HP Connect-It

ソフトウェアバージョン：3.90

SDK

ドキュメントリリース日：15 May 2008
ソフトウェアリリース日：May 2008

法的制限事項

限定保証条項

機密コンピュータソフトウェア。

所有、使用、または複製するには、HP からの有効なライセンスが必要です。

FAR 12.211および12.212準拠。商用コンピュータソフトウェア、コンピュータソフトウェアマニュアル、技術データは、ベンダの標準商用ライセンスに基づき、米国政府にライセンス供与されています。

保証

HP製品およびサービスに対する保証は、当該製品およびサービスに付属の明示的保証規定に記載されているものに限られます。

本書のいかなる内容も当該保証に新たに保証を追加するものではありません。

HPは、本書中の技術的あるいは校正上の誤り、省略に対して責任を負いかねます。

ここに記載されている情報は、予告なしに変更されることがあります。

商標

- Adobe®, Adobe logo®, Acrobat® and Acrobat Logo® are trademarks of Adobe Systems Incorporated.
- Corel® and Corel logo® are trademarks or registered trademarks of Corel Corporation or Corel Corporation Limited.
- Java™ is a US trademark of Sun Microsystems, Inc.
- Microsoft®, Windows®, Windows NT®, Windows® XP, Windows Mobile® and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.
- Oracle® is a registered trademark of Oracle Corporation and/or its affiliates.
- UNIX® is a registered trademark of The Open Group.

目次

はじめに

Connect-It Development Kitを使用すると、ユーザ専用のコネクタを開発および導入できます。 この開発キットでは、J2EE Connector Architecture （1.5）規格に基づいてJavaインタフェースを使用します。 JCA規格は、一連のJavaインタフェースを定義してエンタープライズアプリケーション（ERP、データベースアプリケーションなど）の統合を簡略化します。

本書の対象ユーザ

本書は、JavaおよびJCA規格に十分な専門知識がある開発者向けです。 この規格の詳細については、以下のWebサイトを調べてください。J2EE Connector Architecture [http://java.sun.com/j2ee/connector]

用語

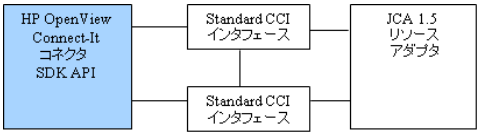
本書全体を通じて、以下の頭字語が使用されます。

- JCA：J2EE Connector Architecture
- RA：Resource Adapter
- EIS：Enterprise Information System
- CCI：Common Client Interface
- SPI：Service Provider Interface

- JDBC：Java Database Connectivity

一般情報

APIは、コネクタをアプリケーションに統合することが可能になるJCA 1.5 API への拡張を定義します。 以下の図に、この仕組みを示します。



コネクタとの通信モードは、接続先の情報システム（EIS）によって異なります。通信は以下に示す2種類があります。

- 送信の通信（同期）

  クライアントがデータ交換を開始します。 例えば、クエリがデータベースに送信されると, これが行われます。
- 受信の通信（非同期）

  EISがデータ交換を開始します。コネクタはリスニングモードです。 メッセージング用にこれが行われます。

SPI拡張

SDKは、メタデータの記述をサポートするSPIクラスへの拡張を提供します。

CCI拡張

SDKは、リレーショナルデータベースであるかどうかにかかわらず、システムへのアクセスを管理できるクライアントレイヤに提供します。 この拡張グループは、標準CCI APIおよびJDBC APIから起動します。

1 データ交換

SDKを使用して設計されたコネクタの目的は、データ交換を情報システムの基準に合わせることです。 データ交換には、データの送信と受信があります。

データおよびデータ型

データを交換するためには、データの構造体を理解する必要があります。 この構造体は、メタデータと呼ばれます。 SDKでは、データを使用して操作を行うために、データの構造体を理解する必要があります。 これは、以下の2つのインタフェースによって行われます。

com.hp.ov.cit.connector.cci.ObjectRecord - 特定のデータを表します。

および

com.hp.ov.cit.connector.cci.ObjectType - 一連の関連データに必要な構造体を表します。

送信または受信された各データを説明する必要があるため、*ObjectRecord*インスタンスはその*ObjectType*記述にリンクされます。

コネクタによって提供されたデータは、一般的に階層またはグラフにまとめられます。 それらのメタデータは、他のメタデータを包含する場合、「複雑」であると言われます。 「子」メタデータは、データのフィールドで構成されます。 メタデータは、他のメタデータを包含しない場合、「単純」であると言われます。 このメタデータは、フィールドで構成されていません。

*ObjectRecord*グラフは、以下の項目で構成されます。

- 単一の*ObjectRecord*ルートデータ項目
- 各*ObjectRecords*は、再帰的にフィールドをトラバースすることによってアクセスできます。

モデル

SDKは、*ObjectType*と*ObjectRecord*のペアで記述される2種類の異なるデータモデルを提供します。 これらのモデルは、*Class/Instance*モデルと*XMLSchema/XML*モデルです。 1つのコネクタにつき、可能なモデルは1つのみです。

クラス／インスタンスモデル

このモデルは、データ構造のオブジェクト表現です。 このモデルは、クラスとインスタンスのJava概念に基づきます。

クラス

クラスには名前があり、その名前空間を形成するパッケージに属します。 これは、クラスに関連付けられたフィールドで構成されます。

このモデル内では、クラスがメタデータを構成します。 これは、*ObjectType*インタフェースによってアクセス可能で、以下のメソッドがあります。

```
public String getName();
public String getNamespace();

public Class getObjectClass();
public boolean isSimple();
public Field getField(String fieldName);
public Field[] getFields();
```

*com.hp.ov.cit.connector.cci.Field*インタフェースによってアクセスするフィールドは、専用の情報とそのクラスに関連する情報で構成されます。 クラスには、以下の特徴があります。

- 変更可能です。
- デフォルト値を設定できます。
- 複数回出現することがあり、リスト内に表示される場合は、インデックスされているものとして表されます。
- 値の設定を必須にすることができます。

これは、以下のメソッドによって*Field*インタフェースにより実行されます。

```
public String getName();
public ObjectType getType();
```

```
public Object getDefault();
public boolean isIndexed();
public boolean isReadOnly();
public boolean isRequired();
```

インスタンス

インスタンスは、クラスに関連付けられ、そのフィールドの1つまたは複数に対する値を含みます。

このモデル内では、インスタンスがデータを構成します。 これは、*ObjectRecord* インタフェースによってアクセス可能で、以下のメソッドがあります。

```
public Object get(String fieldName);
public Object get(String fieldName, int fieldIndex);
public void set(String fieldName, Object value);
public void set(String fieldName, int fieldIndex, Object value);
public void remove(String fieldName);
public void remove(String fieldName, int fieldIndex);
```

単純型

以下の表は、SDKによってサポートされるJava単純型のリストを示します。

java.lang.Boolean
java.lang.Byte
java.lang.Short
java.lang.Integer
java.lang.Long
java.lang.Float
java.lang.Double
java.lang.String
java.util.Date
byte[]
char[]

例

以下のデータモデルについて考えます。

```
A
|- String
|- int
|- B
|- String
|- C*
|- boolean
```

ビジネスクラスは、以下のように表現されます。

```
public class A
{
private String stringField = "This is a string";
private int intField;
private B bField;
}

public class B
{
private String stringField;
private List<C> listOfCField;
}

public class C
{
private boolean booleanField;
}
```

型への操作が以下のように行われます。

```
ObjectType objectTypeA = ...;
Field field = objectTypeA.getField("stringField");
boolean isSimple = field.getType().isSimple(); // true
Object defaultValue = field.getDefault(); // "This is a string"
...
field = objectTypeA.getField("bField");
isSimple = field.isSimple(); // false
ObjectType objectTypeB = field.Type();
field = objectTypeB.getField("listOfCField");
boolean isIndexed = field.isIndexed(); //true;
```

データが以下の方法で保存されます。

```
ObjectRecord objectA = ...;
ObjectRecord objectB = ...;
ObjectRecord objectC = ...;

objectA.set("intField", 5);
objectA.set("bField", objectB);

java.util.List<C> list = new java.util.ArrayList<C>();
list.add(objectC);
objectB.set("listOfCField", list);
```

