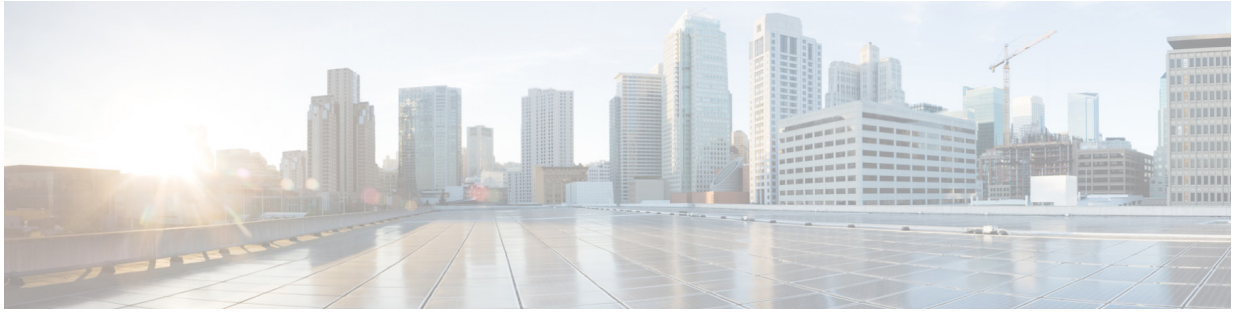


Cisco UCS Director Open Automation の手順書

リリース 1.0
2015 年 4 月に発行

Cisco Systems, Inc.
www.cisco.com

シスコは世界各国 200 箇所にオフィスを開設しています。
各オフィスの住所、電話番号、FAX 番号は次のシスコ Web サイトをご覧ください。
www.cisco.com/go/offices

目次

1 使用する前に

開発者が Cisco UCS Director のプラットフォームに貢献できる機能がプラットフォームの Cisco UCS Director 固有のモジュラ アーキテクチャとその機能によって実現されます。開発者はこの機能を利用して、Cisco UCS Director に複数のコンポーネントを追加することで、カスタマイズと拡張を提供できます。

1.1 Eclipse IDE への SDK バンドル プロジェクトのインポート

Open Automation SDK と、その他すべての Cisco UCS Director ソフトウェアは、Cisco.com のダウンロード エリアで入手できます。SDK をダウンロードするには、UCS Director SDK へのリンクをクリックします。サンプル SDK プロジェクトの zip ファイルをファイル システムに解凍します。

Eclipse IDE に SDK バンドル プロジェクトをインポートするには、以下の手順に従います。

1. SDK バンドル アーカイブを取得し、そのコンテンツを適切なフォルダに抽出します。
2. Eclipse を起動します。
3. [ファイル（File）] > [インポート（Import）] を選択します。
4. [インポート（Import）] ダイアログボックスで、[一般（General）] > [既存プロジェクトをワークスペースへ（Existing Projects into Workspace）] を選択します。
5. [次へ（Next）] をクリックします。
6. [ルートディレクトリの選択（Select root directory）] を選択し、プロジェクトを抽出した場所を参照します。
7. [終了（Finish）] をクリックします。

すべてが問題なく自動的にコンパイルする必要があります。/open-auto-sdk/lib パス下に Cisco UCS Director SDK を置換する必要がある場合とない場合があります。

1.2 FOO モジュールの作成

build.xml **ファイル**の Ant ターゲットを実行します。
ビルドが正常に終了すると、Foo モジュールの zip ファイルが作成されます。

モジュールの名前を変更するには、次の手順を実行します。

手順 1：モジュールのクラス名を変更します。
手順 2：module.properties ファイルの名前を次のように変更します。
　　Name=<新しいモジュール名>
手順 3：ANT ターゲットを実行して zip ファイルを作成します。

---

2 モジュールの管理

モジュールは、Cisco UCS Director への最上位の論理エントリ ポイントです。機能を追加または拡張する場合、モジュールが Cisco UCS Director で開発および展開される必要があります。

2.1 モジュールの作成

モジュールを作成するには、次の手順を実行します。

1. AbstractCloupiaModule を拡張するクラスを実装してモジュールを作成します。
2. 必要なタスクのインターフェイスを実装してタスクを作成します。
3. ANT を実行して、モジュールをパッケージ化します。

サンプル：モジュールを作成します

```
public class FooModule extends AbstractCloupiaModule
{
        // To Register task
        @Override
        public AbstractTask[] getTasks() {
        //You can implement for the Tasks.
        }
         // we registering all top level reports
        @Override
        public CloupiaReport[] getReports() {
                CloupiaReport[] reports = new CloupiaReport[];
                ...
                return reports;
        }
        @Override
        public void onStart(CustomFeatureRegistry cfr) {
         }
  }
```

