使用例は、TCP/IP およびポートを使用する通常のステッチング アルゴリズムが、エージェント間に配置された仮想 IP が割り当てられているロード バランサによって混乱する場合、またはエージェントがネイティブ ライブラリを使用して IO を実行し、使用中の IP およびポートに直接アクセスできない場合です。その場合、フレームのアドレスおよびポートのフィールドが空白のままか、正しくありません。このため、拡張でフレームをアセンブルする必要があります。

```
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);
}
}
```

Java VSE の拡張

Java VSE 拡張では、以下のいずれかのメソッドを上書きできます。

- **onPreRecord()：** このメソッドは、レコーディング中に仮想化されたメソッドが実行を開始する前にコールされます。
- **onPostRecord()：** このメソッドは、レコーディング中に仮想化されたメソッドが実行を停止した後にコールされます。
- **onPreHijack()：** このメソッドは、再生中に仮想化されたメソッドが VSE に渡される前にコールされます。
- **onPostHijack()：** このメソッドは、再生中に仮想化されたメソッドが VSE から戻った後にコールされます。
- **onNewStream()：** このメソッドは、オブジェクトが XML に変換されるたびにコールされます。

例

以下の例では、**onPostRecord()** メソッドを使用して、レコーディング中にクラスの名前を変更します。 この例では、**com.itko.lisa.remote.vse.VSEFrame** クラスの **setClassName()** メソッドをコールしています。 詳細については、インストール ディレクトリの **doc** フォルダで、JavaDocs を参照してください。

```
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);
   }

}
```

ロード バランサおよびネイティブ Web サーバ

部分的なトランザクションから完全なトランザクションを組み立てるプロセスは、ステッチングと呼ばれます。

エージェントが、ロード バランサまたはネイティブ Web サーバが存在するネットワーク上に展開されている場合、CAI が使用するステッチング アルゴリズムが正しく機能しないことがあります。 このシナリオでは、ブローカの **rules.xml** ファイルに **loadbalancer** ディレクティブを追加します。 エージェント間にインストールされているアプライアンスごとに、そのアプライアンスの IP アドレスを指定します。 以下に例を示します。

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

loadbalancer ディレクティブは、**broker** エレメント内に配置する必要があります。

別の方法は、［**常に待機**］プロパティを有効にすることです。

このプロパティは、DevTest ポータルの［エージェント］ウィンドウから設定できます。 プロパティは［設定］タブに表示されます。

Java エージェントのセキュリティ

DevTest Java エージェントでは、拡張やリモート コード呼び出しのためのセキュリティ メカニズムが提供されています。

エージェントは、独自のセキュリティ マネージャをインストールします。カスタム エージェント コードはすべて、セキュリティ マネージャによって適用される特殊な権限で実行されます。 アプリケーションがすでに独自のセキュリティ マネージャを使用している場合は、エージェントのセキュリティ マネージャが既存のセキュリティ マネージャをラップし、通常のアプリケーション コードについてはそれに委任します。

拡張

CA Continuous Application Insight と CA Service Virtualization のエージェント拡張の実行には、以下の制限があります。

- ファイル アクセスは、エージェント ディレクトリと一時ディレクトリに制限されます。
- プロセスの作成は無効です。
- プロセスの終了は無効です。

その結果、テスト中のシステムは、コンピュータの残りの部分からサンドボックス化されます。

一部のアプリケーションは、セキュリティ マネージャが null であるかどうかを明示的にチェックし、その結果に応じて異なるコード パスを使用することができます。 この状況では、エージェントのセキュリティ マネージャによって解決不能な問題が発生する場合があります。 以下のいずれかの方法を使用して、エージェントのセキュリティ マネージャを無効にすることができます。

- コマンドラインで **-Dsecurity.manager.disabled=true** を指定します。
- **Disable security manager** プロパティが有効であることを確認します。

このプロパティは、DevTest ポータルの［エージェント］ウィンドウから設定できます。 プロパティは［設定］タブに表示されます。

注: セキュリティ マネージャを無効にすると、許可のチェックが無効になります。 ただし、トークン認証は引き続き有効にすることができます。

リモート コード呼び出し

エージェントはリモート システムに配置されるため、エージェント API はすべてリモート コード呼び出しを通して機能します。

リモート コード呼び出しのセキュリティは、トークンを使用して処理されます。

エージェントに対して 1 つまたは 2 つのトークンを定義するには、**token** オプションを使用します。最初のトークンは、管理者ロールを表します。2 番目のトークンは、ユーザ ロールを表します。 2 つのトークンを指定する場合は、それらをコロンで区切ります。

例：

- **token=asdf1** は、**asdf1** の値を持つ管理者トークンを指定します。
- **token=asdf1:asdf2** は、**asdf1** の値を持つ管理者トークンと、**asdf2** の値を持つユーザ トークンを指定します。

管理者トークンまたはユーザ トークンには、カンマ、等号、および空白文字を除く任意の文字を含めることができます。 トークンの最大長は 16 文字です。

トークンの概念は、コンソールで使用できます。 これは、コマンドライン構文 **-Dlisa.token=xxxx** または **-Dlisa.token=xxxx:xxxx** で指定できます。 コンソールの場合は、専用のセキュリティ マネージャが存在しないため、どちらのトークンにもすべての権限があります。 コンソール トークンを指定しないと何も実行できなくなるという点で、セキュリティが実現されます。

SSL を使用するための Java エージェントの設定

エージェントとブローカの間の通信を保護するために、SSL（Secure Sockets Layer）を使用できます。

ブローカは、レジストリの SSL 用の設定を継承します。 エージェントは、同じ暗号化されたパスワードおよび同じキーストアを使用します。

以下の方法のいずれかを選択します。

デフォルト キーストア

この方法は、DevTest でデフォルト キーストアを使用します。

次の手順に従ってください:

1. **LISA_HOME** ディレクトリの **local.properties** ファイルを開き、以下の行のコメントを外します。

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

2. **local.properties** ファイルを保存します。
3. レジストリを起動または再起動します。
4. ブローカ URL を指定する場合は、**ssl** スキームを使用します。以下に例を示します。

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

カスタム キーストア

この方法は、カスタム キーストアに基づいています。

注: DevTest ポータルの詳細情報については、「CA Continuous Application Insight の使用」を参照してください。

次の手順に従ってください:

1. 「通信を保護するための SSL の使用」の手順に従って、カスタム キーストアを設定します。
2. **LISA_HOME¥agent** ディレクトリにキーストアを配置します。
3. DevTest ポータルを開きます。

4. 左ナビゲーション ペインで、［設定］-［エージェント］を選択します。
5. 左側の部分で、エージェントを選択します。
6. ［設定］タブをクリックします。
7. 以下のセキュリティ プロパティを設定して、DevTest プロパティをミラーリングするようにカスタム キーストアを設定します。暗号化されたパスワードは、**local.properties** ファイルからコピーできます。
   - キーストアの場所
   - キーストア パスワード（暗号化）
   - 信頼ストアの場所
   - 信頼ストア パスワード（暗号化）
8. 左側の部分で、ブローカを選択します。
9. エージェント用と同じプロパティを設定します。キーストアのパスは異なります。
10. ブローカ URL を指定する場合は、**ssl** スキームを使用します。以下に例を示します。

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

Java エージェントのログ ファイル

以下のログ ファイルを使用して、問題のトラブルシューティング (P. 92) に役立てることができます。

- エージェント ログ ファイルには、**devtest_agent_pid.log** という名前が付けられます。 このファイルは、**LisaAgent.jar** ファイルと同じディレクトリに書き込まれます。
- ブローカが開始されるたびに、**devtest_broker_pid.log** という名前のログ ファイルが作成されます。 また、ブローカは、**pfbroker.log** という名前の統合されたログ ファイルを持ちます。 これらのファイルは、メイン DevTest ログ ファイルと同じディレクトリに書き込まれます。
- コンソール ログ ファイルには、**devtest_agent_console_pid.log** という名前が付けられます。 DevTest ワークステーション にはログ ファイルがあり、DevTest ポータル、各シミュレータ、および VSE にもログ ファイルがあります。 これらのファイルは、メイン DevTest ログ ファイルと同じディレクトリに書き込まれます。

ファイル名の **pid** の部分は、ログ記録を実行しているプロセスのプロセス ID です。

注: メイン DevTest ログ ファイルとその場所については、「*管理」を参照してください。*

以下の設定プロパティで、ログ動作を制御できます。

Java ログ レベル

起動時にログ レベルを設定します。

最大ログ サイズ

エージェント ログの最大サイズを設定します。このサイズを超えるとロールオーバされます。

最大ログ アーカイブ数

ロールオーバされたアーカイブ済みログの最大保持数を設定します。

エージェント ログ

デフォルトの場所をオーバーライドします。

これらのプロパティは、DevTest ポータルの ［エージェント］ ウィンドウから設定できます プロパティは、［設定］ タブに表示されます。

ユーザが **rules.xml** ファイルからのこれらのプロパティを設定する必要がある場合、対応するプロパティ名は以下のとおりです。

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

Java エージェントのトラブルシューティング

重要: ターゲット環境を事前に調査し、それがテストされていることを確認するか、または自分自身でテストしてください。Java エージェントの問題のほとんどは、アプリケーションではなく、OS または JVM に起因しています。 このドキュメントの指示に従うようにしてください。 それが役に立たない場合に備えて、このトピックでは、最も一般的な問題について説明しています。

DevTest Java エージェントに関連する最も重大な問題は起動時に発生します（エージェントが見つからない、プロセスがクラッシュまたはハングアップするなど）。一般に、起動時の問題を切り抜けたら、トラブルシューティングとしては快調です。 そこから、設定を調整したり、拡張を記述したりしていくことになります。

注:

- エージェント環境でサーバを再起動して、さまざまな処理を試してみることは困難な場合があります。通常は、ターゲット コンピュータおよび JVM で **java <エージェント オプション> -version** を実行する方が簡単であり、やってみる価値があります。この方法は、問題が OS/JVM またはコンテナ/アプリケーションのどちらに固有であるかを示します。
- インストール プロセスに役立つコマンドライン ユーティリティについては、「エージェント インストール アシスタントの使用 (P. 21)」を参照してください。

起動時のエラー： VM の初期化中にエラーが発生する。 絶対パスにエージェント ライブラリが見つからない。

ライブラリが実際にそのパスにあり、Java と同じアーキテクチャを使用していることを確認してください（32 ビットと 64 ビットの両方）。

また、ライブラリの何からの依存関係が欠けていないことも確認してください。たとえば、Win32 では **depends.exe**、Mac OS X では **otool -L**、Linux および UNIX では **ldd -d** を使用します。 ライブラリが欠けている場合は、**LD_LIBRARY_PATH** が、ライブラリが存在するディレクトリを含むように設定されていることを確認します。 コンテナが、その起動スクリプトで **LD_LIBRARY_PATH** を上書きする場合があります。 これをスクリプトやコンテナ固有の管理ツールからではなく、シェルで設定している場合は、正しく設定されていると思いこまないでください。

ldd が正常に返されない場合、エージェントは正しく実行されません。 そのため、**ldd** の確認は、解決するために行うべき最初のことです。オペレーティング システムのエージェント バージョンが見つからない場合は、Pure Java エージェントの使用を検討してください。

エージェントが直ちに（または、プロセスを起動した後すぐに）終了する。

「**LISA AGENT: VM terminated**」というメッセージが表示される場合は、おそらくプロセスは正常に終了しています。 いくつかのコンテナはランチャ プロセスを含んでおり、それが直ちに終了することは正常な動作です。

このメッセージが表示されないか、または（**ldd** が正常に返された後に）クラッシュ ダンプが発生する場合は、エージェントのバグが存在する可能性があります。 サポートに連絡し、Pure Java エージェントを試してください。依然としてクラッシュまたはダンプが発生する場合は、JVM のバグが存在する可能性があります。このような状態は、一部のオペレーティング システム上の IBM JVM 1.5 の場合、スレッドをインスツルメントしようとして発生します。 その場合は、コマンドラインの JVM 引数 **-Dlisa.debug=true** を指定してみてください。

エージェントが「**GetEnv on jvmdi returned -3 (JNI_EVERSION)**」というメッセージで終了する。

JVM にそのデバッグ対応ライブラリを使用するよう指示するために、コマンドライン オプション **-Xsov** を指定してみてください。それが機能しない（たとえば、無効なオプションのエラー メッセージが表示される）場合、このオペレーティング システムはサポートされていません。

エージェントが「UTF ERROR" ["../../../src/solaris/instrument/EncodingSupport_md.c":66]: ...」というメッセージで終了する。

このメッセージは、オペレーティング システムまたは JVM、あるいはその両方の正確なバージョンによって異なる場合があります。 このメッセージは通常、次の要素のいずれかを含んでいます： 「UTF ERROR ["../../../src/solaris/instrument/EncodingSupport_md.c":66]: Failed to complete iconv_open() setup」。

この問題は、いくつかの言語パックがインストールされていない場合、一部の Solaris JVM にあるバグのために発生します。 この問題を修正するには、最初に **pkg install SUNWlang-enUS**、次に **export LANG=en_US.UTF-8** を実行することによって、en-US 言語パックをインストールします。あるいは、ネイティブ エージェントを使用します。

起動時にエージェントがハングアップするか、または多数の例外をスローする（LinkageErrors、CircularityErrors など）。

Java を使用した Java バイトコードのインスツルメントの副作用は、最初の方のクラス（**java.*** など）のクラスロードの順番の一部がわずかに変更され、それによりデッドロックまたはバイトコード検証エラーが発生する場合があることです。

JVM とオペレーティング システムのすべての組み合わせについて、これらの問題の既知の発生のすべてを排除しています。 ただし、テストされていない組み合わせが発生している可能性があります。 この問題をサポートに連絡してください。 この問題がハングアップである場合は、サポートの問題にスレッド ダンプを含めてください。Windows では Ctrl + Break、UNIX/Linux では Ctrl + ¥ または kill -3 <pid> を入力することによって、スレッド ダンプを生成します。 また、Java エージェントはクラスロードの順番が少し異なるため、それも試してみることができます。 あるクラスまたはパッケージがハング スレッドに常に含まれていると考えられる場合は、そのための **exclude** ディレクティブを **rules.xml** ファイルに追加してみてください。

エージェントが java.lang.VerifyErrors をスローするか、またはホット スワップが有効にならない。

一部の古い 1.4 JVM には、ホット スワップ（ロードされた後のクラスのインスツルメント）のサポートにバグがあります。

その場合、**Enable hot instrumentation** プロパティを無効にすることにより、ホット スワップをオフにします。

このプロパティは、DevTest ポータルの［エージェント］ウィンドウから設定できます。 プロパティは［設定］タブに表示されます。

インターセプトまたは仮想化するクラスまたはメソッド、あるいはその両方を前もって決定し、これらのクラスまたはメソッドのルールを **rules.xml** ファイルに追加して、サーバを再起動する必要があります。 このプロセスはライブ サーバ上での実行に比べて面倒ですが、これがこの問題を回避するためのわかっている唯一の方法です。

ログに **java.lang.VerifyError** が見られない場合もありますが、エージェントは、指定されたクラスがインスツルメントされていないかのように動作するか、またはランダムな結果（クラッシュなど）を生成します。