XMLスキーマ／XMLモデル

この表現モデルは、XMLデータを扱うシステムに適応します。 メタデータは、一連の独自XMLスキーマから形成されます。 このモデルにより、上記のインタフェースの使用が制限されます。 この場合、*ObjectType*インタフェースの唯一の関連メソッドは以下のようになります。

```
public String getName();
public String getNamespace();

public boolean isXSD();
public org.w3c.dom.ls.LSInput[] getXSD();
```

このモデルは、名前と名前空間によって識別されるメタデータが常に単純であることも仮定されます。 つまり、フィールドを含まずに、1つまたは複数のXMLスキーマのみを含みます。

データ自体は、*ObjectRecord*インタフェースで以下のメソッドによってアクセスできます。

```
public void readXML(org.w3c.dom.ls.LSInput input);
public void writeXML(org.w3c.dom.ls.LSOutput output);
```

これらのメソッドにより、XML表現をインポートまたはエキスポートできます。

- org.w3c.dom.ls.LSInput - XMLデータの入力ソースを表します。
- org.w3c.dom.ls.LSOutput - XMLデータの出力ソースを表します。

2 設計時

本節では、EISに接続してそのメタデータを検出するコネクタによって使用される要素を説明します。

DesignTimeFactoryインタフェース

*com.hp.ov.cit.connector.spi.designtime.DesignTimeFactory*インタフェースは、以下の作業に必要なすべての情報を集中化します。

- 接続を取得する
- EISとのデータ交換の構造を記述する

通信モード

メソッド

public boolean supportsOutbound()

および

public boolean supportsInbound()

は、EISで使用する通信モードを決定するために使用されます。 Connect-It内で、これら2つのモードは排他的です。 コネクタの実装は、一度に1つのモードのみをサポートします。

設計時接続

データ交換タイプは、通信モードにかかわらず記述する必要があります。 これを行うには、実際のものである場合もない場合も通信を使用します。 送信する通信については、この通信をデータ交換自体に使用する通信と別にすることもできます。 例えば、Webサービスの場合、FTP接続によってアクセスできるWSDLファイルを使用してメタデータを記述するのに対して、Webサービスはhttpプロトコルによって実行されます。

*DesignTimeFactory*クラスのAPIは、以下のメソッドを提供します。

- **public boolean requiresSeparateMetaDataConnection()**

  EISで2つの通信タイプを識別するかどうかを指定します。 このメソッドは、受信する通信には使用されません。

- **public javax.resource.cci.ConnectionSpec createMetaDataConnectionSpec()**

  このメソッドは、*ConnectionSpec*インタフェースのJavaBean実装を戻します。 オブジェクトには、設計時の段階で接続に使用する「ユーザ」と「パスワード」など、クライアント固有の情報が含まれます。 設計時接続と実行時接続を区別しない送信の通信に関係するメソッドを以下に示します。

- **public javax.resource.cci.ConnectionSpec createConnectionSpec()**

例えば、メタデータをhttp接続によってアクセスする場合は、URLを知っている必要があります。

```
package com.myeis;

import java.net.URL;
import javax.resource.cci.ConnectionSpec;

public class MyEISConnectionSpec implements ConnectionSpec
{
private URL url;

public URL getUrl()
{
return url;
}

public void setUrl(String url)
{
this.url = url;
}
}
```

接続情報を取得したら、メタデータを記述できます。

メタデータの取得

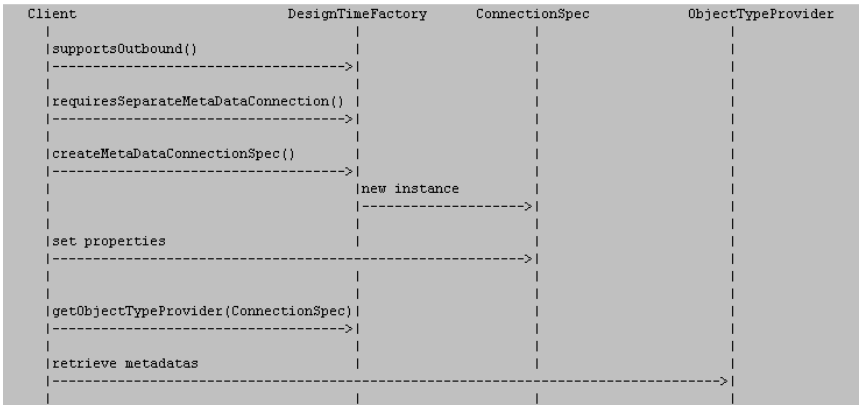
メソッド

public ObjectTypeProvider getObjectTypeProvider(javax.resource.cci.ConnectionSpec metaDataConnSpec)

は、EISと交換するデータの記述を取得するために使用するオブジェクトを戻します。 必要な接続情報は、そのパラメータです。

例

上記の操作を下図に示します。図は、メタデータにアクセスする特定の接続を必要とする送信の通信を示します。



コネクタで送信の通信をサポートするかどうかを確認するために、クライアントが**DesignTimeFactory**を問い合わせます。 サポートする場合は、メタデータへの接続がデータ交換に使用する接続と異なるかどうかを確認するために問い合わせます。 応答に従って、クライアントは、接続の記述を取得するために*createMetaDataConnectionSpec*メソッドまたは*createConnectionSpec*メソッドのいずれかを呼び出します。 クライアントは、メソッドのプロパティを設定し、*DesignTimeFactory*を呼び出して、メタデータ記述用の*ObjectTypeProvider*を取得します。

ObjectTypeProviderインタフェース

*com.hp.ov.cit.connector.spi.designtime.ObjectTypeProvider*インタフェースは、EISデータ型を記述するために使用します。 この記述は無限になる場合があります。 例えば、Aデータ型には、それ自体にAデータ型を含んでいるBデータ型を

含むことがあります。 再帰に関する問題を回避するためには、データ型を1ブロックに記述するのではなく、ナビゲーション可能なモデルをインタフェースの基盤にします。 これにより、第1レベルのデータを最初に検索することができます。 続いて、次の呼び出しがインタフェースに対して行われ、他のレベルのデータを記述できます。 これらの型は、以下のメソッドを呼び出して、接続によって取得されます。

public void close()

は、接続を閉じます。

以下のメソッドは、第1レベルのメタデータを記述するために使用します。

```
public java.util.List<ObjectType> getReceivedTypes();
public java.util.List<ObjectType> getRequestTypes();
public java.util.List<ObjectType> getResponseTypes();
```

EISタイプと通信モード（受信または送信）に応じて、これらのメソッドをサポートする必要性があるか、サポートしない必要性があります。 以下のように、サポート対象メソッドを実装します。

```
public java.util.List<ObjectType> getXXXTypes()
{
java.util.List<ObjectType> types = new java.util.ArrayList<ObjectType>();
types.add(new MyEISObjectType());
...
return types;
}
```

未サポートメソッドの場合：

```
public java.util.List<ObjectType> getXXXTypes() throws javax.resource.NotS
upportedException
{
throw new javax.resource.NotSupportedException();
}
```

受信の通信

このモードでは、以下のメソッドのみをサポートします。

```
public java.util.List<ObjectType> getReceivedTypes()
```

このメソッドは、EISから受信可能なイベントのリストを戻す必要があります。

送信の通信

2種類のデータ交換モードがサポートされます。

- 要求／応答（HTTP要求など）

- クエリ（SQL SELECTクエリなど）

クエリのデータ型は、以下によって取得されます。

```
public java.util.List<ObjectType> getRequestTypes()
```

このメソッドは、EISに送信可能なクエリタイプのリストを戻す必要があります。

getPurchaseOrder(int id)関数が"PurchaseOrder"オブジェクトを戻す場合など、クエリが応答を生成すると、期待された応答タイプを記述するために、以下のメソッドを使用する必要があります。

```
public java.util.List<ObjectType> getResponseTypes()
```

"getPurchaseOrders(PurchaseOrderType)" 関数が"PurchaseOrder"オブジェクトを戻す場合など、クエリが応答を引き起こすと、以下のメソッドが使用されます。

```
public java.util.List<ObjectType> getReceivedTypes()
```

データを含むクエリを送信するのではなく、要素をそれらのメタデータによって検索するために、EISへの問い合わせが行われます。

getResponseTypes()メソッドが単独でサポートされないことに注意してください。

ナビゲーション

第1レベルのタイプを取得した後、以下のメソッドが呼び出されて他のレベルのサブタイプを戻します。

```
public ObjectType getType(String namespace, String name)
```