2.2 モジュールの公開

モジュールをプラットフォームのランタイムに公開するために、module.properties という名前のファイルがモジュールとともに提供されます。プロパティ ファイルは、モジュールの特定のプロパティを定義します。

Cisco UCS Director によって提供される module.properties ファイルを開発者が変更してはいけません。改ざんを防ぐために Cisco UCS Director によってプロパティ ファイルが検証されます。

2.2.1 module.properties ファイルの内容

#id：モジュールの一意な識別子（3 ～ 5 文字の小文字の ASCII 英字に制限することを推奨します）。

moduleID=foo

#version：モジュールの現在のバージョン。

```
version=1.0
```
#ucsdVersion：このモジュールのバージョンが動作するのに最適な Cisco UCS Director のバージョン。
```
ucsdVersion=5.3.0.0
```
#category：すべてのタスクが配置されているパス。
```
category=/foo
```
#format：このモジュールのバージョンがフォーマットされます。
```
format=1.0
```
#name：オープン オートメーション レポートでモジュールを識別するためのわかりやすい文字列。
```
name=Foo Module
```
#description：モジュールが実行する内容を説明する簡潔な記述。
```
description=UCSD Open Automation Sample Module
```
#contact：モジュールのユーザがサポートを依頼するために使用する電子メールアドレス。
```
contact=support@cisco.com
```
#key：Cisco UCS Director Open Automation グループがモジュール検証のために提供する暗号化されたキー。
```
key=5591befd056dd39c8f5d578d39c24172
```

2.3 Cisco UCS Director でのモジュールの展開

Cisco UCS Director のユーザ インターフェイス（UI）では、Open Automation のコントロールが提供され、それを使用してモジュールをアップロードおよび管理できます。これらのコントロールを使用して、モジュールの zip ファイルを Cisco UCS Director にアップロードします。
シェル管理者のアクセス権があることを確認します。シェル管理者のアクセス権は、モジュールを展開するために Cisco UCS Director サービスを再起動したり、モジュールを非アクティブにするために必要になります。

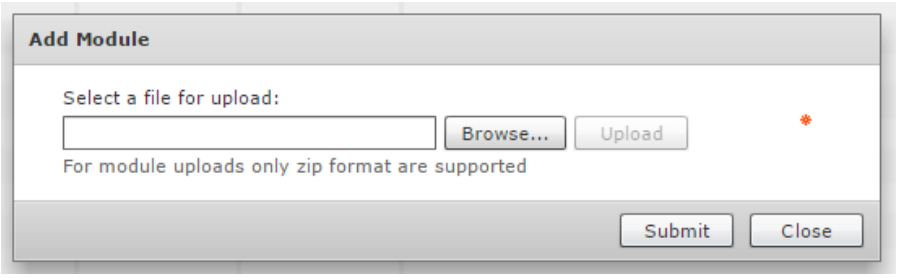
モジュールを展開するには、次の手順を実行します。

手順 1： Cisco UCS Director で、[管理（Administration）] > [オープンオートメーション（Open Automation）] を選択します。
[モジュール（Modules）] タブが開いて、次の列が表示されます。

カラム（Column）	説明
ID	モジュールの ID。
[名前（Name）]	モジュールの名前。
説明（Description）	モジュールの説明。

カラム（Column）	説明
[バージョン（Version）]	モジュールの現在のバージョン。モジュール開発者は、モジュールのバージョンの管理方法を決定する必要があります。
[互換性あり（Compatible）]	このモジュールのサポートに最も適した Cisco UCS Director のバージョンを表示します。
連絡先（Contact）	モジュールのテクニカル サポート担当者の連絡先情報。
アップロード時間（Upload Time）	モジュールがアップロードされる時間。
[ステータス（Status）]	モジュールのステータス。ステータスには、有効、無効、アクティブ、および非アクティブが含まれます。 ユーザは、モジュールを有効にするか、無効にするかを制御できます。有効にすると、Cisco UCS Director はモジュールの初期化を試みます。無効にすると、Cisco UCS Director はモジュールを完全に無視します。Cisco UCS Director がモジュールを例外なしで正常に初期化できる場合、そのモジュールはアクティブです。 ✐ 注：アクティブは、必ずしも、モジュール内のすべてが正しく機能していることを意味しているわけではなく、単に、モジュールが起動したことを示しているに過ぎません。非アクティブは、Cisco UCS Director がモジュールの初期化を試みたときに、重大なエラーが発生して初期化ができなかったことを意味します。非アクティブ フラグの一般的な原因：モジュールが誤ったバージョンの Java でコンパイルされたか、モジュールにクラスが含まれていなかった。
検証済み（Validated）	モジュールが検証されているかどうかを示します。