エージェントは起動するが、コンソールまたはブローカがエージェントを認識できない。

この原因は、通常はエージェントとブローカの間のファイアウォールまたはポートの問題です。

エージェント ログの先頭に、「**Can't connect to broker at tcp://ip:port**」という警告が含まれている場合があります。 エージェントが使用している IP アドレスとポートが正しいことを確認してください。 次に、ブローカが、指定された IP アドレスの指定されたポートでリスンしていることを確認してください。 **netstat -ano | grep port** によって、指定された IP または 0.0.0.0 でリスンしているポートが表示されるはずです。 最後に、エージェント コンピュータから **telnet ip port** を実行してブローカを認識できることを確認することにより、ファイアウォールの問題を見つけてください。 ブローカを認識できる場合は、ブローカが異常な状態にある可能性があります。 レジストリとブローカ ログを確認し、必要に応じてブローカを再起動します。

エージェントのために一部の操作がタイムアウトする。

一部の JVM （特に IBM JVM）は、その一部のネットワーク クラスがインスツルメントされた後、正しく動作しません。 その結果、ネットワーク呼び出しが、明確な理由なしで失敗またはタイムアウトする場合があります。

これらのクラスがインスツルメントされないようにするには、コマンドラインの JVM 引数 **-Dlisa.debug=true** を指定してみてください。

問題が解決されない場合は、**rules.xml** ファイルを編集して、以下のネットワーク パッケージを除外します。

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

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

LisaAgent.jar が使用可能で、読み取り権限を持ち、かつ破損していないことを確認してください。最も簡単な確認方法は、**java -jar LisaAgent.jar -v** を実行することです。

アプリケーションが OSGi を使用しているかどうかを確認してください。 使用している（JBoss 7 でのように）場合は、システムまたはブートストラップ パッケージに **com.itko** を追加します。 **com.itko** を追加する方法はコンテナに依存するため、特定の手順を示すことは困難です。 ただし、これは通常、パッケージのリストまたは類似の JVM 引数を指定するプロパティを含む設定ファイルです。

エージェントを有効にすると、セキュリティ関連の例外がスローされる。

アプリケーションによってスローされた SecurityExceptions または PermissionExceptions が見られる可能性があるのは、セキュリティが有効な状態でエージェントを有効にした場合だけです。この設定は現在、デフォルト設定になっています。

この理由と回避策については、「Java エージェントのセキュリティ (P. 86)」で説明しています。

異常なリソース消費（CPU、メモリ、ファイル ハンドルなど）。

CPU 使用率が異常に高いか、または定期的に急上昇する場合は、障害のある（1 つまたは複数の）スレッドの特定に役立つため、急上昇の期間に注意してください。 また、CAI と VSE も無効にしてみてください。

OutOfMemory エラーが発生する場合は、Java ヒープ使用率をモニタします それが **-Xmx** の制限を超えている場合は、その制限を増やします。ただし、すでに高くなっており、エージェントなしでの通常のアプリケーション使用率を優に超えている場合、制限を増やすことは避けてください。 その場合は、ヒープ ダンプを生成します。WebSphere 用の WAS HeapDump ユーティリティ、または古いバージョン用のフリーの Eclipse MAT ツールを使用できます。 エラーの時点でメモリ使用率が **-Xmx** の制限に達していない場合は、ネイティブ コード内のリークである可能性があります。 この場合は、サポートに連絡してください。

特に UNIX または Linux で不明な IOExceptions（ファイル ハンドルが多すぎるなど）が発生する場合は、ボックスの ulimit を確認してください（**ulimit -n -H** および **ulimit -n -S**）。 この値が低い場合は、ボックスの管理者に増やすよう依頼することを検討してください（最新の J2EE アプリケーションでは、4096 未満は低いと見なされます）。 その後、再起動することを忘れないでください。

エージェントは、制限に近づいたらガベージ コレクションをトリガすることによって、ファイル ハンドルの数を制限未満に維持しようとします。起動時に制限を読み取ることができない場合、エージェントはデフォルト値の 1024 を使用します。 このデフォルト値を使用すると、過剰なガベージ コレクションや、ほぼランダムで、頻繁な CPU 使用率の急上昇が発生する場合があります。 この状況は、ログ内の以下のメッセージによって示されます。

```
Max (or preferred) handles limit approaching - triggering GC...
```

その場合、［**ハンドルの最大数**］プロパティの値が増加します。

このプロパティは、DevTest ポータルの［エージェント］ウィンドウから設定できます。 プロパティは［設定］タブに表示されます。

エージェントが起動したことは確認できるが、CAI データがないか、または完全ではない。

データ ライフサイクル プロセスには、以下の段階が含まれます。

- エージェントでのトランザクションのキャプチャ
- ブローカへの転送
- ブローカでの部分的なトランザクションのアセンブリ
- コンソールへの転送
- データベースへの保存
- データベースからの取得

欠落したデータや不完全なデータは、このいずれかの段階での問題の結果である場合があります。

キャプチャ例外がないかどうか、エージェント ログを確認してください。転送または保存例外がないかどうか、ブローカ ログおよびコンソール ログを確認してください。

例外が存在せず、依然として欠落したデータを見つけることができない場合は、エージェントでの debug または dev ログを有効にしてください。「Sent partial transaction」などのステートメントが表示されない場合は、CAI がおそらく有効になっていません。

これらのいずれの手順によっても最終的な結果が得られない場合は、サポートに連絡してください。

Java VSE のレコーディングまたは再生を有効にしたが、VSE がエージェントからの要求をまったく受信しない。

まず、エージェントが VSE のレコーディングまたは再生モードにあることを確認してください。 エージェント ログを開き、「Starting VSE record/playback...」を検索します。

次に、エージェント ログと VSE ログで例外を探します。 存在しない場合は、仮想化されていると考えているクラスが仮想化されていない可能性があります。 エージェント ログで「Virtualized com.xxx...」などのステートメントを探します。このステートメントがない場合は、アプリケーションがまだそのクラスをロードしていない可能性があります。 別の可能性として、Java 1.4 の古いバージョンを使用している場合は、一部の機能でホット スワップがサポートされていない場合があります。Java VSE は、デフォルトでホット スワップを使用します。その場合、エージェントの **rules.xml** ファイルで仮想化されたクラスを指定し、再起動します。

Java VSE のその他の問題。

Java VSE の実行中に何からの問題（機能的な問題、または異常なリソース使用率）が発生した場合は、エージェント側で CAI を無効にします。

［**自動起動**］プロパティを無効にすることにより、CAI をオフにできます。次に、JVM を再起動します。

このプロパティは、DevTest ポータルの［エージェント］ウィンドウから設定できます。 プロパティは［設定］タブに表示されます。

ブローカ側で CAI を無効にしないでください。そうすると、Java VSE が完全に機能を停止します。

ケース機能が機能していない。

まず、エージェントがブローカに接続していることを確認してください。

エージェントがブローカに接続している場合は、問題が発生しているページの HTML ソースを確認します。そのページの下部に、使用している変数名（**com_itko_pathfinder_defectcapture_xxx** など）で容易に識別できる JavaScript のブロックが存在することを確認してください。 このブロックがない場合は、エージェント ログに例外がないかどうかを確認します。ブロックが存在しないその他の原因には、以下のものがあります。

- ページの HTML に異常がある。たとえば、HTML タグが欠落している。
- エージェントにページ キャプチャの問題がある。これは、未テストのコンテナや静的なページで発生する場合があります。

ブロックが存在する場合は、Java コンテナの前にネイティブ Web サーバ（Apache など）またはロード バランサが存在するかどうかを確認してください。 その場合は、それを CAI JavaScript の要求とリソース ファイルを Java コンテナに転送するように設定します。 これにより、その URL の一部として **defectcapture** という単語が含まれます。 IT 管理者は一般的に、このタスクの実行方法を知っています。

DevTest は **IBM DB2** データベースを使用するように設定されます。
DevTest ポータルで、パス グラフ内の **JDBC** ノードを選択しました。
［**Statements**］および［**接続 URL**］列はブランクです。

JDBC 接続 URL に **progressiveStreaming=2** 文字列を追加して、プログレッシブ ストリーミングを無効にします。 以下に例を示します。

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

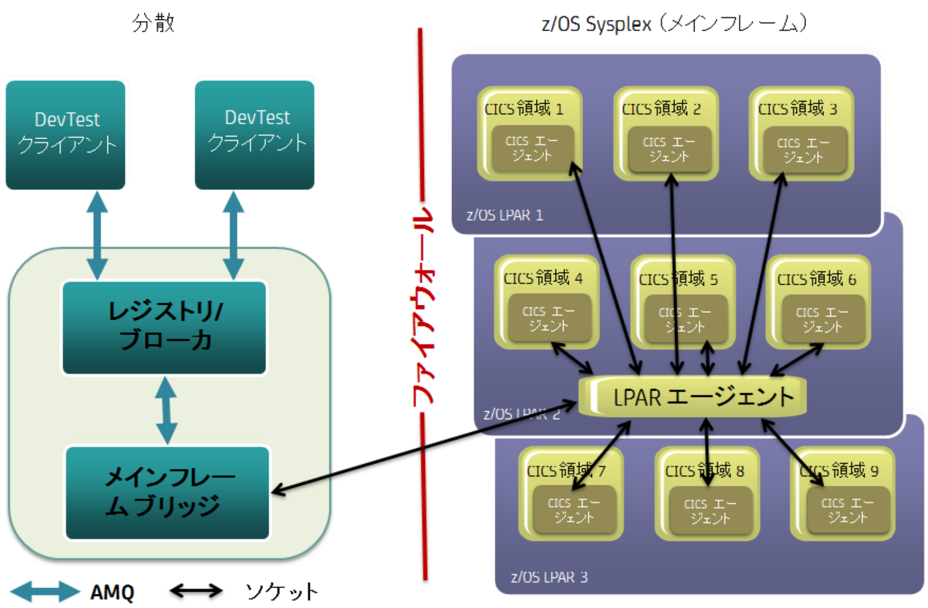
第 2 章: メインフレーム ブリッジ

メインフレーム ブリッジは、DevTest クライアントが DevTest メインフレーム エージェントと通信できるようにするコンポーネントです。

このセクションには、以下のトピックが含まれています。

メインフレーム ブリッジのアーキテクチャ

以下の図は、メインフレーム ブリッジがその他のコンポーネントとどのように対話するかを示しています。



接続が確立された後、トラフィックは DevTest クライアントからブローカへ、さらにメインフレーム ブリッジから実際のメインフレームに移動します。 トラフィックは、逆の順序で移動することもできます。

メインフレーム ブリッジのルール

以下のエージェント プロパティをメインフレーム ブリッジに対して設定する必要があります。

コード ページ

ASCII と EBCDIC の間の変換に使用されるコード ページ。

デフォルト値は「Cp1047」です。

初期パケット バッファ サイズ

メインフレームからのパケットを処理するための初期バッファのサイズ（バイト単位）。 より大きなサイズのパケットを受信した場合は、バッファ サイズが増加されます。

デフォルト値は 1024*65 です。

パケット トレース（LPAR）

LPAR エージェントとの間で送受信されるパケットをトレースするかどうかを指定します。

デフォルトでは、トレースは無効です。

パケット トレース（クライアント）

ワークステーションとの間で送受信されるパケットをトレースするかどうかを指定します。

デフォルトでは、トレースは無効です。

クライアント接続タイムアウト

ブリッジがクライアント モードで起動され、LPAR エージェントへの接続が失敗した場合に、接続の再試行を待機する秒数。

デフォルト値は 30 です。

これらのプロパティは、DevTest ポータルの ［エージェント］ウィンドウから設定できます プロパティは、［設定］タブに表示されます。

メインフレーム ブリッジのプロパティ

LISA_HOME ディレクトリの **local.properties** ファイルに、以下のプロパティを追加する必要があります。

lisa.mainframe.bridge.enabled

レジストリが起動されたときにメインフレーム ブリッジを起動するかどうかを指定します。 ブリッジは外部プロセスとしてではなく、レジストリの内部で実行されます。

デフォルト： false

lisa.mainframe.bridge.mode

メインフレーム ブリッジの実行モードを指定します。 このモードによって、LPAR エージェントとのピア関係が定義されます。ブリッジは、クライアント モードまたはサーバ モードで実行できます。

- クライアント モードでは、LPAR エージェントをサーバ モードで実行されるように設定する必要があります。 ブリッジは、LPAR エージェントへの接続を開始します。
- サーバ モードでは、LPAR エージェントをクライアント モードで実行されるように設定する必要があります。 ブリッジは、LPAR エージェントからの接続を待機します。

値

- クライアント
- サーバ

デフォルト： server

lisa.mainframe.bridge.port

サーバ モードでは、このプロパティは、ブリッジが LPAR からの接続をリスンする「既知のポート」を指定します。 このプロパティは、クライアント モードでは無視されます。

値： 有効なポート番号。

デフォルト： 61617

lisa.mainframe.bridge.server.host

クライアント モードでは、このプロパティは、ブリッジが接続を開始する際の接続先ホストを定義します。このプロパティは、サーバ モードでは無視されます。

値： 有効なポート番号。

デフォルト： localhost

lisa.mainframe.bridge.server.port

クライアント モードでは、このプロパティは、指定されたホスト上でブリッジが接続を開始する際の接続先ポートを定義します。 このプロパティは、サーバ モードでは無視されます。

値： 有効なポート番号。

デフォルト： 3997

lisa.mainframe.bridge.connid

このプロパティは、将来の使用に備えて予約されています。

DevTest z/OS エージェント コンソール

メインフレーム ブリッジは、サーバ コンソールを通じて使用可能なコンソールを提供します。このコンソールは、DevTest CICS エージェントに対して、限定された可視性および管理を提供します。 この機能により、ユーザまたは管理者は、z/OS にアクセスせずに CICS エージェントの限定された管理を実行できます。 z/OS または CICS システム プログラマが、CICS エージェントの包括的な管理を実行する必要があります。

z/OS エージェントコンソールは、CICS エージェントに対して、限定された可視性および管理のために以下の機能を提供します。

- DevTest CICS エージェントがアクティブであることの保証
- CICS エージェントの一部の属性の表示
- CICS 領域の一部の属性の表示
- CICS エージェントの仮想化のレコーディングおよび再生アクティビティのリスト表示
- CICS エージェントの仮想化のレコーディングおよび再生アクティビティのキャンセル/リセット

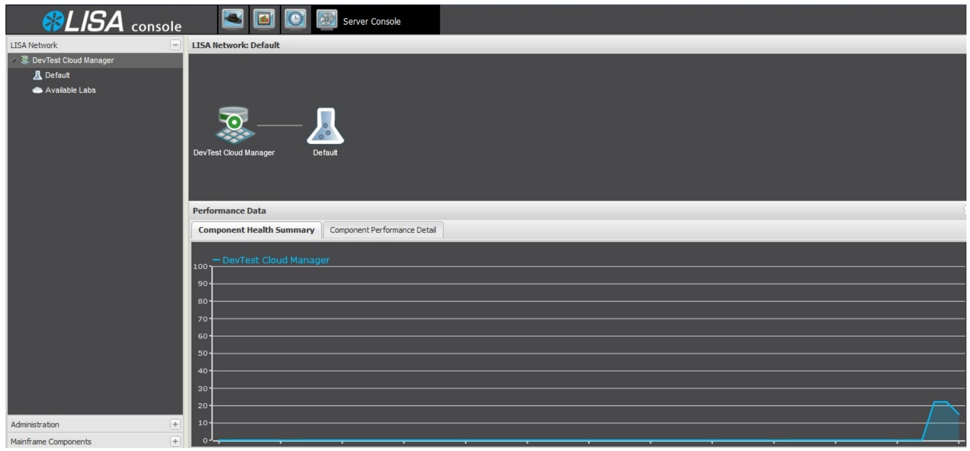
z/OS エージェント コンソールは、z/OS または CICS システム プログラマがエージェント操作を実行するための代替品となることを意図するものではありません。