呼び出し元によってパラメータとして送信された*namespace*、*name*の対の情報を使用し、記述されるレベルを知ることができます。 レベルが末端になると、メソッドから*null*を戻す必要があります。

3 実行時

本節では、EISに接続してデータを交換するコネクタによって使用される要素を説明します。

送信の通信

設定

この通信モードの主要クラスは、*javax.resource.spi.ManagedConnectionFactory*インタフェースを実装するクラスです。 このクラスは、*javax.resource.spi.ResourceAdapter*インタフェースも実装する必要があります。JCA仕様によって概略を記したように、このクラスはJavaBeanでなくてはなりません。 このJavaBeanオブジェクトのフィールドは、クライアントを問わず接続によって必要とされる情報を表します。 例えば、ODBC接続によってアクセスされるデータベースの場合、このデータベースの名前はクライアントを問わず必要とされます。

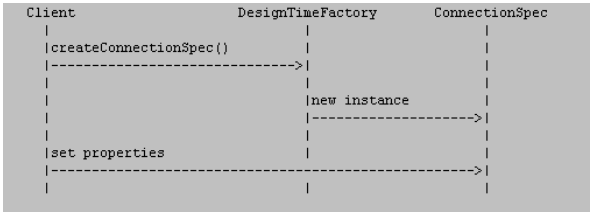
```
package com.mycompany.myeis;

import javax.resource.spi.ManagedConnectionFactory;
import javax.resource.spi.javax.resource.spi.ResourceAdapter

public class MyEISManagedConnectionFactory implements ManagedConnec
tionFactory, ResourceAdapter
```

```
{
private String dataSourceName;

public String getDataSourceName()
{
return dataSourceName;
}

public void setDataSourceName(String dataSourceName)
{
this.dataSourceName = dataSourceName;
}

...
}
```

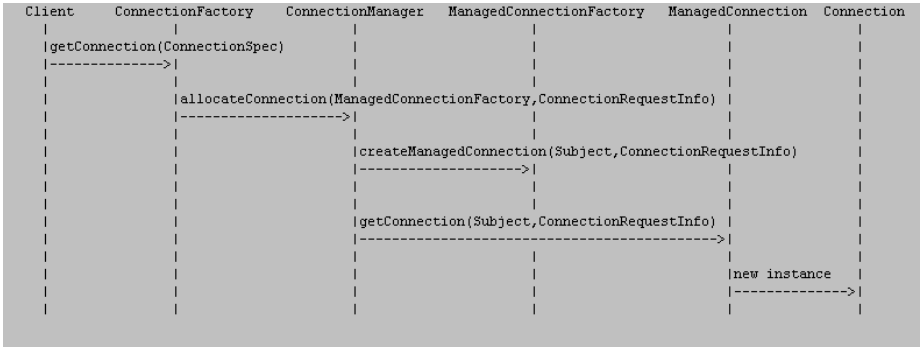
接続

SDKを使用して作成されたコネクタから取得されたクライアント接続は、JCA規格に準拠します。

接続情報を表す*javax.resource.cci.ConnectionSpec*オブジェクトは、最初に取得する必要があります。 これは、以下のスキーマに示すように実行されます。



*javax.resource.cci.ConnectionFactory*インタフェースによってクライアントがEISにアクセスし、入力された情報から接続を作成します。 例を簡潔化するために、一定の詳細が省略されています（接続プーリング、接続リスナー）。



すべての実装は、*com.hp.ov.cit.connector.cci.Connection*タイプの接続オブジェクトを戻します。

交換

接続が確立したら、クライアントアプリケーション（Connect-It）による外部システムとのデータ交換が可能になります。 この段階で、以下に示す設計時の情報に従って2つの交換モードが可能です。

- 応答がある要求または応答がない要求
- クエリ

要求／応答モード

EISとの交換のほとんどは、このカテゴリに分類できます。 例えば、レコードのリレーショナルデータベースへのインサートなどです。 この機能へのアクセスは、メソッドによって行われます。

public Interaction createInteraction()

以下の*com.hp.ov.cit.connector.cci.Interaction*インタフェースが使用されます。

```
public interface Interaction
{
...
public ObjectRecord execute(ObjectRecord request) throws ResourceExceptio
n;
}
}
```

データが入力として供給され、応答が戻る場合と戻らない場合があります。

クエリモード

期待されるデータのプロトタイプがクエリによってEISに送信されます。 類推によって、SQL SELECTクエリは、取得されるレコード内に期待される列を入力で指定します。

この機能へのアクセスは、メソッドによって行われます。

public Statement createStatement()

以下の*com.hp.ov.cit.connector.cci.Statement*インタフェースが使用されます。

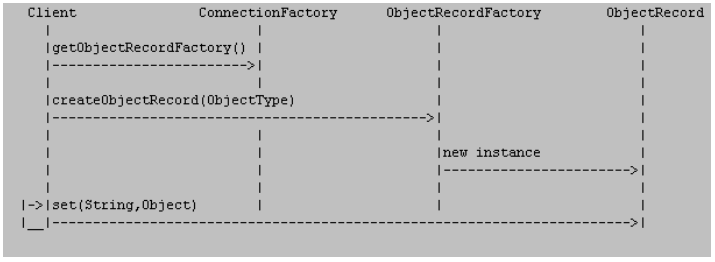
```
public interface Statement
{
...
public ObjectResultSet executeQuery(ObjectRecord prototype) throws Resour
ceException;
}
}
```

next()メソッドおよび**getObjectRecord()**メソッドは、データを取得するための結果セットを繰り返すために使用されます。

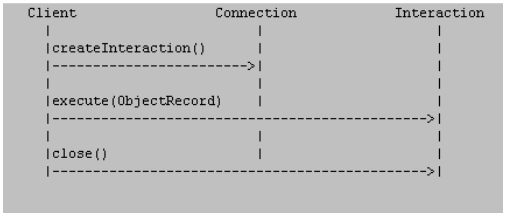
```
public interface ObjectResultSet
{
public boolean next();
public ObjectRecord getObjectRecord();
public void close() throws ResourceException;
}
```

スキーマ

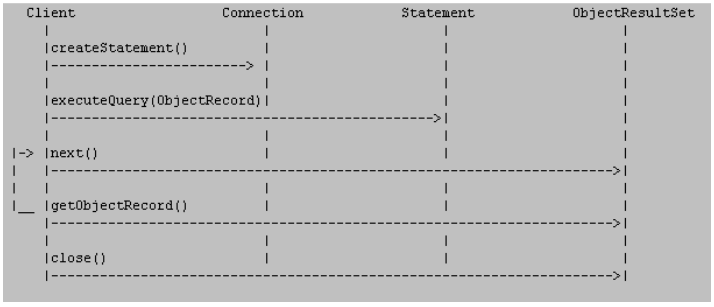
設計時メタデータのデータを作成します。



取得されたデータによってインタラクションを作成します。



取得されたデータプロトタイプからクエリを実行します。



受信の通信

設定

この通信モードの主要クラスは、*javax.resource.spi.ResourceAdapter*インタフェースを実装するクラスです。 JCA仕様によって概略が記されたように、このクラスはJavaBeanでなくてはなりません。 このJavaBeanオブジェクトのフィールドは、クライアントにかかわらず接続によって必要とされる情報を表します。

接続

*javax.resource.spi.ActivationSpec*インタフェースを実装するクラスは、クライアント接続を確立するために必要な情報を表します。
*javax.resource.spi.ResourceAdapter*クラスと同様に、これはJavaBeanオブジェクトでなくてはなりません。

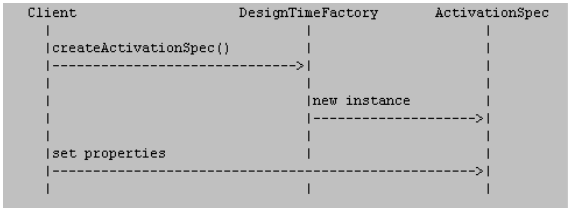
交換

EISは交換を起動します。 コネクタは、イベントリスナーとして機能します。 イベントを受信すると、コネクタは開始時にパラメータとして受け渡された*javax.resource.spi.endpoint.MessageEndPointFactory*オブジェクトによってクライアントに通知します。 これにより、以下のインタフェースの*ConnectionListener*オブジェクトを作成できます。

```
public interface ConnectionListener extends MessageListener
{
public void onException(Exception exception);
public ObjectRecord onRecord(ObjectRecord record);
}
```

スキーマ

接続情報を取得します（設計時）。



ResourceAdapterクラスのライフサイクル

4 配置

Connect-Itでコネクタを使用するには、最初に配置ファイルを作成する必要があります。 SDKでは、専用の配置記述子ファイルを使用し、JCA規格のra.xml記述子は使用しません。 このXMLファイルは、*Spring*フレームワークによって採用されたコンテキスト概念に基づきます。 名前をdesigntime-beans.xmlと付けて、コネクタのJARアーカイブのルートに保存する必要があります。

以下の情報が含まれます。

- *com.hp.ov.cit.connector.spi.designtime.DesignTimeFactory*クラスの完全な名前。
- *javax.resource.spi.ResourceAdapter*クラスの完全な名前。 送信する通信の場合、*javax.resource.spi.ManagedConnectionFactory*クラスが実装されます。

  以下に例を示します。

```
<beans>

<bean id="designTimeFactory" class="com.mycompany.myeis.MyEisDesignTimeFactory">
<property name="resourceAdapter">
<ref bean="resourceAdapter"/>
</property>
</bean>

<bean id="resourceAdapter" class="com.mycompany.myeis.MyEisManagedConnectionFactory"/>

</beans>
```

5 設定

コネクタを使用するために、一定数の設定ファイルが*Connect-It*によって必要になります。 この名前は、既存の*Connect-It*コネクタすべての中で一意である必要があります。 "Java"パッケージ命名規則に従うことをお勧めします。 この例では、コネクタに*com.mycompany.myeis*という名前を使用します。

記述ファイル

これは、コネクタの実際の記述に対する主要ファイルです。 一意の名前、後述するファイル名への参照、その有効化キーなど、コネクタに関連するすべてのプロパティを分類します。 このファイルの拡張子は、.dscにする必要があります。 ファイル名は、myeis.dscにすることをお勧めします。

例：

```
{CONNECTORDESC
InternalName=com.mycompany.myeis
ParentInternalName=Application_connectors
Name=My EIS
HTMLHelp=This is a description of my connector
Key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
IconFile=myeis.bmp
Sched.CanUsePointer=0
Cnx.HasCnx=1
```

```
Wizard.File=myeis-wizard.xml
Java.Class=com.hp.ov.cit.container.RAContainer
Java.Configuration.File=myeis-config.xml
Java.JVMConfiguration.File=myeis-jvmconf.xml
Java.HasOptions=1
}
```

アイコンファイル

コネクタナビゲーションツリーにアイコンを表示するために、6x16ビットマップを提供する必要があります。 このファイルには、myeis.bmpという名前を付けます。

設定ファイル

このファイルは、ユーザによる設定が必要な一連のJavaBeansプロパティで構成されます。 このファイルは、以下のコマンドラインによってエキスポートされたシナリオ設定に含まれるプロパティを指定するためにも使用されます。

```
conitsvc -export[:<property file>] <scenario>
```

*myeis-config.xml*ファイルの例：

```
<configuration>
<property name="ra_url" type="String" export="true">
<definition>
<default/>
</definition>
<export>
<description>URL</description>
</export>
</property>
<property name="cs_userName" type="String" export="true">
<definition>
<default/>
</definition>
<export>
<description>User</description>
</export>
</property>
</configuration>
```

ウィザードファイル

コネクタ用のXMLフォーマットウィザード定義ファイル。

*Connect-It*のコネクタ設定用ページを記述するために使用されます。 これは、接続定義ページで構成されます。 また、インタフェースコントロールが部品（テキスト、チェックボックス、ボタン）、ラベル、位置などの面で記述されます。

ユーザによる設定が必要なJavaBeansプロパティすべてがこのファイル内になくてはなりません。 以下の命名規則を使用する必要があります。

- 接頭辞*ra_*は、*javax.resource.spi.ResourceAdapter*インタフェースの実装に関連する各プロパティに追加する必要があります。
- 接頭辞*mdcs_*は、設計時（メタデータ）*javax.resource.cci.ConnectionSpec*インタフェースの実装に関連する各プロパティに追加する必要があります。
- 接頭辞*cs_*は、*javax.resource.cci.ConnectionSpec*インタフェースの実装に関連する各プロパティに追加する必要があります。
- 接頭辞*as_*は、*javax.resource.spi.ActivationSpec*インタフェースの実装に関連する各プロパティに追加する必要があります。

*myeis-wizard.xml*ファイルの例：

```
<wizard>

<page name="pgConnector">
<title>Connection</title>

<description>Configure connection to MyEIS</description>

<description>Enter the URL</description>
<control type="Textbox" name="ra_url">
<Value>$(GetValue[ra_url])</Value>
<label>URL</label>
<XOffset>2500</XOffset>
<labelLeft>1</labelLeft>
<Mandatory>1</Mandatory>
<MandatoryMsg>You must specify an URL value</MandatoryMsg>
<bind>Value</bind>
</control>

<description>Enter the user name</description>
<control type="Textbox" name="cs_userName">
<Value>$(GetValue[cs_userName])</Value>
<label>User</label>
<XOffset>2500</XOffset>
<labelLeft>1</labelLeft>
```

```
<bind>Value</bind>
</control>

<Transition>
<To script="true">{trConnector}</To>
</Transition>
</page>

</wizard>
```

JVM設定ファイル

JVMを開始するために、アプリケーションにクラスパス設定ファイルを提供する必要があります。

Connect-Itには、SDKを使用して作成されたコネクタであっても最小設定が必要です。 この設定は、*CONNECT-IT_HOME*/*config*/*shared*/*jca-container-jvmconf.xml*にあるファイルに記述されます。 これをユーザ専用のJVM設定ファイルに含む必要があります。

*myeis-jvmconf.xml*ファイルの例：

```
<jvmConfiguration id="com.mycompany.myeis">
<jarLocation>./com.mycompany.myeis</jarLocation>
<jars>
<jar groupId="com.mycompany.myeis" optional="false" provided="true"
version="1.00" versionNeeded="true">myeis</jar>
</jars>
<import>../shared/jca-container-jvmconf.xml</import>
</jvmConfiguration>
```

6 パッケージング

以下の方法で、コネクタをConnect-Itインストールにパッケージ化する必要があります。

```
Connect-It/
|
|- lib/
|   |- com.mycompany.myeis/
|      |- myeis-1.00.jar
|      |- myeis-3rdparty1.jar
|      |- myeis-3rdparty2.jar
|      |- ...
|
|- config/
|- com.mycompany.myeis/
|- myeis.bmp
|- myeis-jvmconf.xml
|- myeis.dsc
|- myeis-wizard.xml
|- myeis-config.xml
```

注意:

名前が一意であることを確認するには、コネクタの設定およびアーカイブディレクトリが"Java"パッケージ命名規則に従う必要があります。 上記の例の名前 *com.mycompany.myeis*は、これらの規則に従っています。

Javaアーカイブ

以下の構造をmyeis-1.00.jarアーカイブに使用する必要があります。

```
myeis-1.00.jar
|
|- designtime-beans.xml
|
|- com/
|   |- mycompany/
|      |- myeis/
|         |- MyEisDesignTimeFactory.class
|         |- MyEisManagedConnectionFactory.class
|         |- MyEisConnectionManager.class
|         |- ...
|
|- META-INF/
|   |- Manifest.mf
```

7 拡張機能

Interface com.hp.ov.cit.connector.spi.ContainerContext

コネクタがConnect-Itを経由してインスタンス化されると、コンテナの特定機能にアクセスできるよう、使用される特殊化された<javax.resource.spi.BootstrapContext>クラスが実装に与えられます。このコンテキストクラスにより、以下の機能が有効になります。

イベントリスナ

Connect-Itシナリオに関する実行イベントの通知を受信することが可能です。これには、以下のメソッドを使用するリスニングクラスを使用します。

```
public void addContainerListener(ContainerListener listener);
public void removeContainerListener(ContainerListener listener);
```

SDKでは、以下に挙げる2つのリスニングクラスタイプが導入されました。

- com.hp.ov.cit.connector.spi.ExecutionListener：シナリオの開始や停止の際の通知をリスンします。
- com.hp.ov.cit.connector.spi.SessionListener：実行中のシナリオに関する、セッション開始通知、およびセッション終了通知をリスンします。