このコンソールでは、以下の操作は許可されていません。

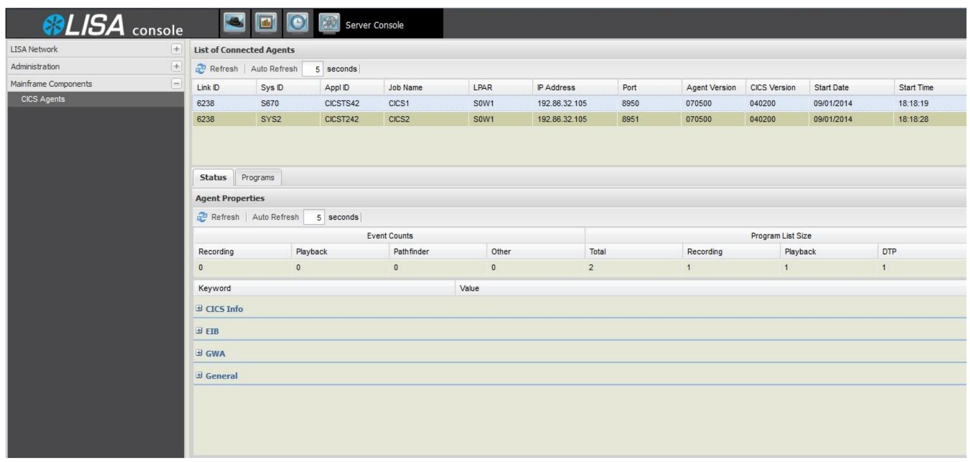
- DevTest z/OS エージェントの起動、停止、および再起動
- DevTest z/OS エージェント設定の変更

DevTest コンソールの起動

メインフレーム ブリッジが有効な場合、DevTest コンソールにはメインフレーム コンポーネント ツリーが存在します。



メインフレーム コンポーネント ツリーを展開するには、［+］をクリックします。 接続されている z/OS CICS エージェントが表示されます。



上部ペインでエージェントを選択し、下部ペインでエージェントの表示または管理を行います。

下部ペインには、［ステータス］および［プログラム］の 2 つのタブがあります。

これらのタブの下には、いくつかのエージェント トランザクション数が表示されています。 この場合の「トランザクション」は、CICS トランザクション全体ではなく、単一の CICS イベント（CICS LINK など）または DTP ALLOCATE を意味しています。

イベント数

レコーディング

エージェントの初期化以降に記録されたトランザクションの数が表示されます。

再生

エージェントの初期化以降に仮想化されたトランザクションの数が表示されます。

Pathfinder

エージェントの初期化以降に、CAI にレポートされたトランザクションの数が表示されます。

その他

エージェントの初期化以降に発生したその他のトランザクションの数が表示されます。

プログラム リスト サイズ

合計

プログラム リスト内のエントリの総数が表示されます。 プログラム リストには、記録または仮想化されるトランザクションごとに 1 つのエントリが入っています。

レコーディング

プログラム リスト内のレコーディング モードのトランザクションの数が表示されます。

再生

プログラム リスト内の再生（仮想化）モードのトランザクションの数が表示されます。

DTP

プログラム リスト内の分散トランザクション処理（DTP）であるトランザクションの数（レコーディングおよび再生の両方）が表示されます。

[ステータス]タブ

［ステータス］タブには、エージェント属性が表示されます。

CICS 情報ツリーには、エージェントが実行されている CICS 領域からのいくつかの CICS 設定情報があります。

EIB ツリーには、CICS エージェント（CICS トランザクション）の EXEC インターフェース ブロックがあります。

GWA ツリーには、CICS エージェントのグローバル作業域（GWA）があります。この領域は、CICS エージェントによって使用されるさまざまな CICS 出口との通信に使用されます。

全般ツリーには、エージェント バージョン、トランザクション ID などがあります。

[プログラム]タブ

［プログラム］タブには、CICS エージェントに展開されるすべてのトランザクション（CICS LINK、DTP など）が表示されます。

例には、2 つのプログラムがあります。

1. PROGRAM1 は CICS LINK のレコーディング モードです。
2. PROGRAM2 は DTP （分散トランザクション処理）コマンドの再生モードです。

これらのプログラムのいずれかがエージェント内で「孤立」している場合、そのプログラムのチェック ボックスをオフにして［プログラムのリセット］ボタンをクリックすると、そのプログラムを削除できます。 このアクションは、そのプログラムを CICS エージェントから削除します。

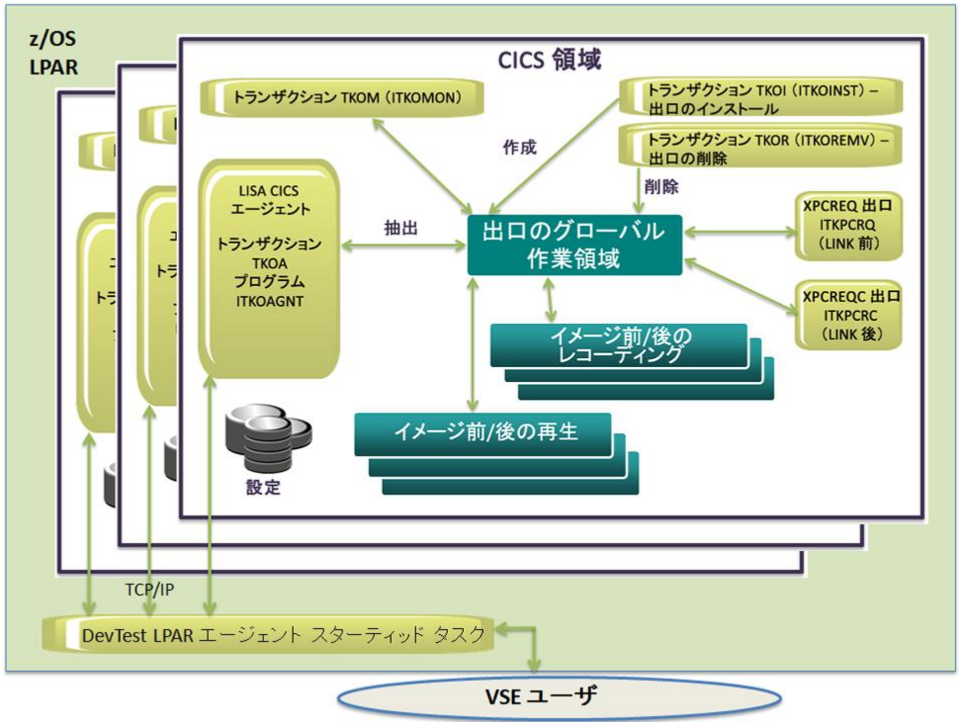
第 3 章: DevTest CICS エージェント

DevTest CICS エージェントは、VSE をサポートするための機能を提供する z/OS CICS 常駐エージェントです。 このエージェントは、実行時間の長いタスク TKOA と CICS 出口の XPCREQ、XPCREQC、XEIIN、XEIOUT の組み合わせとして実装されます。 このエージェントは、CICS PLT を使用して自動的に、またはインストールするための TKOI トランザクションと削除するための TKOR トランザクションを使用して手動でアクティブ化できます。必要な唯一の設定は、DevTest LPAR エージェントの場所です。

このセクションには、以下のトピックが含まれています。

CICS エージェントの概要

以下の図は、DevTest LPAR エージェントと DevTest CICS エージェントのコンポーネントを示しています。



CICS エージェントをインストールする方法

DevTest CICS エージェントのパッケージは、以下のファイルで構成されています。

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

インストールの手順は以下のとおりです。

1. ターゲットの z/OS システムにファイルを FTP 送信します（P. 174）。
2. 区分データ セット（PDS）をリストアします（P. 113）。
3. TCP/IP インターフェースがインストールされ、CICS で起動されていることを確認します（P. 113）。
4. CICS エージェント設定ファイルを作成します（P. 114）。
5. CICS に対する CICS エージェント リソースを定義します（P. 114）。
6. CICS PLT に出口の有効化/無効化を追加します（P. 116）。
7. CICS 起動 JCL を変更します（P. 117）。

ターゲットの z/OS システムへのファイルの FTP 送信

使用している CICS バージョンに一致するロード ライブラリを使用します。

- CICS バージョン 3.2 の場合は、iTKOCICSAgentLoadCICS32 を使用します。
- CICS バージョン 4.1 の場合は、iTKOCICSAgentLoadCICS41 を使用します。
- CICS バージョン 4.2 の場合は、iTKOCICSAgentLoadCICS42 を使用します。
- CICS バージョン 5.1 の場合は、iTKOCICSAgentLoadCICS51 を使用します。
- CICS バージョン 5.2 の場合は、iTKOCICSAgentLoadCICS52 を使用します。

使用している CICS バージョンを確認する簡単な方法は、以下のトランザクションを実行し、CICSTslevel の値を確認することです。CICSTslevel には、バージョン/リリース/メンテナンスのレベルが vvrrmm: CECI INQUIRE SYSTEM CICSTslevel の形式で入ります。

iTKOCICSAgentLoadCICSvvv および iTKOCICSAgentCntl ファイルは、以下の例に示す z/OS 属性と共に、バイナリ モードでターゲットの z/OS システムに FTP 送信されます。

以下のコマンドライン FTP の例は、配布ファイル データ セット **iTKOCICSAgentCntl** および **iTKOCICSAgentLoadCICS42** の z/OS データ セット ITKO.CICSAGNT.CNTL.UNLOAD および ITKO.CICSAGNT.LOAD.UNLOAD への正しい転送を示しています。 z/OS のファイル名は、データ セットの命名規則に合わせて変更できます。

```
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.
```

区分データ セット（PDS）のリストア

CICS エージェント ロード ライブラリ PDS および制御 （JCL） ライブラリは、通常はバッチ ジョブで実行される TSO RECEIVE コマンドでリストアされます。

以下のサンプル JCL は、ボリューム VOL001 上の FTP 送信されたファイル データ セット **ITKO.CICSAGNT.LOAD** および **ITKO.CICSAGNT.CNTL** から PDS をリストアします。 データ セット名は、サイトの命名規則に合わせて変更できます。

```
//job
//*
//* LOAD ライブラリを TSO RECEIVE でアンロードします
//*
//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)

/*
```

TCP/IP インターフェースがインストールされていることの確認

CICS エージェントは、CICS TCP/IP インターフェースがインストールされ、起動されていることを必要とします。 TCP/IP インターフェースがインストールされていることを確認するには、EZAO、INQUIRE、CICS の各トランザクションを使用します。

CICS エージェント設定ファイルの作成

CICS エージェントは、TCP/IP 経由で LPAR エージェントと通信します。CICS エージェント設定ファイルでは、LPAR エージェントの IP アドレスと TCP ポート番号が識別されます。 このファイルは、エージェントの起動時に一時的なデータ（TD）のキューとしてアクセスされます。このデータ セットには、DSORG=PS、RECFM=FB、LRECL=80 の各属性があります。

1 列目にアスタリスク（*）を含むファイル内の行は無視されます。1 列目にアスタリスクを含まない最初のレコードには、LPAR エージェントの IP アドレスとポート番号が含まれます。 このレコードは、以下の形式である必要があります。

バイト 1 ～ 15	バイト 16	バイト 17 ～ 21	バイト 22 ～ 80
LPAR エージェントの IP アドレス	空白	LPAR エージェントのポート番号	未使用

制御 PDS のメンバ CONFIG には、以下の例が含まれています。

```
* ITKO CICS Agent Configuration
* Columns 1 - 15 contain LPAR Agent IP address (15 chars)
* Columns 17 - 21 contain LPAR Agent port number (5 chars)
192.168.0.100    2998
```

CICS に対する CICS エージェント リソースの定義

以下の表に、CICS に対して定義する必要のあるリソースのリストを示します。

名前	タイプ	説明
TKOC	TDQUEUE	設定の一時的なデータのキュー
TKOI	TRANSACTION	CICS 出口をインストールし、エージェント トランザクション（TKOA）を起動するためのトランザクション。 通常は、PLTPI の処理中に実行されます。
TKOR	TRANSACTION	CICS 出口を削除し、エージェント トランザクション（TKOA）を停止するためのトランザクション。 通常は、PLTSD の処理中に実行されます。

TKOA	TRANSACTIO N	エージェント トランザクション。 通常は、TKOI によって起動され、TKOR によって停止されますが、モニタ トランザクション（TKOM）によって起動および停止されることもあります。
TKOM	TRANSACTIO N	モニタ トランザクション。 出口やエージェントを手動で起動および停止したり、エージェントの処理をモニタしたりするために使用されます。
ITKOAGNT	PROGRAM	エージェント プログラム。 TKOA によって起動されます。
ITKOINST	PROGRAM	出口のインストール プログラム。 TKOI によって起動されます。
ITKOREMV	PROGRAM	出口の削除プログラム。 TKOR によって起動されます。
ITKPCRQ	PROGRAM	CICS XPCREQ（CICS LINK 出口の前）。 TKOI によって有効になり、TKOR によって無効になります。
ITKPCRC	PROGRAM	CICS XPCREQC 出口（CICS LINK 出口の後）。 TKOI によって有効になり、TKOR によって無効になります。
ITKOXEIN	PROGRAM	CICS XEIIN 出口（CICS コマンド出口の前）。TKOI によって有効になり、TKOR によって無効になります。
ITKOXEIO	PROGRAM	CICS XEIOUT 出口（CICS コマンド出口の後）。TKOI によって有効になり、TKOR によって無効になります。
ITKOTABN	PROGRAM	CICS XPCABND 出口（タスク異常終了出口）。TKOI によって有効になり、TKOR によって無効になります。
ITKOMON	PROGRAM	モニタ プログラム。

DFHCSDUP を使用してこれらのリソースをインストールするジョブの JCL は、DFHCSDUP という名前の制御 PDS 内にあります。

CICS PLT への出口の有効化/無効化の追加

TKOI トランザクションを使用して、CICS エージェントおよび出口を手動で起動できます。

TKOR トランザクションを使用して、CICS エージェントおよび出口を手動で起動できます。

CICS 出口を有効にしてエージェントを起動する簡単な方法は、PLTSI（CICS 起動）内にエントリを配置することです。PLTPI の例を以下に示します。SIP に PLTPI=SI が含まれている場合は、以下のテーブルが DFHPLTSI としてアセンブルされます。CICS TCP/IP インターフェースは、DevTest 出口およびエージェントの前に起動されます。

```
DFHPLT TYPE=INITIAL,SUFFIX=SI
*
* 以下のプログラムは PLTPI の 2 次パスで実行されます
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* 以下のプログラムは PLTPI の 3 次パスで実行されます
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOINST
         DFHPLT TYPE=FINAL
         END
```