シナリオパスへのアクセス

呼び出しを経由して実行するシナリオの完全パスを取得することができます。

```
public String getScenarioAbsolutePath();
```

これらの機能を使用する場合、<javax.resource.spi.ResourceAdapter>実装に以下のコードを入力する必要があります。

```
public void start(BootstrapContext bootstrapContext) throws ResourceAdapte
rInternalException
{
if (bootstrapContext instanceof ContainerContext)
{
//store this CIT context for use
}
else
{
//who is my container?
throw new ResourceAdapterInternalException();
}
}
```

com.hp.ov.cit.connector.spi.designtime.ObjectTypeProviderExクラス

このクラスは、<com.hp.ov.cit.connector.spi.designtime.ObjectTypeProvider>インタフェースの固有の実装です。これにより、サポート対象データ型に関する追加情報をコンテナに提供するのに、このインタフェースを使用する任意の実装が利用できるようになります。

<com.hp.ov.cit.connector.cci.ObjectType>インタフェースにより、単純データ（全体、ブーリアン値など）を含むJavaクラス型が提供されます。ただし、特に日付などの特定の型では、Javaクラスは型の意味を記述するのには不十分です（日付、日付/時刻、または時刻の違いなど）。このような特別なクラスでは、以下のメソッドにより考慮される単純型に関するコンテナ追加情報を与えることで、このような問題に対応します。

```
public String getXSDBuiltinDatatype(ObjectType simpleType)
```

このメソッドは、XML Schema [http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#built-in-datatypes] 仕様の「ビルトイン」データ型の名前を戻すことで、単純型の説明を入力します。

デフォルトでは、基本クラスは操作対象となっている単純型のどのような追加情報も入力しません。

通常の使用方法を以下に挙げます。

```
public class MyObjectTypeProvider extends ObjectTypeProviderEx
{
```

```
....

@Override
public String getXSDBuiltinDatatype(ObjectType simpleType)
{
if( simpleType instanceof MyDateObjectType)
{
return "date";
}
else if( simpleType instanceof MyDatetimeObjectType)
{
return "dateTime";
}
else if( simpleType instanceof MyTimeObjectType)
{
return "time";
}
else
{
//sorry ...no additionnal info along the Java class type
return null;
}
}
}
```

8 使用

SDKを使用して開発されたコネクタの実装は、以下の項目に関連付けられています。

- Connect-It認証証明書でのSDKアクセス宣言
- SDKを使用して作成されたコネクタに生成されたキー

認証証明書

認証証明書は、以下の項目を有効化します。

- SDKによって作成されたコネクタの使用を可能にするランタイム
- 新規作成コネクタのキー（ランタイムによって使用されるキー）の生成に使用するメニュー

▶『Connect-It - ユーザガイド』の「インストール」の章「認証証明書を入力する」

キーの生成

コネクタの使用を可能にするキー

キーを生成するには：

1 Connect-It Scenario Builderを起動します。

2 **Java/ Generate SDK activation key**を選択します。

3 表示されたウィンドウに、以下を入力します。
   - コネクタの名前
   - モード（生成、取り込み）
4 生成されたキーは、記述ファイルにコピーする必要があります。
   ▶ 『Connect-It ガイド - SDK』の「データベース記述ファイル [献 67]」の節

このキーは、コネクタの有効化および使用を可能にする認証証明書に関連付けられています。

I 付録

9 ウィザードファイル

本節では、コネクタの設定ウィザードのXMLファイルに使用するシンタックスに関する情報を説明します。

一般的な構造

ウィザードは、ページで構成されます。各ページには入力フィールド、ラベル、および説明を設定できます。 各ページは、次のページへの遷移を定義します。

```
<wizard>

<include/>
<property/>
<page>
<transition/>
</page>

</wizard>
```

ウィザード要素

ルート要素は、*wizard*でなくてはなりません。

以下のサブ要素があります。

要素	オプション	説明
include	はい	外部ファイルから定義をインクルードするために使用します。
property	はい	スクリプト化されたプロパティを定義するために使用します。
page	はい	ウィザードを構成するページを定義します。

要素のインクルード

ファイルをインクルードするために使用します。 シンタックスは、以下のようになります。

```
<include type="..." [basedir="..."]>the file name</include>
```

属性	オプション	説明
type	いいえ	インクルードのタイプを定義します。
basedir	はい	インクルードするファイルのディレクトリを定義します。
included	はい	要素を無視するか無視しないために使用します。

インクルードのタイプを以下に示します。

- string
- wizard

文字列インクルードタイプ

リソースファイルをインクルードするために使用します（ローカリゼーション文字列）。 デフォルトで、インクルードするファイルのパスは 現在のパスに相対的です。

このファイルに定義された文字列は、以下のシンタックスでアクセスします。
$(IDS_*NAME_OF_THE STRING*)

例：

myeisstrings.strファイルを調べます。

```
EIS_TITLE, "Title for the EIS"
EIS_DESCRIPTION, "Description of the EIS"
....
```

リソースは、文字列IDをインクルードすることによって、ウィザードファイル内で使用されます。

```
<wizard>
<include type="string">eisstrings.str</include>
<title>$(IDS_EIS_TITLE)</title>
...
</wizard>
```

注意:

- **リソースへのアクセスは、インクルード後に定義された要素にのみ有効です。**
- **インクルードは、ウィザードの生成時に実行されます。 その値をスクリプト化することはできません。**

ウィザードインクルードタイプ

別のウィザードファイルをインクルードするために使用されます。 このタイプのインクルードを指定できる要素は、*wizard*および *page*です。

パラメータは、インクルードされたウィザードに送信することができ、以下のシンタックスでアクセスできます。

```
$(GetValue[NAME_OF_THE PARAMETER])
```

例えば、値が*myValue*の*myParameter*をmyIncludedWizard.xmlウィザードに送信するには、以下のシンタックスが必要です。

```
<include type="wizard" myParameter="myValue>myIncludedWizard.xml</include>
```

ページ要素

ウィザードは、ページで構成されます。 以下の属性があります。

属性	オプション	説明
name	いいえ	ページの名前を定義します。 各ページの名前は一意です。

属性	オプション	説明
included	はい	要素を無視するか無視しないために使用します。

以下のサブ要素があります。

要素	オプション	説明
Transition	いいえ	次のページへの遷移を定義します。
Description	はい	ページ、ページセクション、またはコントロールに記述を追加するために使用します。
Title	はい	ページのタイトルを定義します。
property	はい	スクリプト化されたプロパティを定義するために使用します。
control	はい	ページにコントロールを定義します。
linebreak	はい	改行を定義します。
separator	はい	垂直セパレータを定義します。

```
<page name="..." included="...">


<title/>
<image/>
<description/>
<property/>
<control/>
<linebreak/>

<separator/>
<transition/>

</page>
```

注意:

コネクタのウィザードの第1ページは、pgConnectorという名前にする必要があります。

プロパティ要素

プロパティは、*string*、*long*など基本的な値タイプです。

属性	オプション	説明
name	いいえ	プロパティの名前を定義します。
included	はい	要素を無視するか無視しないために使用します。
script	はい	スクリプト化された内容を指定するために使用します。

例：

```
<page name="myPage">
<property name="IsVisible" type="Long" script="true">RetVal = 1</property
>
</page>

<property name="DelimString" script="true">RetVal = ""</property>
```

プロパティは、XMLツリー構造でプロパティの完全パス（ルートなし）を参照する*property full path*シンタックスによって使用されます。

例：

```
<visible script="true">{myPage.IsVisible} &lt;&gt; 1</visible>
<value script="true">{DelimString}</value>
```

コントロール要素

グラフィカルコントロールを定義するために使用します。 以下の属性があります。

属性	オプション	説明
name	いいえ	プロパティの名前を定義します。
type	いいえ	コントロールのタイプを定義します。 ページに対して一意でなくてはなりません。
included	はい	要素を無視するか無視しないために使用します。