DevTest 出口を無効にしてシャットダウン時にエージェントを停止する方法は、PLTSD（CICS シャットダウン）内にエントリを含めることです。PLTSD の例を以下に示します。 SIP に PLTSD=SD が含まれている場合は、以下のテーブルが DFHPLTSD としてアセンブルされます。 DevTest 出口およびエージェントは、CICS TCP/IP インターフェースの前に停止されます。

```
DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* 以下のプログラムは PLTSD の 1 次パスで実行されます
*
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOREMV
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* 以下のプログラムは PLTSD の 2 次パスで実行されます
*
         DFHPLT TYPE=FINAL
         END
```

CICS 起動 JCL の変更

既存の DFHRPL ロード ライブラリにコピーすることによって、CICS に DevTest CICS エージェント ロード モジュールを追加します。 あるいは、DFHRPL 連結に DevTest CICS エージェント ロード ライブラリを追加することもできます。

エージェント設定ファイルを定義するには、JCL も追加する必要があります（手順 5 (P. 114) から）。 データ セットの名前が ITKO.AGENT.CONFIG であり、TD キュー TKOC が DDNAME（ITKOACFG）で定義されている場合は、以下の JCL ステートメントを使用できます。

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

CICS エージェントのストレージに関する要件

DevTest CICS エージェントは、長い期間実行されるタスクです。 エージェントは、ESDSA 内の約 16K のプログラム ストレージおよび ECDSA 内の約 1K の動的ストレージを必要とします。

DevTest CICS 出口は、ERDSA 内の約 34K のプログラム ストレージおよび ECDSA 内の約 3K の動的ストレージ（トランザクションごとに）を必要とします。

VSE レコーダを使用する場合、CICS LINK の記録されたイメージは ECDSA 内に作成されます。

出口は記録されたイメージを作成し、CICS エージェントに渡します。 CICS エージェントは記録されたイメージを LPAR エージェントに送信し、解放します。したがって、その存在期間は短いものです。

記録されたイメージのサイズは、およそ 330 + 2 * （（COMMAREA 長、またはすべての CONTAINER 長の合計） + INPUTMSG 長）です。

記録されたイメージを収容するために必要な ECDSA の容量は、平均サイズ（前のパラグラフで計算）およびトランザクション レートによって異なります。

仮想サービスが展開されている場合、CICS LINK の再生イメージも ECDSA 内に作成されます。

仮想化された CICS LINK の開始時に、再生要求が出口で作成され、CICS エージェントに渡されます。 CICS エージェントは LPAR エージェントに再生要求を転送します。

CICS エージェントが再生応答を受信した場合、その応答は元の要求と照合されます。 要求と応答の両方が出口に渡され、出口で解放され、CICS LINK が完了します。

再生要求および応答のサイズは、およそ 300 + （COMMAREA 長、またはすべての CONTAINER 長の合計）です。

再生イメージを収容するために必要な ECDSA の容量は、平均サイズ（前のパラグラフで計算）およびトランザクション レートによって異なります。

CICS 出口の考慮事項

DevTest CICS エージェントは、以下の CICS 出口点を使用します。

- XPCREQ （CICS LINK 出口の前）
- XPCREQC （CICS LINK 出口の後）
- XEIIN （CICS コマンド出口の前）
- XEIOUT （CICS コマンド出口の後）
- XPCABND （トランザクション ABEND）

XPCREQ と XPCREQC は、CICS LINK の仮想化に使用されます。

XEIIN と XEIOUT は、CICS 分散トランザクション処理（DTP）の仮想化に使用されます。

XPCABND （およびその他の出口）は、CA Continuous Application Insight に使用されます。

XPCREQ と XPCREQC は必須です。 DTP の仮想化や CAI が必要ない場合は、XEIIN と XEIOUT を削除できます。 これらを削除するには、ロード ライブラリから ITKOXEIN と ITKOXEIO を削除します（または、それらの名前を変更します）。

CAI が必要ない場合は、XPCABND を削除できます。 これを削除するには、ロード ライブラリから ITKOTABN を削除します（または、その名前を変更します）。

ほかの出口との共存

XPCREQ、XPCREQC、XEIIN、XEIOUT、XPCABND タイプのいずれかの出口がすでにインストールされている場合、DevTest 出口はそれらと共存できます。

CICS は各出口を、それらが有効にされた順番で呼び出します。 PLT 内に ITKOINST エントリを配置することによって、その順番を制御できます。

DevTest CICS エージェント出口の動作の概要

- DevTest CICS エージェント出口には、ゼロ以外の（UEPCRCA によって処理された）以前のリターン コードを使用して入ることができます。この場合、出口は、（別の出口によって設定された）以前のリターン コードに設定されたリターン コード（R15）で戻ります。 DevTest 出口が、CICS が呼び出す最初の出口でない場合、DevTest 出口はその他の出口と共存できます。
- DevTest CICS LINK 出口（XPCREQ および XPCREQC）は、DevTest 制御ブロックのアドレスを渡すために CICS によってその出口に渡されたトークン（UEPPCTOK）を使用します。別の XPCREQ または XPCREQC がそのアドレスを壊している場合は、異常終了が発生するか、または DevTest XPCREQC 出口によって「**ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit**」という 1 回限りのメッセージが書き込まれる可能性があります。
- DevTest CICS エージェントの**前の** 出口（XPCREQ および XEIIN）が、それを仮想化するためのコマンドをバイパスする必要がある場合、R15 は、**バイパス**（UERCBYP）のリターン コードを返します。CICS が DevTest 出口の後に同じタイプの別の出口を呼び出した場合、その出口は、バイパスのリターン コードを元の CICS に伝達する必要があります。 そうしないと、そのコマンドはバイパスされず、仮想化は失敗します。
- （UEPRECUR によって処理された）**再帰**インジケータがゼロ以外である場合、DevTest XPCREQ および XPCREQC 出口は直ちに戻ります。このリターンにより、このタイプの別の出口が CICS LINK コマンドを使用している場合に発生する可能性のある問題が解消されます。
- XPCABND（トランザクション ABEND）出口は常に UEPCRCA が R15 で戻るため、ほかの XPCABND 出口と共存する必要があります。 このルールに対する唯一の例外は、**タスクのパージ**を返す XPI 関数のイベントです（以下を参照）。

- DevTest CICS エージェント出口のいずれかが CICS XPI（出口プログラミング インターフェース）から**タスクのパージ**のリターン コードを受信した場合、その出口は、**パージ**（UERCPURG）が R15 で戻り、（UEPCRCA によって処理された）以前のリターン コードを UERCPURG で上書きします。 CICS が DevTest 出口の後に同じタイプの別の出口を呼び出した場合、その出口は、パージのリターン コードを元の CICS に伝達する必要があります。

CICS エージェントのユーザ出口

ITKOUEX1 - DevTest CICS エージェント初期化出口

この出口は、DevTest CICS エージェントの初期化中に呼び出されます。 この出口を使用すると、以下のことが可能です。

- この CICS 領域がメンバになっているユーザ グループを示す
- この CICS 領域がメンバになっている CICSPlex を示す
- ストレージを割り当て、トークン パラメータを使用してそのアドレス/長さをほかの出口に渡す
- ほかのユーザ出口に必要なその他の初期化を実行する

この出口は CICS LINK を介して起動されるため、CICS がサポートする任意の言語で記述できます。 初期化中に、ITKOUEX1 への CICS LINK が試行されます。

この出口が必要でない場合は、以下のアクションのいずれかを実行します。

- DFHRPL 連結に ITKOUEX1 という名前のモジュールを含めないようにします。
- PPT で出口を無効にします。

パラメータは COMMAREA で渡されます。

以下の表では、それらのパラメータについて説明しています。

パラメータ	長さ(バイト単位)	入力または出力	データ タイプ	説明
トークン	8	入力/出力	不明	ユーザ出口で使用される可能性のある 8 バイト。 出口の呼び出しの後も保持されます。
グループ	8	出力	文字	出口によって設定されます。 この CICS がメンバになっているユーザ グループを識別します。
CICSPlex	8	出力	文字	出口によって設定されます。 この CICS がメンバになっている CICSPlex を識別します。
Sysid	4	入力	文字	CICS ASSIGN SYSID() からの CICS SYSID。
Applid	8	入力	文字	CICS ASSIGN APPLID() からの CICS APPLID。

Jobname	8	入力	文字	CICS INQUIRE SYSTEM JOBNAME() からの CICS ジョブ名。
LPAR	8	入力	文字	この CICS が存在する CVTSNAME からの LPAR 名。
Sysplex	8	入力	文字	この CICS が存在する ECVTSPLX からの SYSPLEX 名。

ITKO.CICSAGNT.CNTL （ITKOUEX1）にはサンプル COBOL ITKOUEX1 が含まれます。 このサンプルは、ユーザ グループを CICS ジョブ名の最初の 4 文字に設定します。

ITKO.CICSAGNT.CNTL （ITKOEX1C）には COBOL COMMAREA コピーブックが含まれます。

ITKO.CICSAGNT.CNTL （ITKOEX1A）にはアセンブラ COMMAREA コピーブック が含まれます。

ITKO.CICSAGNT.CNTL （COMPUEX1）には COBOL コンパイル JCL が含まれます。

ITKOUEX2 - DevTest CICS エージェント終了出口

この出口は、DevTest CICS エージェントの終了中に呼び出されます。 この出口は、ユーザ出口に必要な任意のクリーンアップ（ストレージを解放したり、ファイルを閉じたりなど）を実行するために使用できます。

この出口は CICS LINK を介して起動されるため、CICS がサポートする任意の言語で記述できます。エージェントの終了中に、ITKOUEX2 への CICS LINK が試行されます。 この出口が必要でない場合は、以下のアクションのいずれかを実行します。

- DFHRPL 連結に ITKOUEX2 という名前のモジュールを含めないようにします。
- PPT で出口を無効にします。

パラメータは COMMAREA で渡されます。

以下の表では、それらのパラメータについて説明しています。

パラメータ	長さ（バイト単位）	入力または出力	データ タイプ	説明
トークン	8	入力/出力	不明	ユーザ出口で使用される可能性のある 8 バイト。 出口の呼び出しの後も保持されます。

ITKO.CICSAGNT.CNTL （ITKOUEX2）にはサンプル COBOL ITKOUEX2 が含まれます。

ITKO.CICSAGNT.CNTL （ITKOEX2C）には COBOL COMMAREA コピーブックが含まれます。

ITKO.CICSAGNT.CNTL （ITKOEX2A）にはアセンブラ COMMAREA コピーブック が含まれます。

ITKO.CICSAGNT.CNTL （COMPUEX2）には COBOL コンパイル JCL が含まれます。

ITKOUEX3 - LISA CICS COMMAREA 統合出口

この出口は、COMMAREA の外部のストレージを参照するアドレスを含む COMMAREA を管理するために使用されます。 この出口は、1 つの連続した COMMAREA を作成するために、ストレージ フラグメントを COMMAREA と統合します。

この出口は、以下の時点でコールされます。

- COMMAREA の外部のデータを参照するアドレスを含まない置き換え用の COMMAREA を作成するために、CICS LINK のレコーディング中（前のイメージと後のイメージの両方について）。 この出口は、新しいストレージ領域に対して GETMAIN を実行する役割を果たします。 DevTest CICS エージェントは、この領域に対して FREEMAIN を実行します。 また、COMMAREA 内の場所を指す COMMAREA 内のアドレスにも問題があります。 この場合、この出口はレコーディング中にアドレスを相対的なオフセットに変換し、再生中に実際のアドレスに戻す必要があります。
- DevTest 応答から元の COMMAREA にリストアするために、CICS LINK の再生（仮想化）中。
- この出口が任意の追加の初期化タスクを実行できるようにするために、CICS エージェントの初期化中（ユーザ出口 1 の後）。
- この出口が任意の追加の終了タスクを実行できるようにするために、CICS エージェントの終了中（ユーザ出口 2 の前）。

この出口は CICS LINK を介して起動されるため、CICS がサポートする任意の言語で記述できます。 ただし、アドレス操作や可変長データの移動にはアセンブラ言語が最適です。 初期化中に、ITKOUEX3 への CICS LINK が試行されます。

この出口が不要な場合は、以下のいずれかのアクションを実行します。

- DFHRPL 連結に ITKOUEX3 という名前のモジュールを含めないようにします。
- PPT で出口を無効にします。

パラメータは COMMAREA で渡されます。

以下の表では、それらのパラメータについて説明しています。

パラメータ	長さ（バイト単位）	入力または出力	データ タイプ	説明
トークン	8	入力/出力	不明	ユーザ出口で使用される可能性のある 8 バイト。 出口の呼び出しの後も保持されます。
Call Type	1	入力	バイナリ	12 は再生を示します。 16 は終了を示します。
Image Type	1	入力	バイナリ	4 は前のイメージを示します。 8 は後のイメージを示します。
COMMAREA Address	4	入力	アドレス	アプリケーションの CICS LINK で渡される COMMAREA のアドレス。
COMMAREA Length	2	入力	バイナリ	アプリケーションの CICS LINK で渡される COMMAREA の長さ。
Calling Program	8	入力	文字	CICS LINK コマンドを発行するプログラムの名前。
Target Program	8	入力	文字	LINK でのリンク先のプログラムの名前。
New COMMAREA Address	4	入力/出力	アドレス	レコーディングの場合： 新しい統合された COMMAREA のアドレスが、この出口によってここに配置されます。この出口は通常、この領域を GETMAIN で取得します。 再生の場合：VSE 応答からの新しい統合された COMMAREA のアドレスがここに配置されます。この出口は、その内容をアプリケーションの COMMAREA にコピーします。

New COMMAREA Length	2	入力/出力	バイナリ	レコーディングの場合： 新しい統合された COMMAREA の長さが、この出口によってここに配置されます。 再生の場合：VSE 応答からの新しい統合された COMMAREA の長さが、この出口が使用するためにここに配置されます。
Return Code	1	出力	バイナリ	0 は COMMAREA が正常に統合されたことを示します。 DevTest CICS エージェントは、**New COMMAREA Address** と **New COMMAREA Length** を使用して記録されたイメージを作成します。 4 は、この出口が成功したことを示します。ただし、新しい COMMAREA は作成されていません。 DevTest CICS エージェントは、CICS LINK からの COMMAREA を使用して記録されたイメージを作成します。 8 は、この出口が失敗し、新しい COMMAREA が存在しないことを示します。 DevTest CICS エージェントは、以下のパラメータで示すように、EIBRCODE、EIBRESP、および EIBRESP2 で CICS LINK からアプリケーションに戻ります。 12 は、リターン コード 8 のすべての処理、およびこの出口を将来の DevTest CICS LINK レコーディングおよび再生から削除する必要があることを示します。
EIBRCODE	6	出力	バイナリ	**Return Code** が 8 または 12 である場合に使用される EIBRCODE 値。
EIBRESP	4	出力	バイナリ	**Return Code** が 8 または 12 である場合に使用される EIBRRESP 値。
EIBRESP2	4	出力	バイナリ	**Return Code** が 8 または 12 である場合に使用される EIBRRESP2 値。

アセンブラ言語サンプル ITKOUEX3 は、ITKO.CICSAGNT.CNTL(ITKOUEX3) にあります。

このサンプルは、Natural パラメータ CALLRPL=(ALL,3) または CALLRPL=(ALL,4) で COBOL をコールしたときに Software AG Natural 言語によって生成される形式で COMMAREA を管理します。COMMAREA のこの形式では、COMMAREA の長さは 12（CALLRPL=(ALL,3) が使用されている場合）および 16 （CALLRPL=(ALL,4) が使用されている場合）です。 COMMAREA 内の最初のフルワードはパラメータ リストのアドレスです。アドレス内で 1 に設定されている高位ビットは、最後のパラメータを示します。3 番目のフルワードはパラメータ長のリストのアドレスです。

ITKO.CICSAGNT.CNTL （ITKOEX3A）にはアセンブラ COMMAREA コピーブック が含まれます。

ITKO.CICSAGNT.CNTL （ASMBUEX3）にはアセンブリ JCL が含まれます。

CICS エージェントの操作

DevTest CICS エージェントおよび出口の起動

DevTest CICS エージェントおよび出口は、CICS 起動 PLT 処理中に起動されます。エージェントおよび出口は、TKOI トランザクションを実行するか、またはモニタ トランザクション TKOM を使用して手動で起動することもできます。 TKOI トランザクションは出口をインストールし、エージェント トランザクション（TKOA）を起動します。

DevTest CICS エージェントおよび出口の停止

DevTest CICS エージェントおよび出口は、CICS シャットダウン PLT 処理中に停止されます。 エージェントおよび出口は、TKOR トランザクションを実行するか、またはモニタ トランザクション TKOM を使用して手動で停止することもできます。

まれな状況として、CICS エージェント トランザクション（TKOA）が終了に失敗する場合があります。 この状況であると考えられる場合は、CEMT I TAS コマンドを使用して、TKOA がまだ実行されているかどうかを確認します。 実行されている場合は、TKOA を終了するために、必要に応じて CEMT SET TAS(nnn) PURGE または CEMT SET TAS(nnn) FORCEPURGE コマンドを使用します。

仮想化された CICS LINK コマンドへの影響

まれなケースとして、CICS LINK の仮想化が失敗する場合があります。 この場合、CICS LINK は「無効な要求」（INVREQ）状態（EIBRCODE=x'E00000000000' および EIBRESP=16）でアプリケーションに戻ります。 この状況では通常、アプリケーションに INVREQ 状態の処理が実装されていない限り、トランザクションの異常終了が発生します。 EIBRESP2 を使用すると、仮想化の失敗の正確な原因を特定できます。

500 / x'1F4'

> VSE から再生応答が受信されませんでした。これは、ほとんどの場合、タイムアウトが原因です。タイムアウト間隔は、10 秒に設定されています。 タイムアウト値を変更する必要がある場合は、テクニカル サポートに問い合わせてください。

501 / x'1F5'

DevTest CICS エージェントがダウンしています。 「CICS LINK の後」の出口が、DevTest CICS エージェントがダウンしていることを検出しました。 再生応答は返されません。

502 / x'1F6'

仮想化が失敗しました。サービス仮想化から再生応答が受信され、CICS LINK COMMAREA または CHANNEL/CONTAINER の更新が失敗しました。CICS コマンドの失敗メッセージについては、CICS ジョブのログを参照してください。

503 / x'1F7'

DevTest LPAR エージェントがシャットダウンされ、保留中の再生要求がすべてキャンセルされました。

仮想化された CICS DTP/MRO コマンドへの影響

まれなケースとして、CICS DTP/MRO コマンド（ALLOCATE、FREE、SEND、RECEIVE、CONVERSE、EXTRACT ATTRIBUTES）の仮想化が失敗する場合があります。 この場合、CICS コマンドは「無効な要求」（INVREQ）状態（EIBRCODE=x'E00000000000' および EIBRESP=500）でアプリケーションに戻ります。 この状況では通常、アプリケーションに INVREQ 状態の処理が組み込まれていない限り、トランザクションの異常終了が発生します。EIBRESP2 を使用すると、仮想化の失敗の正確な原因を特定できます。

501 / x'1F5'

DevTest CICS エージェントがダウンしています。 「CICS LINK の後」の出口が、DevTest CICS エージェントがダウンしていることを検出しました。 再生応答は返されません。

502 / x'1F6'

サービス仮想化から応答が受信されませんでした。

503 / x'1F7'

サービス仮想化から無効な応答が受信されました。CICS コマンドの失敗メッセージについては、CICS ジョブのログを参照してください。

504 / x'1F8'

セッション テーブルがいっぱいでした。

505 / x’1F9’

再生キューへの再生要求の追加に失敗しました。

506 / x’1FA’

サービス仮想化から応答が受信される前に、XEIIN WAIT タイムアウトが期限切れになりました。

507 / x’1FB’

要求または応答の GETMAIN が失敗しました。

モニタ トランザクション（TKOM）

モニタ トランザクション（TKOM）を使用すると、以下のことが可能です。

- エージェントおよび出口のステータスの確認
- グローバル作業領域（GWA）の 16 進数での表示
- 整形またはデコードされた GWA フィールドの一部の表示
- レコーディング数および再生数の確認
- プログラム テーブル数の確認
- プログラム テーブルの表示
- エージェントの起動および停止
- 出口の起動および停止
- 以下の 3 つの主なキューの現在の深度（数）とウォータマークの上限（最大）の確認

  レコーディング キュー（RecQ）

  - 記録されたイメージのキューを定義します。

  再生キュー（PlbQ）

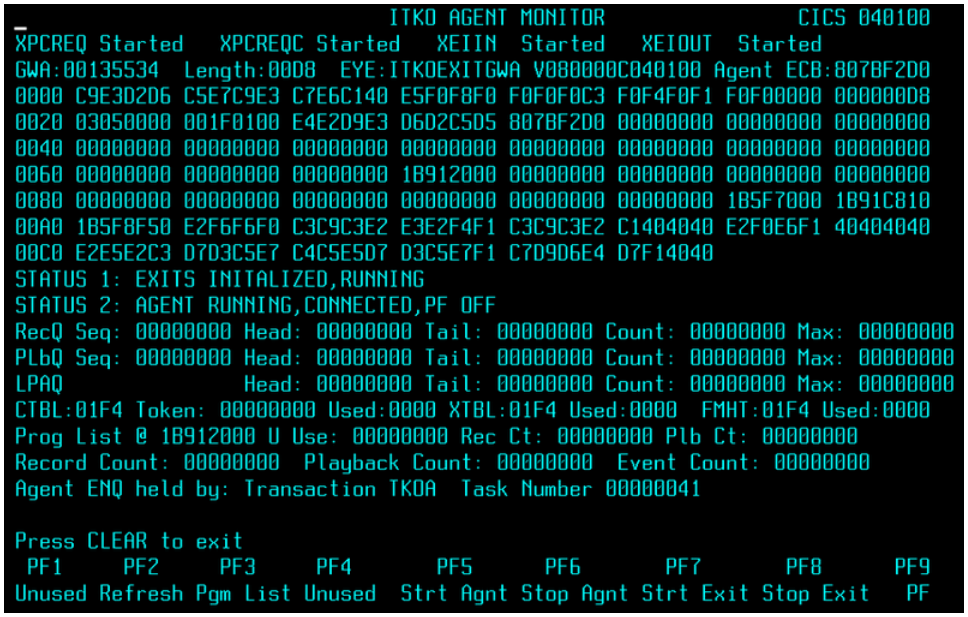
  - DevTest LPAR エージェントに送信される再生イメージのキューを定義します。

  LPAR エージェント キュー（LPAQ）

  - DevTest LPAR エージェントからの応答を待機している再生イメージのキューを定義します。

- エージェント トランザクションのタスク番号の確認

モニタ トランザクションは、以下のように表示されます。


ITKO AGENT MONITOR CICS 040100
XPCREQ Started XPCREQC Started XEIIN Started XEIOUT Started
GWA:00135534 Length:00D8 EYE:ITKOEXITGWA V080000C040100 Agent ECB:807BF2D0
0000 C9E3D2D6 C5E7C9E3 C7E6C140 E5F0F8F0 F0F0F0C3 F0F4F0F1 F0F00000 000000D8
0020 03050000 001F0100 E4E2D9E3 D6D2C5D5 807BF2D0 00000000 00000000 00000000
0040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0060 00000000 00000000 00000000 1B912000 00000000 00000000 00000000 00000000
0080 00000000 00000000 00000000 00000000 00000000 00000000 1B5F7000 1B91C810
00A0 1B5F8F50 E2F6F6F0 C3C9C3E2 E3E2F4F1 C3C9C3E2 C1404040 E2F0E6F1 40404040
00C0 E2E5E2C3 D7D3C5E7 C4C5E5D7 D3C5E7F1 C7D9D6E4 D7F14040
STATUS 1: EXITS INITALIZED,RUNNING
STATUS 2: AGENT RUNNING,CONNECTED,PF OFF
RecQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
PLbQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
LPAQ Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
CTBL:01F4 Token: 00000000 Used:0000 XTBL:01F4 Used:0000 FMHT:01F4 Used:0000
Prog List @ 1B912000 U Use: 00000000 Rec Ct: 00000000 Plb Ct: 00000000
Record Count: 00000000 Playback Count: 00000000 Event Count: 00000000
Agent ENQ held by: Transaction TKOA Task Number 00000041
Press CLEAR to exit
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9
Unused Refresh Pgm List Unused Strt Agnt Stop Agnt Strt Exit Stop Exit PF

CICS エージェントのメッセージ

メッセージは、DevTest CICS エージェントおよび出口によってオペレータコンソールと CICS ジョブのログに書き込まれます。

DevTest 出口インストーラ（トランザクション TKOI、プログラム ITKOINST）は、以下のメッセージを書き込みます。

- 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
- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

DevTest 出口アンインストーラ（トランザクション TKOR、プログラム ITKOREMV）は、以下のメッセージを書き込みます。

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

CICS XPCREQ （CICS LINK の前）出口（モジュール ITKPCRQ）または CICS XPCREQC（CICS LINK の後）出口（モジュール ITKPCRC）は、以下のメッセージを書き込みます。

- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx
- ITKO0010 – [Recorded Image | Playback Image | LPAR Agent] PLO failed in [ITKPCRQ | ITKPCRC] Exit
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx
- ITKO0012 - Image update failed with reason code 999

- ITKO0013 - PDU Parsing Error in ITKPCRC Exit
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

DevTest CICS エージェント（トランザクション TKOA、プログラム ITKOAGNT）は、以下のメッセージを書き込みます。

- ITKO0101 - Agent waiting for exits
- ITKO0102 - Agent initialization complete
- ITKO0103 - Agent shutting down
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent
- ITKO0105 - iTKO Agent started while another agent is running
- ITKO0106 - iTKO Agent connected to LPAR Agent
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999
- ITKO0109 - Socket func xxxxxxxx failed. RETCODE=xXXXXXXXX ERRNO=999999
- ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0111 - iTKO Agent disconnected from LPAR Agent
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent
- ITKO0114 - iTKO Agent error sending response to LPAR Agent
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address
- ITKO0117 - iTKO Agent released nnnnnn recorded images
- ITKO0118 - iTKO Agent canceled nnnnnn playback images
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images
- ITKO0120 - ITKO Agent released playback image: token xxxxxxxxxxxxxxxx Reason xxx
- ITKO0121 - iTKO Agent error reading configuration
- ITKO0122 - Agent TCP/IP API Timeout
- ITKO0123 - Agent waiting for TCP/IP API
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx

- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed
- ITKO0126 - Handshake error encountered. Shutting down Agent

CICS XEIIN （CICS コマンドの前）出口（モジュール ITKOXEIN）または CICS XEIOUT（CICS コマンドの後）出口（モジュール ITKOXEIO）は、以下のメッセージを書き込みます。

- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed
- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed
- ITKO205 - ITKOXEIN Task table full - discarding image
- ITKO206 - ITKOXEIN XPI WAIT Failed
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed
- ITKO208 - ITKOXEIO Add to recording queue failed
- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed
- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed

詳細:

DevTest CICS エージェントのメッセージの説明 (P. 135)

DevTest CICS エージェントのメッセージの説明

各メッセージの説明を表示するには、以下のリンクをクリックしてください。

ITKO0001

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

メッセージ タイプ

致命的なエラー

説明:

CICS コマンドが失敗し、ITKOINST は中止されました。 16 進数の EIBFN 値によって、失敗した CICS コマンドが識別されます。 失敗したコマンドは以下のとおりです。

EIBFN	**CICS コマンド**
x0C02	GETMAIN
x1008	START
x2202	ENABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

アクション

EIBRCDE、EIBRESP、および EIBRESP2 コードを使用して、失敗の原因を特定し、それを修正します。 いくつかの考えられる失敗の原因には、以下のものがあります。

1. TKOI トランザクションを実行しているユーザが、CICS システム プログラミング インターフェース（SPI）コマンド（EXTRACT EXIT など）を実行する権限を持っていない。
2. TKOA が定義されていないか、または誤って定義されていたため、エージェント トランザクション（TKOA）の START コマンドが失敗した。

ITKO0002

ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started

メッセージ タイプ

情報

説明:

示されている出口（XPCREQ または XPCREQC）が示されているアドレスにインストールされ、示されているアドレスのグローバル作業領域が示されている長さに設定されました。この出口は有効になり、起動されました。

通常、これらのメッセージは 2 つ（XPCREQ と XPCREQC のそれぞれに対して 1 つ）表示されます。 その両方の GWA のアドレスと長さは同じです。

アクション

なし

ITKO0003

ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started

メッセージ タイプ

致命的なエラー

説明:

示されている出口（XPCREQ または XPCREQC）が有効になりましたが、その出口を起動できませんでした。 ITKO のレコーディングおよび再生機能が正しく機能していません。

このメッセージには通常、ITKO0001 メッセージが付随します。

アクション

付随している ITKO0001 メッセージを診断し、問題を修正します。

TKOR を使用して出口を削除し、TKOI を使用して再インストールします。

ITKO0004

ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

メッセージ タイプ

致命的なエラー

説明:

ITKO 出口に小さすぎるグローバル作業領域が割り当てられました。

このメッセージには、ITKO0001 メッセージが付随する場合があります。

アクション

このエラーを ITKO サポートに報告してください。

ITKO0005

ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

メッセージ タイプ

致命的なエラー

説明:

CICS コマンドが失敗し、ITKOREMV は中止されました。 16 進数の EIBFN 値によって、失敗した CICS コマンドが識別されます。 失敗したコマンドは以下のとおりです。

EIBFN	CICS コマンド
x0C02	FREEMAIN
x1004	DELAY
x2204	DISABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

アクション

EIBRCDE、EIBRESP、および EIBRESP2 コードを使用して、失敗の原因を特定し、それを修正します。 考えられる失敗の原因の 1 つは、TKOR トランザクションを実行しているユーザが、CICS システム プログラミング インターフェース（SPI）コマンド（EXTRACT EXIT など）を実行する権限を持っていないことです。

ITKO0006

ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled

メッセージ タイプ

情報

説明:

示されている出口（XPCREQ または XPCREQC）は無効になりました。

アクション

なし

ITKO0007

ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit

メッセージ タイプ

致命的なエラー

説明:

この出口のためのグローバル作業領域が存在しないか、または小さすぎます。

アクション

このエラーを ITKO サポートに報告してください。

ITKO0008

ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

メッセージ タイプ

致命的なエラー

説明:

CICS コマンドが、示されている出口プログラムで失敗しました。 16 進数の EIBFN 値によって、失敗した CICS コマンドが識別されます。 失敗したコマンドは以下のとおりです。

EIBFN	CICS コマンド
0202	ADDRESS
0208	ASSIGN INVOKINGPROG
0C02	GETMAIN
0C02	FREEMAIN
3414	GET CONTAINER
3416	PUT CONTAINER
6C02	WRITE OPERATOR
9626	STARTBROWSE CONTAINER
9628	GETNEXT CONTAINER

アクション

EIBRCDE、EIBRESP、および EIBRESP2 コードを使用して、失敗の原因を特定し、それを修正します。考えられる失敗の原因の 1 つは、CICS のストレージ不足状態のために GETMAIN が失敗したことです。

ITKO0009

ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx

メッセージ タイプ

警告

説明:

示されているトランザクション、タスク、およびターミナルの呼び出し元プログラムから呼び出し先プログラムへの CICS LINK のレコーディング時に、記録されたイメージを処理するためのリソースを取得できませんでした。 付随している ITKO0008 メッセージでリソース（記録されたイメージに対する GETMAIN の失敗など）を識別できます。 DevTest LPAR エージェントが DevTest CICS エージェントから切断されていた可能性があります。

アクション

記録されたイメージがドロップされる原因になった状況を調査します。

ITKO0010

ITKO0010 - [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit

メッセージ タイプ

警告

説明:

PLO （z/OS Perform Locked Operation） コマンドが失敗しました。

PLO は、出口とエージェントの間の通信に使用される 3 つのキュー/リストの操作をシリアライズするために使用されます。 記録されたイメージのキューには、エージェントが処理する記録されたイメージが含まれています。 再生イメージのキューには、エージェントが処理する再生要求が含まれています。 LPAR エージェントのリストには、エージェントが出口に渡すために受信する再生応答が含まれています。

記録されたイメージのキュー エントリの場合、出口は記録されたイメージを破棄します。

再生イメージのキュー エントリの場合、イメージは破棄され、トランザクションは再生（仮想化）なしで続行されます。

LPAR エージェントのリスト エントリ（再生応答）の場合、LPAR エージェントのリスト上のイメージは出口によって切り離されます。

アクション

サポートにエラーをレポートします。

ITKO0011

ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx

メッセージ タイプ

エラー

説明:

出口 XPCREQ （モジュール ITKPCRQ）または出口 XPCREQC （モジュール ITKOPCRC）で、前または後のイメージの作成が示されている理由コードで失敗しました。

以下の理由コードは、情報の提供の目的でのみ示されています。

8 - 作成ルーチンにイメージが渡されませんでした。

12 - 作成ルーチンに前のイメージも後のイメージもありません。

16 - 作成ルーチンへの INPUTMSG の挿入で PDU の長さを超えました。

20 - 作成ルーチンへの COMMAREA の挿入で PDU の長さを超えました。

24 - 作成ルーチンへのコンテナの挿入で PDU の長さを超えました。

28 - 作成ルーチンへの元データの挿入で PDU の長さを超えました。

32 - 作成ルーチンで MVCL ロジック エラーが発生しました。

36 - 作成ルーチンに渡されたイメージ内に TKOCLNK がありません。

40 - 作成ルーチン内の CPDUVARN がゼロです。

44 - 作成ルーチンへの EIB の挿入で PDU の長さを超えました。

アクション

このエラーを ITKO サポートに報告してください。

ITKO0012

ITKO0012 - Image update failed with reason code 999

メッセージ タイプ

エラー

説明:

出口 XPCREQ （モジュール ITKPCRQ）または出口 XPCREQC （モジュール ITKOPCRC）で、後のイメージの更新が示されている理由コードで失敗しました。

以下の理由コードは、情報の提供の目的でのみ示されています。

4 - CICS コマンド エラーが発生しました。

5 - 更新ルーチンにイメージが渡されませんでした。

6 - 更新ルーチンに渡されたイメージ内に後の EIB がありません。

7 - 更新ルーチンに渡されたイメージ内に TKOCLNK がありません。

8 - 更新ルーチンに渡されたイメージ内で CVARTYP が CVARLINK ではありません。

9 - 更新ルーチンに渡されたイメージ内で CLNKEYE が 'CLNK' ではありません。

10 - 更新ルーチンに渡されたイメージ内で CVARLEN がゼロです。

11 - 更新ルーチンに渡されたイメージ内で CTNREYE が 'CTNR' ではありません。

アクション

サポートにエラーをレポートします。

ITKO0013

ITKO0013 - PDU Parsing Error in ITKPCRC Exit

メッセージ タイプ

エラー

説明:

出口 XPCREQC （モジュール ITKPCRC）で、ユーザ出口 3 で展開された COMMAREA の更新が失敗しました。

アクション

サポートにエラーをレポートします。

ITKO0014

ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

メッセージ タイプ

エラー

説明:

出口 XPCREQC （モジュール ITKPCRC）の後の CICS LINK で、正しい DevTest CICS エージェント制御ブロックをアドレス指定していないトークン（UEPPCTOK）が渡されました。 この原因はおそらく、XPCREQ 出口または XPCREQC 出口の競合です。 このメッセージは 1 回だけ書き込まれます。

アクション

CICS 領域にインストールされている XPCREQ 出口と XPCREQC 出口を調査します。

ITKO0101

ITKO0101 - AGENT waiting for exits

メッセージ タイプ

情報

説明:

出口が有効になる前に DevTest CICS エージェントが起動されました。このエージェントは、処理を続行する前に、出口が有効になるのを待機します。

アクション

なし。

ITKO0102

ITKO0102 - AGENT initialization complete

メッセージ タイプ

情報

説明:

DevTest CICS エージェントは初期化を完了し、DevTest LPAR エージェントへの接続を開始しています。

アクション

なし。

ITKO0103

ITKO0103 - AGENT shutting down

メッセージ タイプ

情報

説明:

DevTest CICS エージェントはシャットダウンしています。

アクション

なし。

ITKO0104

ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent

メッセージ タイプ

警告

説明:

PLO （z/OS Perform Locked Operation） コマンドが失敗しました。

PLO は、出口とエージェントの間の通信に使用される 3 つのキュー/リストの操作をシリアライズするために使用されます。 記録されたイメージのキューには、エージェントが処理する記録されたイメージが含まれています。 再生イメージのキューには、エージェントが処理する再生要求が含まれています。 LPAR エージェントのリストには、エージェントが出口に渡すために受信する再生応答が含まれています。

記録されたイメージと再生イメージのキュー エントリの場合、イメージはスキップされ、記録されたイメージおよび再生イメージ キューの次のパスで PLO が再試行されます。

LPAR エージェントのリスト エントリ （再生応答） の場合、イメージは破棄され、CICS LINK を発行しているユーザ トランザクションは INVREQ 応答（EIBRCODE=x'E00000000000'、EIBRESP=16、および EIBRESP2=500） を受信します。

アクション

サポートにエラーをレポートします。

ITKO0105

ITKO0105 - iTKO Agent started while another agent is running

メッセージ タイプ

警告

説明:

DevTest CICS エージェント（トランザクション TKOA）が、そのエージェントの別のインスタンスの実行中に起動されました。 CICS 領域ではエージェントを 1 つしか実行できないため、2 番目のエージェントは停止します。

アクション

なし。

ITKO0106

ITKO0106 - iTKO Agent connected to LPAR Agent

メッセージ タイプ

情報

説明:

DevTest CICS エージェントは、DevTest LPAR エージェントへの TCP/IP 接続を確立しました。

アクション

なし。

ITKO0107

ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999

メッセージ タイプ

情報

説明:

DevTest CICS エージェントは設定（TD キュー TKOC）を読み取り、示されている IP アドレスとポート番号を使用して DevTest LPAR エージェントに接続しています。

アクション

なし

ITKO0108

ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999

メッセージ タイプ

致命的なエラー

説明:

出口のグローバル作業領域（GWA）が存在しないか、または正しい長さではありません。 DevTest CICS Agent は停止します。

アクション

サポートにエラーをレポートします。

ITKO0109

ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xXXXXXXXX ERRNO=999999

メッセージ タイプ

警告

説明:

LPAR エージェントへの TCP/IP 接続でエラーが発生しました。

「**Socket func RECV failed. RETCODE=x00000000 ERRNO=000000**」というメッセージは通常、LPAR エージェントからの切断を示します。 このメッセージには、ITKO0111 メッセージが付随します。また、SEND 関数のエラーも LPAR エージェントの切断を示す場合があります。

ERRNO 値は、IBM 発行の「z/OS Communications Server IP CICS Sockets Guide」に記載されています。

アクション

このエラーが DevTest LPAR エージェントの切断またはその他の環境の問題のためであるかどうかを判断し、問題を修正します。 そうでない場合は、このエラーをサポートに報告してください。

ITKO0110

ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

メッセージ タイプ

致命的なエラー

説明:

CICS コマンドが DevTest CICS エージェントで失敗しました。 16 進数の EIBFN 値によって、失敗した CICS コマンドが識別されます。失敗したコマンドは以下のとおりです。

EIBFN	**CICS コマンド**
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

アクション

EIBRCDE、EIBRESP、および EIBRESP2 コードを使用して、失敗の原因を特定し、可能な場合はそれを修正します。 考えられる失敗の原因の 1 つは、TKOA トランザクションを実行しているユーザ ID が、CICS システム プログラミング インターフェース（SPI）コマンド（EXTRACT EXIT など）を実行する権限を持っていないことです。

ITKO0111

ITKO0111 - iTKO Agent disconnected from LPAR Agent

メッセージ タイプ

情報

説明:

DevTest CICS エージェントが DevTest LPAR エージェントから切断されました。

アクション

なし。

ITKO0112

ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent

メッセージ タイプ

エラー

説明:

DevTest CICS エージェントは記録されたイメージの送信中であり、DevTest LPAR エージェントとの TCP/IP 接続が失われました。 記録されたイメージはドロップされます。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0113

ITKO0113 - iTKO Agent error sending playback image to LPAR Agent

メッセージ タイプ

エラー

説明:

DevTest CICS エージェントは再生要求イメージの送信中であり、LPAR エージェントとの TCP/IP 接続が失われました。 記録されたイメージはドロップされ、進行中の CICS LINK が INVREQ 応答（EIBRCODE=x'E00000000000'、EIBRESP=16、および EIBRESP2=500）で完了します。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0114

ITKO0114 - iTKO Agent error sending response to LPAR Agent

メッセージ タイプ

エラー

説明:

DevTest CICS エージェントは DevTest LPAR エージェントへの応答の送信中であり、TCP/IP 接続が失われました。

アクション

CICS LPAR エージェントの切断の原因を調査します。

ITKO0115

ITKO0115 - iTKO Agent error receiving request from LPAR Agent

メッセージ タイプ

エラー

説明:

DevTest CICS エージェントは DevTest LPAR エージェントからの要求の受信中であり、TCP/IP 接続が失われました。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0116

ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address

メッセージ タイプ

致命的なエラー

説明:

DevTest CICS エージェント設定ファイル（TD キュー TKOC）内の DevTest LPAR エージェント IP アドレスは、有効なドット付き 10 進表記の IPv4 アドレスではありません。

アクション

設定されている IP アドレスを修正します。

ITKO0117

ITKO0117 - iTKO Agent released nnnnnn recorded images

メッセージ タイプ

警告

説明:

DevTest LPAR エージェントは、記録されたイメージのキューへの格納中に DevTest CICS エージェントから切断されました。 DevTest CICS エージェントは、保留中の記録されたイメージをキューから削除し、それらのイメージを解放します。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0118

ITKO0118 - iTKO Agent canceled nnnnnn playback images

メッセージ タイプ

警告

説明:

DevTest LPAR エージェントは、再生要求イメージのキューへの格納中に DevTest CICS エージェントから切断されました。 DevTest CICS エージェントは、保留中の再生要求イメージをキューから削除し、それらのイメージを解放します。 各再生要求イメージは、仮想化される CICS LINK を表します。 これらの CICS LINK コマンドは、INVREQ 応答（EIBRCODE=x'E00000000000'、EIBRESP=16、および EIBRESP2=500）で完了します。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0119

ITKO0119 - iTKO Agent canceled nnnnnn pending playback images

メッセージ タイプ

警告

説明:

DevTest LPAR エージェントは、保留中の再生応答イメージの待機中に DevTest CICS エージェントから切断されました。 DevTest CICS エージェントは、保留中の再生応答イメージをキューから削除し、それらのイメージを解放します。保留中の各再生応答イメージは、仮想化される CICS LINK を表します。 これらの CICS LINK コマンドは、INVREQ 応答（EIBRCODE=x'E00000000000'、EIBRESP=16、および EIBRESP2=500）で完了します。

アクション

DevTest LPAR エージェントの切断の原因を調査します。

ITKO0120

ITKO0120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxx

メッセージ タイプ

警告

説明:

DevTest CICS エージェントは、DevTest LPAR エージェントから、保留中の再生要求に一致しない再生応答を受信しました。 CICS LINK 仮想化で待機中の CICS トランザクションを識別できませんでした。 再生イメージは解放（破棄）されます。

以下の理由コードが有効です。コード 5 および 6 は、CICS エージェントの 30 秒のタイムアウト後に、仮想サービス環境から再生応答が受信されたことを示します。 このメッセージには通常、ITKO0124 メッセージが付随します。

1 - PDU が短すぎます。

2 - TKOCPDU の形式エラー。

3 - TKOCPDU トークンのプレフィックスが ITKO ではありません。

4 - TKOCPDU トークンのサフィックスが x'00000000' です。

5 - LPAR エージェント キューが空です。要求がタイムアウトした可能性があります。

6 - LPAR エージェント キュー上に見つかりません。要求がタイムアウトした可能性があります。

7 - CVAR の形式エラー。

8 - CICS コマンドのヘッダがありません。

9 - 後のイメージ EIB がありません。

10 - 要求 CPDU のエラー。

11 - LPAR エージェント キュー上の一致するイメージがすでにポストされています。

12 - CMALEYE が 'CMAL' ではありません。

13 - CMFREYE が 'CMFR' ではありません。

14 - CMSDEYE が 'CMSD' ではありません。

15 - CMRCEYE が 'CMRC' ではありません。

16 - CMCVEYE が 'CMCV' ではありません。

17 - CMEXEYE が 'CMEX' ではありません。

18 - Receive コマンドに受信されたデータ ブロックがありません。

19 - Converse コマンドに受信されたデータ ブロックがありません。

20 - Receive コマンドのデータ ブロックが短すぎます。

21 - Converse コマンドのデータ ブロックが短すぎます。

アクション

理由コード 5 および 6 の場合は、タイムアウトを調査します。その他の理由コードの場合は、このエラーをサポートに報告してください。

ITKO0121

ITKO0121 - iTKO Agent error reading configuration

メッセージ タイプ

致命的なエラー

説明:

TD キュー TKOC 設定の読み取り中にエラーが発生しました。

アクション

TD キュー TKOC の定義と内容を確認します。

ITKO0122

ITKO0122 - Agent TCP/IP API Timeout

メッセージ タイプ

致命的なエラー

説明:

DevTest CICS エージェントが CICS TCP/IP インターフェースの期限切れを待機し、DevTest CICS エージェントは停止しました。

アクション

CICS TCP/IP インターフェースが起動されたことを確認し、エージェントを再起動します。

ITKO0123

ITKO0123 - Agent waiting for TCP/IP API

メッセージ タイプ

警告

説明:

DevTest CICS エージェントは、CICS TCP/IP インターフェースの初期化を待機しています。 DevTest CICS エージェントは、10 秒間隔で 30 回（5 分間）再試行します。 待機がタイムアウトする前に CICS TCP/IP インターフェースが初期化されない場合、DevTest CICS エージェントは停止します。

アクション

対処する必要はありません。

ITKO0124

ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx

メッセージ タイプ

警告

説明:

再生イメージが仮想サービス環境に送信され、30 秒のタイムアウトの前に応答が受信されませんでした。 再生要求はパージされ、CICS コマンドが INVREQ 応答で終了します。

アクション

VSE にエラーがないかどうかチェックします。

ITKO0125

ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed

メッセージ タイプ

警告

説明:

ITKOUEX1、ITKOUEX2、または ITKOUEX3 への CICS LINK が失敗しました。 付随している ITKO0110 メッセージを参照してください。

アクション

ITKO0110 メッセージで識別された問題を修正します。

ITKO0126

ITKO0126 - Handshake error encountered. Shutting down Agent.