属性	オプション	説明
script	はい	スクリプト化された内容を指定するために使用します。

コントロールタイプにかかわらず、以下のサブ要素があります。

要素	オプション	型	説明
visible	はい	ブーリアン	コントロールの表示有無を指定します。
enabled	はい	ブーリアン	コントロールをグレーアウトするかどうかを指定します。
readonly	はい	ブーリアン	コントロールを編集できるかどうかを指定します。
mandatory	はい	ブーリアン	コントロールに値が必要かどうかを指定します。
mandatorymsg	はい	文字列	必須の属性が存在してそれが1に等しい場合に値が入力されないと、エラーメッセージを指定します。
label	はい	string	コントロールの上にテキストを定義します。
labelleft	はい	ブーリアン	'1'または'true'の場合、ラベルを左側に配置します。
xoffset	はい	long	コントロールの左側にスペースを定義します。
bind	はい	string	関連ページが検証されたときに値が使用されたコントロール要素を指定します。 例： <bind>value</bind> 要素<value>の値を扱うために使用します。
property	はい	string	スクリプト化されたプロパティを定義するために使用します。

他のサブ要素が関連のコントロールタイプに応じて使用可能です。 主要なコントロールとそれらのサブ要素を以下に示します。

コントロールタイプ	サブ要素	型	説明
textbox	value	string	入力テキストの値
	multiline	long	0 = 単一行 それ以外の場合はコントロールサイズの百分率
	password	boolean	フィールドの暗号化の有無を指定する値。 1 = 暗号化されたフィールド
checkbox	value	boolean	コントロールのチェック有無を指定します。
	caption	string	コントロールラベル
combobox	value	string	選択した項目の値
	values	string	カンマで区切られた取り得る項目のリスト（ラベル=値）。 例： <values>English=en,French=fr</values>
numbox	value	long	コントロールの数値
	minvalue	long	最小値を指定します。
	maxvalue	long	最大値を指定します。
label	caption	string	コントロールラベル
fileedit	value	string	選択したファイルのパス
	openmode	long	編集タイプを定義します。 ■ 1 = OPEN ■ 2 = SAVE ■ 4 = OPEN_DIR ■ 8 = SAVE_DIR ■ 16 = APPEND
	filters	string	ファイルフィルタを定義します。 Example: <filters>XMLファイル (*.xml)|*.xml|XMLスキーマファイル (*.xsd)|*.xsd|</filters>
	defext	string	使用するデフォルト拡張子。 例： <defext>txt</defext>

コントロールタイプ	サブ要素	型	説明
	serializationId	string	ファイル選択コントロールのIDを定義します。 複数のコントロールで同じIDを使用できます。 このIDは、最後に選択されたファイルのパスを保存するために使用します。
optionbuttons	value	sting	選択した項目の値
	values	string	カンマで区切られた取り得る項目のリスト（ラベル=値） 例：<values>ISO-8859-1=0,UTF-8=1,Shift-JIS=2<values>
	border	boolean	コントロールにフレームがあるかどうかを指定します。

例：

```
<control type="TextBox" name="Server">
<value>$(GetValue[Server])</value>
<caption>$(IDS_SERVER_LABEL)</caption>
<xoffset>2500</xoffset>
<bind>value</bind>
</control>
```

bind属性

*bind*属性は、コントロールをコネクタの設定プロパティにリンクするために使用されます。 現在は、値*value*のみがSDKによってサポートされます。'cs_myprop'という名前のコントロールに指定された場合、コントロールの<value>要素の値は、'cs_myprop'設定プロパティの値としてコネクタに送信されます（コネクタの**ConnectionSpec**プロパティの'myprop'プロパティの値として）。

パスワード管理

パスワードなどの設定プロパティの管理には、ウィザードで特別な処理が必要です。 パスワードを含むプロパティが'cs_password'である場合、このプロパティのウィザードコントロール名が'clearcs_password'でなくてはなりません。

例：

```
<control type="TextBox" name="clearcs_password">
<value>$(GetValue[cs_password])</value>
<password>1</password>
<label>$(IDS_PASSWORD_LABEL)</label>
<xoffset>2500</xoffset>
<labelleft>1</labelleft>
<bind>value</bind>
</control>
```

改行要素とセパレータ要素

これらの要素は、ウィザードページをフォーマットするために使用されます。 以下の属性があります。

属性	オプション	説明
included	はい	要素を無視するか無視しないために使用します。

遷移要素

すべてのページには、遷移要素が必要です。 この要素は、次のページが何かを指定します。 以下の属性があります。

属性	オプション	説明
script	はい	スクリプト化された内容を指定するために使用します。

例：

```
<transition><to>nextPage</to></transition>

<transition>
<to script="true">
if( $(GetValue[ShowAdvancedWiz]) = 1 ) then
RetVal = "pgAdvanced"
else
RetVal = {trConnector}
end if
</to>
</transition>
```

注意:

コネクタのウィザードの最終ページにおける遷移は、スクリプト化された値*{trConnector}*と同等にする必要があります。

script属性

ウィザードは、Basicシンタックスによって記述された単純なスクリプトをサポートします。 これらのスクリプトは、ウィザード実行時に評価されます。

*script*属性は、値を含むすべての要素に使用可能です。 属性の値が*true*である場合に評価されるスクリプト化された式として、要素の値を指定するために使用されます。

例：

```
<... script="true">
if {Protocol.Value} = "ftp" or {Protocol.Value} = "http" then
RetVal = 1
else
RetVal = 0
end if
</...>
```

ウィザードで使用するBasicスクリプトでは、シンタックス{...}でウィザードコントロールまたはプロパティの値を参照します。 これらの値は、XMLツリー構造でプロパティの完全パス（ルートなし）を使用して参照されます。

included属性

この属性は、大部分の要素に使用できます。 これはオプションです。 当要素を無視するか無視しないかを指定するブール値が含まれます。

この属性が取り得る複数の値を以下に示します。

- *0*または*1*（または*0*以外の何か）
- *false*または*true*
- *and*、*or*、および*not*演算子を使用する式

この属性の値が*false*である場合、所属先の要素の内容が無視されます。

注意:

この属性の値は、ウィザード生成時に評価され、実行時には評価されません。そのため、要素のインクルードは、コントロールまたはその他のスクリプト化された式の値に応じて変えることができません。この属性の値は、通常、GetValue関数によって評価されます。

関数

以下に定義された関数は、Basicスクリプト関数ではありません。それらは、ウィザード生成時に評価され、実行時には評価されない関数です。

関数のフォーマット：

```
$(FunctionName[param1,param2<,optionalparam>,...])
```

GetValue関数

この関数は、ウィザードから値を動的に取得するために使用されます。この関数は、コネクタの設定プロパティの現在値を取得できるので、最も多く使用されるウィザード関数です。

シンタックスは、以下のようになります。

```
$(GetValue[name,default])
```

*name*パラメータは、検索する値の名前を指定します。*default*パラメータは、現在値が見つからない場合のデフォルト値を定義します。

以下の既存の値には、事前に定義された名前があります。

- OSUnix：プラットフォームがUNIXの場合は*1*を戻して、それ以外の場合は*0*を戻します。
- OSWindows：プラットフォームがWindowsの場合は*1*を戻して、それ以外の場合は*0*を戻します。
- WizardDir：インストールウィザードのディレクトリの完全パスを戻します（CONNECT-IT_HOME/config/wiz）。
- NameID：コネクタの名前を戻します。
- ShowAdvancedWiz：ウィザードがアドバンストモードの場合は*1*を戻して、それ以外の場合は*0*を戻します。
- ConfigDir：コネクタの設定ディレクトリの完全パスを戻します。

*GetValue*関数を呼び出すと、値の検索が以下の項目に対して行われます。

1. 記述ファイルに定義された特定の値
2. コネクタの設定プロパティ

3　事前に定義された値

例：

```
<value>$(GetValue[mylogin])</value>

<property name="trConnector" script="true">
if( $(GetValue[Cnx.HasCnx, 1]) = 1 then
RetVal = "pgConnection"
else
...
</property>

<control type="checkbox" name="UseWindowsRegistry" included="$(GetValu
e[OSWindows])">
<value>$(GetValue[UseWindowsRegistry])</value>
<caption>$(IDS_SERVER_LABEL)</caption>
<xoffset>2500</xoffset>
<bind>value</bind>
</control>
```

Dump関数

この関数を使用することによって、文字列をフォーマットしてスクリプト内で使用します。文字列は引用符で囲まれ、文字列内の引用符はエスケープされます。この関数は、**GetValue**関数を使用するか.strファイルの文字列によって文字列を取得するスクリプト内で大変便利です。 シンタックスは、次のようになります。

```
$(Dump[string])
```