メッセージ タイプ

エラー

説明:

DevTest CICS エージェントと DevTest LPAR エージェントの間のハンドシェイクが失敗しました。 この失敗は通常、DevTest CICS エージェント設定内の DevTest LPAR エージェントの誤って設定されたポート番号によるものです。 このエラーにより、DevTest CICS エージェントが DevTest LPAR エージェントではないサーバに接続します。

アクション

DevTest CICS エージェント設定を修正します。

ITKO0200

ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）が、XPI INQ_APPLICATION_DATA コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

ITKO0201

ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）または XEIOUT 出口（モジュール ITKOXEIO）が、動的ストレージの割り当て中に XPI GETMAIN コマンドでエラーを受信しました。

アクション

CICS のストレージ不足状態を調査します。

ITKO0202

ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）または XEIOUT 出口（モジュール ITKOXEIO）が、動的ストレージの解放中に XPI FREEMAIN コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

ITKO0203

ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）または XEIOUT 出口（モジュール ITKOXEIO）が、記録されたイメージ用のストレージの割り当て中に XPI GETMAIN コマンドでエラーを受信しました。

アクション

CICS のストレージ不足状態を調査します。

ITKO0204

ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）または XEIOUT 出口（モジュール ITKOXEIO）が、イメージ ストレージの解放中に XPI FREEMAIN コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

ITKO0205

ITKOXEIN Task table full - discarding image

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）がタスク テーブルにエントリを割り当てようとしましたが、テーブルがいっぱいでした。

アクション

サポートにエラーをレポートします。

ITKO0206

ITKOXEIN XPI WAIT Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）が、再生応答の待機中に XPI WAIT コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

ITKO0207

ITKOXEIN XPI Freemain plbk req/rsp failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）が、再生要求または応答の解放中に XPI FREEMAIN コマンドでエラーを受信しました。

アクション

CICS のストレージ不足状態を調査します。

ITKO0208

ITKOXEIO Add to recording queue failed

メッセージ タイプ

エラー

説明:

XEIOUT 出口（モジュール ITKOXEIO）が記録されたイメージをレコーディング キューに追加しようとしましたが、追加の試行が失敗しました。

アクション

サポートにエラーをレポートします。

ITKO0209

ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）または XEIOUT 出口（モジュール ITKOXEIO）が、RECEIVE または CONVERSE コマンドの SET パラメータ用に割り当てられたストレージの解放中に XPI FREEMAIN コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

ITKO0210

ITKOXEIN XPI Getmain SET() Stg Failed

メッセージ タイプ

エラー

説明:

XEIIN 出口（モジュール ITKOXEIN）が、RECEIVE または CONVERSE コマンドの SET パラメータでのストレージの割り当て中に XPI GETMAIN コマンドでエラーを受信しました。

アクション

サポートにエラーをレポートします。

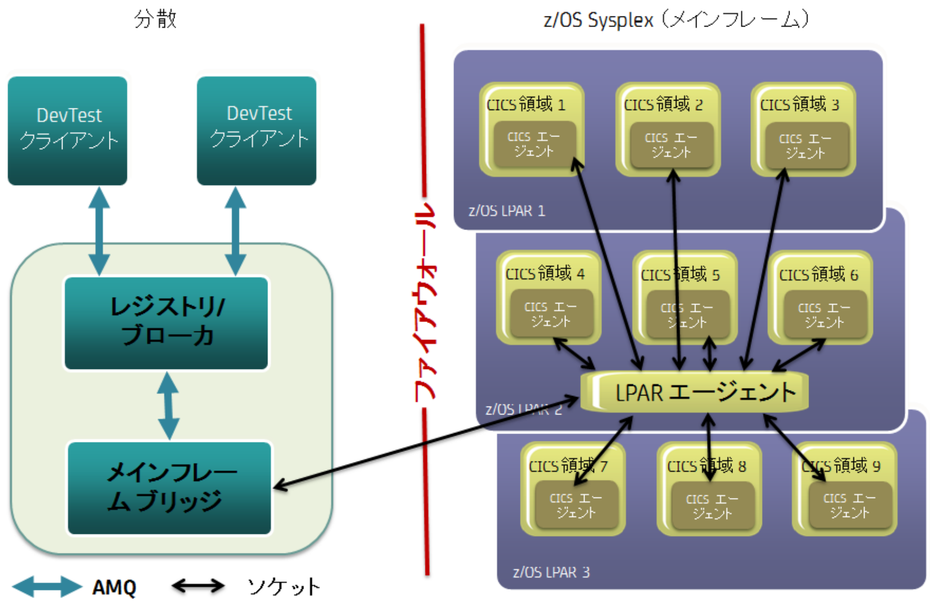
第 4 章: DevTest LPAR エージェント

DevTest LPAR エージェントは、スターティッド タスクとして実行される z/OS 常駐エージェントです。 このエージェントは、z/OS Sysplex の任意の LPAR にインストールできます。このエージェントでは、DevTest レジストリ（したがって DevTest クライアント）がその他の DevTest z/OS エージェントと通信するための単一の接点が提供されます。 このエージェントにより、DevTest は稼働中の z/OS CICS エージェントや、それらのエージェントに関するいくつかの基本的な設定情報を認識できるようになります。また、LPAR エージェントは、z/OS Sysplex に対して複数の DevTest LPAR エージェントが定義されている場合に DevTest z/OS CICS エージェントをグループ化するための手段も提供します。このエージェントは、クライアント モードで動作できます（その場合は、DevTest レジストリへの接続を確立します）。 このエージェントはまた、サーバ モードでも動作できます（その場合は、DevTest レジストリからの接続を受け入れます）。

このセクションには、以下のトピックが含まれています。

LPAR エージェントの概要

以下の図は、DevTest LPAR エージェントと DevTest CICS エージェントのコンポーネントを示しています。



LPAR エージェントをインストールする方法

LPAR エージェントのパッケージは、以下の 3 つのファイルで構成されています。

- iTKOLPARAgentDoc.pdf
- iTKOLPARAgentLoad
- iTKOLPARAgentCntl

インストールの手順は以下のとおりです。

1. ターゲットの z/OS システムにファイルを FTP 送信します（P. 174）。
2. 拡張区分データ セット（PDSE）と区分データ セット（PDS）をリストアします（P. 175）。
3. LPAR エージェントをスターティッド タスクとして実行されるようにセットアップします（P. 176）。
4. LPAR エージェント設定メンバを作成します（P. 178）。

ターゲットの z/OS システムへのファイルの FTP 送信

iTKOLPARAgentLoad および iTKOLPARAgentCntl ファイルは、以下の例に示す z/OS 属性と共に、バイナリ モードでターゲットの z/OS システムに FTP 送信されます。

以下のコマンドライン FTP の例は、データ セット ITKO.LPARAGNT.CNTL.UNLOAD および ITKO.LPARAGNT.LOAD.UNLOAD の z/OS への正しい転送を示しています。 z/OS のファイル名は、データ セットの命名規則に合わせて変更できます。

```
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.
```

PDSE と PDS のリストア

LPAR エージェント ロード ライブラリ PDSE および制御（JCL）ライブラリ PDS は、通常はバッチ ジョブで実行される TSO RECEIVE コマンドでリストアされます。

以下のサンプル JCL は、ボリューム VOL001 上の FTP 送信されたデータ セット ITKO.LPARAGNT.LOAD および ITKO.LPARAGNT.CNTL から拡張区分データ セット（PDSE）と区分データ セット（PDS）をリストアします。データ セット名は、サイトの命名規則に合わせて変更できます。

```
//job
//*
//* LOAD ライブラリを TSO RECEIVE でアンロードします
//*
//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)
/*
```

LPAR エージェントのセットアップ

ITKO.LPARAGNT.CNTL 内の LPARAGNT メンバは、LPAR エージェントをスターティッド タスクとして実行するプロシージャのモデルとして使用できます。 新しいプロシージャ メンバが作成された後で、タスクを起動できる PROCLIB にそれをコピーする必要があります。 このプロシージャは、以下の 4 つの JCL ステートメントで構成されます。

PROC

スターティッド タスクの名前を指定するステートメント。

EXEC

プログラム名とパラメータを指定するステートメント。

STEPLIB

LPAR エージェント ロード モジュールを含む PDSE ロード ライブラリ。

config

LPAR エージェント設定が定義されている PDS メンバを指定する DD ステートメント。

EXEC ステートメントで指定された PARM によって環境変数が定義されます。 PARM は、LPAR エージェント設定データ メンバを指すステートメントの DD 名も指定します。 PARM データはスラッシュ（/）で区切ります。スラッシュの前の値は実行時オプションを指定し、スラッシュの後の値は、アプリケーションへのコマンドライン引数を指定します。 環境変数は実行時オプションであり、LPAR エージェント設定データはコマンドライン引数です。

現在、LPAR エージェントが認識する環境変数はタイム ゾーンのみです。 この変数はオプションであり、指定されていない場合は、デフォルトで UTC になります。 この変数は、LPAR エージェントが生成するメッセージ内の日付と時刻の値を計算するために使用されます。 有効な値は、EST5EDT、CST6CDT、MST7MDT、PST8PDT などの標準の UNIX TZ 文字列です。

ITKO.LPARAGNT.CNTL の LPARAGNT メンバには、以下のサンプル プロシージャがあります。

```
//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=*
```

LPAR エージェント設定メンバの作成

DevTest LPAR エージェントに必要な設定データは、プレーンテキストでキーワードと値のペアとして指定されます。 LPAR エージェントを起動するために使用されるプロシージャの DD ステートメントで PDS メンバを指定します。 EXEC ステートメントの PARM 値で DD 名を指定します。 このファイルは以下のように解析されます。

- ハッシュ文字（#）は、コメント インジケータとして使用されます。
- 空白行や、ハッシュ文字で始まる行はすべて無視されます。
- キーワードと値は等号で区切ります。
- キーワードと値はどちらも、先頭と末尾の空白はすべて切り捨てられます。
- キーワードは大文字と小文字が区別されませんが、ほとんどの値は大文字と小文字が区別されます。
- LPAR エージェントによって理解される有効なキーワードのリストがあります。

LPAR エージェントを設定するために使用できるキーワードは以下のとおりです。

name

LPAR エージェントの名前を指定します。この名前は、ログ記録などに使用されます。

デフォルト： DevTest LPAR Agent

モード

LPAR エージェントのモード（**サーバ** モード（キーワード： Server）または**クライアント** モード（キーワード： Client））を指定します。サーバ モードでは、エージェントはポート *PortClient* 上の DevTest レジストリからの接続を受け入れます。 この値は、**lisa.properties** 内の **lisa.mainframe.bridge.mode** プロパティの反対である必要があります。

PortClient

サーバ モードで実行されている場合、クライアントが接続するウェルノウン ポートの値を定義します。このポート番号は、**lisa.properties** 内の **lisa.mainframe.bridge.server.port** プロパティが指定する値に一致している必要があります。

デフォルト： 3997

サーバ

クライアント モードで実行されている場合、接続対象のサーバの IP アドレスを指定します。この IP アドレスは通常、レジストリを実行している DevTest サーバ ホストの IP アドレスに一致します。

ServerPort

クライアント モードで実行されている場合、DevTest サーバ レジストリが使用するポート番号を定義します。 このポートは、**lisa.properties** 内の **lisa.mainframe.bridge.port** プロパティが指定する値に一致している必要があります。

PortAgent

CICS エージェントが接続するウェルノウン ポートの値を定義します。このポート番号は、この LPAR エージェントに接続しているすべての CICS エージェントで設定されている値に一致している必要があります。

デフォルト： 3998

InitialTrace

パケットのトレースを最初に有効化するかどうかを指定します。

値

- **True：** すべての接続でトレースは有効です
- **False：** すべての接続でトレースは無効です

Limits： これらの値は大文字と小文字を区別しません。

デフォルト： False

BufferSize

割り当てられる初期バッファ サイズを定義します。 この値は通常、予測される最大のデータ ペイロードのサイズの 2 倍に設定されます。この値を超えるサイズのパケットが受信された場合は、バッファ サイズを設定された最大まで再割り当てすることができます。

デフォルト： 65536

MaxBufferSize

許可されるパケットの最大サイズを定義します。 この値より大きいパケットは破棄され、メッセージがログに記録されます。 0（ゼロ）の値はサイズの検証を無効にし、クライアントからの任意のサイズのパケットが許可されます。

デフォルト： 0

サーバ モードで実行するための以下のサンプル設定ファイルは ITKO.LPARAGNT.CNTL の CONFIGSV メンバにあります。

```
#
# 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
#
```

クライアント モードで実行するための以下のサンプル設定ファイルが ITKO.LPARAGNT.CNTL の CONFIGCL メンバにあります。

```
#
# 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

#
```

LPAR エージェントのストレージに関する要件

DevTest LPAR エージェントは、最小で 3 MB の領域を使用するスタンドアロンのアドレス空間として動作します。 記録されたイメージが大き過ぎるか、または到着率が高い場合、大きな領域が必要になることがあります。

LPAR エージェントの操作

DevTest LPAR エージェントの起動

標準的な操作では、LPAR エージェントはスターティッド タスクとしてセットアップされ、LPAR IPL で起動されます。 標準の MVS START オペレータ コマンド（'S'）を使用すると、LPAR エージェントを明示的に起動できます。

DevTest LPAR エージェントの停止

LPAR エージェントは、標準の MVS STOP オペレータ コマンド（'P'）に応答します。 また、管理ソケット接続経由で「SHUTDOWN」メッセージを発行するか、または MVS MODIFY オペレータ コマンド（'F'）を使用してエージェントを停止することもできます。 LPAR エージェントは、どの手法でも同じように終了します。

LPAR エージェントのモニタおよび管理

LPAR エージェントは、標準の MVS オペレータ コマンドを使用して、ある程度までモニタおよび管理することができます。

標準の MVS オペレータ コマンドを使用するには、MODIFY コマンド（'F'）を使用して、LPAR エージェントに要求を発行します。 その応答は、コンソール ログに書き込まれます。サポートされている LPAR エージェント コマンドは以下のとおりです。

?