例：

```
<value script="true">RetVal = $(Dump[$(GetValue[theValue])])</value>
```

EspaceCommas関数

この関数は、文字列内でカンマをエスケープするために使用します。関数は、文字列がカンマを文字セパレータとして使用する文字列のサブ要素である場合に使用できます（例：*optionbuttons*コントロールの値要素）。

```
$(EscapeCommas[string])
```

File関数

この関数は、ファイルの完全パスを取得するために使用します。 シンタックスは以下のようになります。

```
$(File[name,basedir])
```

*name*パラメータは、ファイルの名前を指定します。 *basedir*パラメータは、ファイルのディレクトリを定義します。 デフォルトのディレクトリは、ウィザードのディレクトリです。

例：

```
<image>$(File[myfile.bmp])</image>
```

10 設定ファイル

本節では、設定ファイルに使用するシンタックスに関する情報を説明します。
ファイルが以下の方法で構成されます。

```
configuration>

<property>
<definition>
<default/>
</definition>
<export>
<description/>
</export>
</class>
</property>

<property>
<definition>
<default/>
</definition>
<export>
<description/>
</export>
</class>
</property>
```

```
</configuration>
```

設定要素

ルート要素は、*configuration*でなくてはなりません。 ikano サブ要素があります。

要素	オプション	説明
property	はい	コネクタのJavaコードで必要な1つまたは複数のプロパティを定義します。

プロパティ要素

Java設定プロパティを指定します。
以下の属性があります。

属性	オプション	型	説明
name	いいえ	文字列	プロパティの名前を定義します。
type	いいえ	文字列	プロパティタイプを指定します。
export	はい	ブーリアン	エキスポート時にプロパティを使用する必要があるかどうかを指定します（-exportオプション）。

以下のサブ要素があります。

要素	オプション	説明
Definition	はい	プロパティ定義
export	はい	エキスポートの定義

要素	オプション	説明
class	はい	対応するJavaクラスの定義

定義要素

以下のサブ要素があります。

要素	オプション	型	説明
default	はい	文字列	ウィザードの初期化時に使用するデフォルト値を指定します。

エキスポート要素

以下のサブ要素があります。

要素	オプション	型	説明
説明	はい	文字列	プロパティのエキスポート時に使用する説明を指定します。 エキスポートされたプロパティファイル内でコメントとして表示されます。

クラス要素

Javaクラスは、各プロパティタイプに暗黙的に関連付けられます。 この要素を使用すると、プロパティタイプの暗黙クラスに過剰な負荷がかかります。

下記の例で、*String*プロパティタイプが宣言され、*java.net.URI*クラス内のJavaBeanプロパティに相当します。

```
<property name="myURIProperty" type="String" export="true">
<class>java.net.URI</class>
</property>
```

プロパティタイプ

以下の表は、サポート対象のプロパティタイプとそれらのデフォルトJavaBeanプロパティタイプを示します。

型	JavaBeanタイプ
Boolean	java.lang.Boolean
Byte	java.lang.Byte
Short	java.lang.Short
Long	java.lang.Integer
LingInt	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
String	java.lang.String
Memo	java.lang.String
Date	java.util.Date
Time	java.sql.Time
Timestamp	java.sql.Timestamp
Password	java.lang.String
File	java.io.File
Url	java.net.URL

サポート対象JavaBeanタイプの完全なリストについては、JavaBeansマニュアルを調べてください。

11 JVM設定ファイル

本節では、JVM設定ファイルに使用するシンタックスに関する情報を説明します。

ファイルが以下の方法で構成されます。

```
<jvmConfiguration>

<jarLocation/>
<jarLocation/>

<jars>
<jar/>
<jar/>
<jar/>
</jars>

<jvmOptions>
<jvmOption/>
<jvmOption/>
</jvmOptions>

<import/>
<import/>

</jvmConfiguration>
```

jvmConfiguration要素

ルート要素は、*jvmConfiguration*でなくてはなりません。
以下の属性があります。

属性	オプション	型	説明
id	いいえ	文字列	設定の固有識別子を定義します。

以下のサブ要素があります。

要素	オプション	型	説明
jarLocation	はい	文字列	コネクタによって使用されるクラスパスのパスを定義します。
jars	はい		コネクタによって使用されるアーカイブを定義します。
import	はい	文字列	外部ファイルからクラスパスをインクルードするために使用します。
jvmOptions	はい		JVMオプションを定義するために使用します。

jarLocation要素

コネクタのクラスパスは、コード実行に必要な複数のアーカイブ（.jarや.zipファイルなど）を参照する1つ以上のパスで構成されます。 コネクタごとに、アーカイブを検索するためのパスを定義することができます。 パス値は、Connect-Itインストールlibディレクトリに相対的であるか、絶対パスです。 アーカイブは、パスが宣言された順番で検索されます。

例：

```
<jarLocation>./com.mycompany.myeis</jarLocation>
<jarLocation>c:/myEIS/myEISPath</jarLocation>
```

デフォルトで、*jarLocation*要素を指定しなければ、使用されるパスはConnect-Itインストール**lib**ディレクトリになります。

Jars要素

以下のサブ要素があります。

要素	オプション	型	説明
jar	はい	string	クラスパスのアーカイブエントリを定義します。

Jar要素

以下の属性があります。

属性	オプション	型	デフォルト	説明
groupId	いいえ	string		アーカイブのグループ識別子を定義します。
provided	はい	boolean	true	当アーカイブがクラスパス検索パスのいずれかにあるか、パスをユーザが入力する必要があるかを指定します。 値'true'は、インストールによって入力されることを指定します。 この場合、'optional'属性が無視されます。
optional	はい	boolean	false	アーカイブがオプションであるかどうかを指定します。 値'false'は、アーカイブがクラスパス検索パスのいずれかにあるか、アプリケーション（'Java/Configure the JVM'メニュー）またはコネクタ（ウィザードの'Configure the JVM'ページ）で定義された追加クラスパスにあるかを指定します。

属性	オプション	型	デフォルト	説明
version	はい	string		アーカイブバージョンをアーカイブに付けるために使用します。 アーカイブの完全名は、name-version.jarになります。この拡張子が見つからない場合、完全名name-version.zipで、新しい検索が行われます。
versionNeeded	はい	boolean	true	アーカイブを名前とバージョンを使用して検索する必要があるかどうかを指定します。 ■ 値'true'は、検索を名前とバージョンで行うことを指定します。 ■ 値'false'は、検索を名前とバージョンで行ってからパスごとに名前のみで行うことを指定します。

値は、追加するアーカイブの名前を参照する必要があります。

アプリケーションによって入力されるxercesImpl-2.6.2.jarライブラリのサンプルのクラスパスエントリ

```
<jar groupId="xerces" optional="false" provided="true" version="2.6.2" versionNeeded="true">xercesImpl</jar>
```

jvmOptions要素

この要素は、追加のJVMオプションを定義するために使用されます。

以下のサブ要素があります。

要素	オプション	型	説明
jvmOption	はい	string	JVMオプションを定義します。

例：

```
<jvmOptions>
<jvmOption>-Xmx125m</jvmOption>
<jvmOption>-Dcom.sun.management.jmxremote</jvmOption>
</jvmOptions>
```

import要素

コネクタの設定に加えて、追加のJVM設定要素を入力することができます。 これらの要素は、同じシンタックスを使用する1つまたは複数のファイルで宣言されます。 import宣言を行う場所に応じて、宣言が現在の定義の前または後になります。 値は、インポートするファイルの相対パスを参照する必要があります。

例：

```
<import>../shared/jca-container-javaconf.xml</import>
```

12 データベース記述ファイル

本節では、記述ファイルに使用するシンタックスに関する情報を説明します。

ファイル構造

ファイルが以下の方法で構成されます。

```
{CONNECTORDESC
//property list
//property name=property value
Name=
InternalName=
...
}
```

プロパティ

以下の表は、コネクタのプロパティを示します。

プロパティ	型	オプション	デフォルト値	説明
Common properties				

プロパティ	型	オプション	デフォルト値	説明
Name	string	いいえ		表示するコネクタ名
InternalName	string	いいえ		コネクタの内部の名前（一意）
ParentInternalName	string	はい		所属先の親ノードの名前
HTMLHelp 文字列 はい html フォーマットでの説明				
Key	string	いいえ		有効化キー
Icon				
IconFile	string	いいえ		コネクタのアイコンの相対パス（.bmp）
Schedulers				
Sched.CanUsePointer	boolean	はい	true	SDKは、スケジュールポインタをサポートしません。 この値は0に設定する必要があります。 *Sched.CanUsePointer=0*
Cache				
Cache.SupportCache	boolean	はい	true	SDKは、メタデータキャッシュをサポートしません。この値は0に設定する必要があります。 *Cache.SupportCache=0*
Timezone				
Tmz.HandleServerDelay	boolean	はい	true	SDKは、サーバ時間の違いをサポートしません。 この値は0に設定する必要があります。 *Tmz.HandleServerDelay=0*
External formats				
ExtFmt.Use	boolean	はい	true	SDKは、拡張フォーマットをサポートしません。この値は0に設定する必要があります。 *ExtFmt.Use=0*

プロパティ	型	オプション	デフォルト値	説明
Wizard				
Wizard.File	string	はい		ウィザードファイルの相対パス
Java				
Java.Class	string	いいえ		コネクタのJavaクラスを以下のように指定する必要があります。 Java.Class=com.hp.ov.cit.container.RAContainer
Java.Configuration.File	string	はい		設定ファイルの相対パス
Java.JVMConfigurationFile	string	はい		JVM設定ファイルの相対パス
Java.HasOptions	string	はい	false	この値は、1. Java.HasOptions=1に設定する必要があります。
Java.SupportProxy	ブーリアン	はい	false	プロキシサーバ設定がサポートされているかどうか
Java.HasDefaultProxyRegistration	ブーリアン	はい	false	プロキシサーバのレコードが、<java.net.ProxySelector>クラスで導入された関数を使用するかどうか
Miscellaneous				

プロパティ	型	オプション	デフォルト値	説明
RedeployOnChange	string	はい		このリスト内のプロパティを変更すると、コネクタを再配布する必要があります。 例： RedeployOnChange=cs_CacertsFile cs_KeystoreFile as_KeystorePassword

例

以下は、SDKを使用して作成された*outbound*タイプコネクタのサンプル設定ファイルです。

```
{CONNECTORDESC
Name=MyEIS
InternalName=com.mycompany.myeis
ParentInternalName=com.mycompany
HTMLHelp=Connector to interact with my eis
Key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
IconFile=myeis.bmp

EventDriven=0
SupportParallelization=1
Transac.CanSupportTransactions=1
Cnx.HasCnx=1
Cnx.CanDisableReconnection=0
Wizard.File=myeis-wizard.xml
Java.Class=com.hp.ov.cit.container.RAContainer
Java.Configuration.File=myeis-config.xml
Java.JVMConfiguration.File=myeis-jvmconf.xml
Java.HasOptions=1

// The following properties must always have these values
Sched.CanUsePointer=0
Cache.SupportCache=0
Tmz.HandleServerDelay=0
```

```
ExtFmt.Use=0
}
```

追加情報

複数の記述

記述ファイルには、それぞれが*CONNECTORDESC*セクションに対応する複数の記述を入れることができます。 コネクタごとに1つの記述ファイルを記述することを推奨しますが、複数の記述を1つのファイルに入れると、コネクタカテゴリを定義したり同じEISの複数バージョンを管理したりする場合に便利です。

コネクタ階層

*ParentInternalName*プロパティを使用することによって、親ノードまたはカテゴリの内部名をコネクタ階層内で指定します。 カテゴリは、さらに簡単なフォーマットで記述ファイルにも定義されます。

```
{CONNECTORDESC
InternalName=...
ParentInternalName=...
Name=...
HTMLHelp=...
IconFile=...
}
```

*ParentInternalName*プロパティを指定しない場合、カテゴリ（またはコネクタ）は階層のルートにあります。

Connect-Itには、事前に定義されたいくつかのカテゴリがあります。

カテゴリ	内部名
アプリケーションコネクタ	Application_connectors
プロトコルコネクタ	Protocol_connectors
ERPコネクタ	ERP_connectors
インベントリコネクタ	Gateways

13 Javaコード

JavaBeans

サポート対象タイプ

JCA仕様のいくつかのインタフェースをJavaBeansとして実装する必要があります。 以下のインタフェースは、SDKによって使用されます。

javax.resource.spi.ManagedConnectionFactory javax.resource.spi.ResourceAdapter javax.resource.cci.ConnectionSpec javax.resource.spi.ActivationSpec

以下の表は、プロパティによって認証される値タイプを示します。

java.lang.Boolean
java.lang.String
java.lang.Integer
java.lang.Double
java.lang.Byte
java.lang.Short
java.lang.Long
java.lang.Float

java.lang.Character

SDKでは、よく使用される他のタイプがこのリストに追加されます。 以下のタイプがサポートされます。

java.util.Date
java.sql.Time
java.sql.Timestamp
java.io.File
java.net.URL
java.net.URI

検証

場合によって、JavaBeanオブジェクトプロパティの取り得る値が他のプロパティに従って決まります。 オブジェクトはプロパティの更新順序を制御しないため、SDKではインタフェースによってこの問題の代替方法が提示されます。

```
public interface ValidatingBean
{
public void validate() throws InvalidPropertyException;
}
```

このインタフェースは、プロパティのすべてが更新された後に実装するJavaBeanについて検証または初期化フェーズを管理するために使用します。

ロギング

SDKは、*Jakarta Commons Logging (JCL)*フレームワークを使用して、Connect-Itログ内にメッセージを記録します。 以下のコードを追加して、この関数をJavaクラスから使用します。

```
package com.mycompany.myeis;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyEISClass
{
private static final Log log = LogFactory.getLog(MyEISClass.class);

...
}
```

JCLは、各メッセージの優先レベルを定義します。 以下のレベルがConnect-Itによって使用されます。

- error - エラーメッセージ
- info - 情報
- warn - 警告メッセージ
- debug - 「デバッグ」モードが有効になっている場合のみ記録されるデバッグメッセージ

メッセージをConnect-Itログに記録するには、これらの *org.apache.commons.logging.Log*インタフェースメソッドを使用します。

```
log.error(Object message);
log.error(Object message, Throwable t);
log.warn(Object message);
log.warn(Object message, Throwable t);
log.info(Object message);
log.info(Object message, Throwable t);
log.debug(Object message);
log.debug(Object message, Throwable t);
```

Log4Jサポート

JCLフレームワークを使用して、実装されたロギングシステムであるLog4J、JDK Loggingなどに統一的にアクセスします。

デフォルトでは、Log4Jライブラリの設定が使用されます。Jog4Jレイヤでログされるすべてのメッセージは、呼び出しが直接かJCL API経由であるかに関わらず、Connect-Itで考慮されます。

JDKロギングサポート

静的設定による、JDKによって提供されるロギングフレームワークのサポートが追加されています。デフォルトの静的設定は <JRE_HOME>\lib\logging.propertiesファイルで記述されます。

コネクタがインスタンス化されると、JDKのルートロガーのログレベルがConnect-Itアプリケーション用に設定されたレベルになるように変更されます。JDKロギングフレームワークの直接呼出しを経由して得られるすべてのログイベントは、アプリケーションにリダイレクトされます。

国際化

SDKでは、Javaの標準国際化メカニズムを使用します。 このメカニズムをコードによって実装するには、国際化に必要な文字列を含む1つまたは複数の*properties*ファイルを作成する必要があります。

例

com/mycompany/myeis/i18n/mymessages.propertiesファイル

```
connection.error = Connection error.
execution.failed = Execution failed.
```

com/mycompany/myeis/i18n/mymessages_fr.propertiesファイル

```
connection.error = Erreur de connexion.
execution.failed = Echec de l'execution.
```

com/mycompany/myeis/MyEISClass.javaファイル

```
package com.mycompany.myeis;

import java.util.ResourceBundle;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyEISClass
{
private static final ResourceBundle bundle = ResourceBundle.getBundle("com
.mycompany.myeis.i18n.mymessages");
private static final Log log = LogFactory.getLog(MyEISClass.class);

public void execute()
{
try
{
...
}
catch(Exception e)
{
log.error(bundle.getString("execution.failed"), e);
}
}
}
```