使用可能なコマンドのリストを表示します。

使用方法： F LPARAGNT,?

L

現在の接続とそれに関連付けられている接続 ID のリストを表示します。

使用方法： F LPARAGNT,L

T

パケットのトレースを切り替えます。

使用方法： F LPARAGNT,T <id> <state>

<id>

"L" コマンドで見つけた接続 ID を定義します。

<state>

（オプション）接続に対するパケットのトレースの状態を指定します。

状態を指定しないで T コマンドを発行すると、その接続に対するトレースの状態が返されます。

値： 「ON」および「OFF」

SHUTDOWN

LPAR エージェントをシャットダウンします。

使用方法： F LPARAGNT,SHUTDOWN

以下の出力は、**L** コマンドの例です。

```
F LPARAGNT,L

+ITKO9003 - Command accepted

+ID     Type IP                 Port          In        Out   Connected

+3     AT   192.168.0.100   6891         1        0  12/15/11 09:38:44

+4     A    192.168.0.100   6892         1        0  12/15/11 09:38:45
```

Type は、「エージェント接続」の場合は **A**、「クライアント接続」の場合は **C** です。 **A** または **C** の後に **T** が続いている場合は、その接続に対してパケットのトレースが有効になっていることを示します。 IP アドレスとポート番号によって、接続のソースが識別されます。In と Out はそれぞれ、ピアから受信したパケットとピアに送信したパケットの数を示します。Connected の日付と時刻は、接続が作成された日時を示します。Connected の日付と時刻では、プロシージャの EXEC ステートメントの PARM で指定されたタイム ゾーンの値が使用されます。

以下の出力は、T コマンドの例です。

```
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
```

LPAR エージェントのメッセージ

メッセージは、オペレータ コンソールと、スターティッド タスクの JES ログ内のデータ セットに書き込まれます。

オペレータ コンソールには、以下のメッセージが書き込まれる場合があります。 これらのメッセージは、情報としてのみ示されています。

- ITKO9001 - LPAR Agent process initialized and waiting for connections
- ITKO9002 - The Operator command was too large
- ITKO9003 - The command was accepted
- ITKO9004 - The Start command was accepted
- ITKO9005 - The Stop command was accepted
- ITKO9006 - LPAR Agent process terminating

サンプルの JES ログ（スターティッド タスクの DD SYS00001）を以下に示します。

```
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.
```

用語集

アサーション

*アサーション*は、1 つのステップとそのすべてのフィルタが実行された後に実行されるエレメントです。 アサーションにより、ステップの実行結果が予測と一致することが検証されます。 アサーションは、通常、テスト ケースまたは仮想サービス モデルのフローを変更するために使用されます。 グローバル アサーションは、テスト ケースまたは仮想サービス モデルの各ステップに適用されます。詳細については、「*CA Application Test の使用*」の「アサーション」を参照してください。

アセット

*アセット*は、1 つの論理的な単位にグループ化される設定プロパティのセットです。 詳細については、「*CA Application Test の使用*」の「アセット」を参照してください。

一致許容差

*一致許容差*は、CA Service Virtualization が受信要求をサービス イメージ内の要求と比較する方法を制御する設定です。 オプションは、EXACT、SIGNATURE、および OPERATION です。 詳細については、「*CA Service Virtualization の使用*」の「一致許容差」を参照してください。

イベント

*イベント*は、発生したアクションに関するメッセージです。 テスト ケースまたは仮想サービス モデル レベルでイベントを設定できます。 詳細については、「*CA Application Test の使用*」の「イベントについて」を参照してください。

会話ツリー

*会話ツリー*は、仮想サービス イメージにおいてステートフル トランザクションの会話パスを表すリンクされたノードのセットです。 各ノードは、withdrawMoney などの操作名でラベル付けされます。 getNewToken、getAccount、withdrawMoney、deleteToken は、金融機関システムの会話パスの一例です。 詳細については、「*CA Service Virtualization の使用」を参照してください。*

仮想サービス モデル（VSM）

*仮想サービス モデル*は、実際のサービス プロバイダなしでサービス要求を受信および応答します。 詳細については、「*CA Service Virtualization の使用*」の「仮想サービス モデル（VSM）」を参照してください。

監査ドキュメント

*監査ドキュメント*では、1 つのテスト、またはスイート内の 1 つのテストセットに対する成功条件を設定できます。 詳細については、「*CA Application Test の使用*」の「監査ドキュメントの作成」を参照してください。

クイック テスト

*クイック テスト*機能を使用すると、最小のセットアップでテスト ケースを実行できます。 詳細については、「*CA Application Test の使用*」の「クイック テストのステージング」を参照してください。

グループ

グループ、または仮想サービス グループは、VSE コンソールでまとめてモニタできるように、同じグループ タグでタグ付けされている仮想サービスのコレクションです。

継続的検証サービス（CVS）ダッシュボード

*継続的検証サービス（CVS）ダッシュボード*では、長期間にわたって定期的に実行するテスト ケースおよびテスト スイートをスケジュールできます。 詳細については、「*CA Application Test の使用*」の「継続的検証サービス（CVS）」を参照してください。

コーディネータ

*コーディネータ*はテスト ランの情報をドキュメントとして受け取り、1 つ以上のシミュレータ サーバで実行されるテストをコーディネートします。 詳細については、「*CA Application Test の使用*」の「コーディネータ サーバ」を参照してください。

コンパニオン

*コンパニオン*は、すべてのテスト ケースの実行の前後に実行されるエレメントです。 コンパニオンは、単一のテスト ステップではなく、テスト ケース全体に適用されるフィルタとして理解できます。 コンパニオンはテスト ケース内で（テスト ケースに対して）グローバルな動作を設定するために使用されます。 詳細については、「*CA Application Test の使用*」の「コンパニオン」を参照してください。

サービス イメージ（SI）

*サービス イメージ*は、CA Service Virtualization で記録されたトランザクションの正規化バージョンです。 各トランザクションは、ステートフル（会話型）またはステートレスです。 サービス イメージを作成する方法の 1 つは、仮想サービス イメージ レコーダを使用することです。 サービス イメージは、プロジェクトに格納されます。 サービス イメージは、*仮想サービス イメージ*（VSI）とも呼ばれます。詳細については、「*CA Service Virtualization の使用*」の「サービス イメージ」を参照してください。

サブプロセス

*サブプロセス*は、別のテスト ケースによってコールされるテスト ケースです。 詳細については、「*CA Application Test の使用*」の「サブプロセスの作成」を参照してください。

シミュレータ

*シミュレータ*は、コーディネータ サーバの管理下でテストを実行します。 詳細については、「*CA Application Test の使用*」の「シミュレータ サーバ」を参照してください。

ステージング ドキュメント

*ステージング ドキュメント*には、テスト ケースを実行する方法に関する情報が含まれます。 詳細については、「*CA Application Test* の使用」の「ステージング ドキュメントの作成」を参照してください。

設定

*設定*は、プロパティの名前付きのコレクションであり、通常はテスト中のシステムの環境に固有の値を指定します。 ハードコードされた環境データをなくすことにより、設定を変更するだけで、異なる環境内のテスト ケースまたは仮想サービス モデルを実行できます。 プロジェクトのデフォルト設定の名前は project.config です。 プロジェクトは多数の設定を持つことができますが、一度にアクティブになるのは 1 つの設定のみです。 詳細については、「*CA Application Test の使用*」の「設定」を参照してください。

対話型テスト ラン（ITR）

*対話型テスト ラン（ITR）*ユーティリティを使用すると、テスト ケースまたは仮想サービス モデルをステップごとに実行できます。 テスト ケースまたは仮想サービス モデルを実行時に変更し、結果を確認できます。 詳細については、「*CA Application Test の使用*」の「対話型テスト ラン（ITR）ユーティリティの使用」を参照してください。

ディセンシタイズ

*ディセンシタイズ*は、機密データをユーザ定義の代替データに変換するために使用されます。 クレジット カード番号や社会保障番号は機密データの例です。 詳細については、「*CA Service Virtualization の使用*」の「データのディセンシタイズ」を参照してください。

データ セット

*データ セット*は、実行時にテスト ケースまたは仮想サービス モデルにプロパティを設定するために使用できる値のコレクションです。 データ セットによって、テスト ケースまたは仮想サービス モデルに外部のテスト データを使用することができます。データ セットは、DevTest の内部または外部（たとえば、ファイルやデータベース テーブル）に作成できます。 詳細については、「*CA Application Test の使用*」の「データ セット」を参照してください。

データ プロトコル

*データ プロトコル*は、データ ハンドラとも呼ばれます。 CA Service Virtualization では、データ プロトコルは、要求の解析処理を行います。 一部のトランスポート プロトコルは、要求を作成するジョブの委任先のデータ プロトコルを許可（または要求）します。 結果として、プロトコルは要求ペイロードを認識する必要が生じます。 詳細については、「*CA Service Virtualization の使用*」の「データ プロトコルの使用」を参照してください。

テスト ケース

*テスト ケース*は、テスト中のシステムのビジネス コンポーネントをテストする方法の仕様です。 各テスト ケースには、1 つ以上のテスト ステップが含まれます。 詳細については、「*CA Application Test の使用*」の「テスト ケースの作成」を参照してください。

テスト スイート

*テスト スイート*は、順番に実行されるようにスケジュールされたテスト ケース、その他のテスト スイート、またはその両方のグループです。 スイート ドキュメントは、スイートのコンテンツ、生成するレポート、および収集するメトリックを指定します。詳細については、「*CA Application Test の使用*」の「テスト スイートの作成」を参照してください。

テスト ステップ

*テスト ステップ*は、実行される単一のテスト アクションを表すテスト ケース ワークフローのエレメントです。 テスト ステップの例としては、Web サービス、Java Bean、JDBC、JMS メッセージングなどがあります。テスト ステップには、フィルタ、アサーション、データ セットなどの DevTest エレメントを含めることができます。 詳細については、「*CA Application Test の使用*」の「テスト ステップの作成」を参照してください。

トランザクション フレーム

*トランザクション フレーム*は、DevTest Java エージェントまたは CAI Agent Light がインターセプトしたメソッド コールに関するデータをカプセル化します。詳細については、「*CA Continuous Application Insight の使用*」の「ビジネス トランザクションおよびトランザクション フレーム」を参照してください。

ナビゲーション許容差

*ナビゲーション許容差*は、CA Service Virtualization が会話ツリーを検索して次のトランザクションを見つける方法を制御する設定です。 オプションは、CLOSE、WIDE、および LOOSE です。 詳細については、「*CA Service Virtualization の使用*」の「ナビゲーション許容差」を参照してください。

ネットワーク グラフ

ネットワーク グラフは、DevTest クラウド マネージャおよび関連するラボをグラフで表示するサーバ コンソールの領域です。詳細については、「*CA Application Test の使用*」の「ラボの開始」を参照してください。

ノード

DevTest の内部では、テスト ステップは*ノード*とも呼ばれます。 これが、一部のイベントがイベント ID 内にノードを持つ理由です。

パス

*パス*には、Java エージェント がキャプチャしたトランザクションに関する情報が含まれます。詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

パス グラフ

*パス グラフ*には、パスおよびそのフレームのグラフ表示が含まれています。 詳細については、「*CA Continuous Application Insight の使用*」の「パス グラフ」を参照してください。

反応時間

*反応時間*は、テスト ステップを実行する前にテスト ケースが待機する時間です。詳細については、「*CA Application Test の使用*」の「テスト ステップの追加 - 例」および「ステージング ドキュメント エディタ - ［ベース］タブ」を参照してください。

フィルタ

*フィルタ*は、ステップの前後に実行されるエレメントです。フィルタは、結果のデータを処理、またはプロパティに値を格納する機会を提供します。グローバル フィルタは、テスト ケースまたは仮想サービス モデルの各ステップに適用されます。 詳細については、「*CA Application Test の使用*」の「フィルタ」を参照してください。

プロジェクト

*プロジェクト*は、関連する DevTest ファイルのコレクションです。 ファイルには、テスト ケース、スイート、仮想サービス モデル、サービス イメージ、設定、監査ドキュメント、ステージング ドキュメント、データ セット、モニタ、および MAR 情報ファイルなどが含まれます。 詳細については、「*CA Application Test の使用*」の「プロジェクト パネル」を参照してください。

プロパティ

*プロパティ*は、ランタイム変数として使用できるキー/値ペアです。 プロパティには、さまざまなタイプのデータを格納できます。 一般的なプロパティには、LISA_HOME、LISA_PROJ_ROOT、LISA_PROJ_NAME などがあります。 設定は、プロパティの名前付きのコレクションです。 詳細については、「*CA Application Test の使用*」の「プロパティ」を参照してください。

マジック ストリング

*マジック ストリング*は、サービス イメージの作成中に生成される文字列です。 マジック ストリングは、仮想サービス モデルによって応答内で意味のある文字列値が提供されることを確認するために使用されます。 {{=request_fname;/chris/}} は、マジック ストリングの一例です。 詳細については、「*CA Service Virtualization の使用*」の「マジック ストリングとマジック デート」を参照してください。

マジック デート

レコーディング中、日付パーサは要求および応答をスキャンします。 日付表示形式の広範な定義に一致する値は、マジック デートに変換されます。 マジック デートは、仮想サービス モデルによって応答内で意味のある日付値が提供されることを確認するために使用されます。 {{=doDateDeltaFromCurrent("yyyy-MM-dd","10");/*2012-08-14*/} は、マジック デートの一例です。詳細については、「*CA Service Virtualization の使用*」の「マジック ストリングとマジック デート」を参照してください。

メトリック

*メトリック*により、テストおよびテスト中のシステムのパフォーマンス/機能面に定量的手法および測定単位を適用できます。 詳細については、「*CA Application Test の使用*」の「メトリックの生成」を参照してください。

モデル アーカイブ（MAR）

*モデル アーカイブ（MAR）*は、DevTest Solutions における主要な展開アーティファクトです。 MAR ファイルには、プライマリ アセット、プライマリ アセットを実行するために必要なすべてのセカンダリ ファイル、情報ファイル、および監査ファイルが含まれます。 詳細については、「*CA Application Test の使用*」の「モデル アーカイブ（MAR）の操作」を参照してください。

モデル アーカイブ（MAR）情報

*モデル アーカイブ（MAR）情報ファイル*は、MAR を作成するために必要な情報が含まれるファイルです。詳細については、「*CA Application Test の使用*」の「モデル アーカイブ（MAR）の操作」を参照してください。

ラボ

*ラボ*は、1 つ以上のラボ メンバの論理コンテナです。 詳細については、「*CA Application Test の使用*」の「ラボとラボ メンバ」を参照してください。

レジストリ

*レジストリ*は、すべての DevTest サーバ および DevTest ワークステーション コンポーネントの登録を一元的に行うための場所です。 詳細については、「*CA Application Test* の使用」の「レジストリ」を参照してください。

仮想サービス環境（VSE）

*仮想サービス環境（VSE）*は、仮想サービス モデルを展開して実行するために使用する DevTest サーバ アプリケーションです。 VSE は CA Service Virtualization とも呼ばれます。 詳細については、「*CA Service Virtualization の使用*」を参照してください。