手順 2: [追加(Add)] をクリックして、新しいモジュールを追加します。
[モジュールを追加(Add Module)] ダイアログボックスが表示されます。



手順 3: [参照(Browse)] をクリックして、アップロードするモジュールの zip ファイルの場所を選択します。
手順 4: 必要な情報を入力して、[送信(Submit)] をクリックします。

> 重要:新しく追加されたモジュールをロードするには、そのモジュールを有効にしてから、Cisco UCS Director サービスを再起動する必要があります。有効にすると、Cisco UCS Director はモジュールの初期化を試みます。無効にすると、モジュールは無視されます。

手順 5: モジュールを選択し、[有効(Enable)] をクリックして、モジュールを有効にします。
手順 6: モジュールを選択し、[追加(Add)] をクリックして、モジュールをアクティブ化します。
[モジュールのアクティブ化(Activate Module)] ダイアログボックスが表示されます。
手順 7: [送信(Submit)] をクリックすると、モジュールがアクティブ化されます。エラーが発生した場合は、モジュールが許容されなかった理由を明確に示すエラー メッセージが表示されます。Cisco UCS Director がモジュールを例外なしで正常に初期化できる場合、そのモジュールはアクティブです。

> 注:アクティブは、必ずしも、モジュール内のすべてが正しく機能していることを意味しているわけではなく、単に、モジュールが起動したことを示しているに過ぎません。非アクティブは、Cisoc UCS Director がモジュールの初期化を試みたときに、重大なエラーが発生して初期化ができなかったことを意味します。非アクティブ フラグの一般的な原因:モジュールが誤ったバージョンの Java でコンパイルされたか、モジュールにクラスが含まれていなかった。

手順 8: Cisco UCS Director サービスを停止および再起動します。

a. シェル管理者のアクセス権で**シェル メニュー**を開きます。**Cisco UCS Director のシェル メニュー**が開き、**下のメニューから番号を選択する**ように求めるプロンプトが表示されます。

b. SELECT プロンプトで、サービスを停止させる番号を入力し、次にサービスを再開させるための適切な番号を入力します。

Cisco UCS Director は再起動に数分かかります。システムの準備が整ったら、モジュールがロードされ、タスクが UI に表示されます。

2.3.1 モジュールの非アクティブ化

変更内容を反映させるためにモジュールを非アクティブにします。モジュールを非アクティブにするには、Cisco UCS Director サービスを停止させてから再起動する必要があります。

モジュールを非アクティブにするには、次の手順を実行します。

手順 1: 非アクティブ化する必要のあるモジュールを [モジュール(Modules)] テーブルで選択し、続いて [非アクティブ化(Deactivate)] コントロールをクリックします。

手順 2: Cisco UCS Director サービスを停止および再起動します。モジュールのアクティブ化の後と同じ手順に従います。

---

3 モジュールの操作

モジュールには、次のコンポーネントを含めることができます。

- タスク:ワークフローを定義する際に使用できるワークフロー タスク。
- レポート:Cisco UCS Director UI に表示されるレポート。レポートにはアクション ボタンが含まれている場合があります。
- ウィザード:特定のアクション(複数も可)を実行するためにユーザからの入力を収集する UI コンポーネント。
- トリガー:満たされたときに何らかのアクション(複数も可)に関連付けできる条件。例:VM のシャットダウン、VM の起動など。

上記の主要コンポーネントはそれぞれ、次に示す 1 つ以上のサブ コンポーネントを利用できます。

- オブジェクト ストア
- 注釈
- 一覧

- テーブル
- コネクタ
- ログ

3.1 オブジェクト ストア

オブジェクト ストアは、データベースの持続化のためにシンプルな API を提供します。データベースへのオブジェクトの持続化を必要とするモジュールは、通常、オブジェクト ストア API を使用してすべての CRUD（作成、読み取り、更新、削除）操作を実行します。

Cisco UCS Director では、データベースとして MySQL を使用します。プラットフォーム ランタイムは、DataNucleus が提供する Java Data Object（JDO）ライブラリを利用して、オブジェクト クエリ表現によってすべての SQL 操作を抽象化します。このプロセスより、データの持続化について開発の簡素化、高速化が可能になります。オブジェクト ストアのマニュアルには、JDO を使用して CRUD 操作を行う方法を示す項も用意されています。

3.2 注釈

注釈は、モジュール開発において最も重要な部分の 1 つです。ほとんどのアーティファクトは注釈に基づいて生成されます。注釈によって開発作業が簡単かつ容易になります。

注釈は、持続化、レポート生成、ウィザード生成、および各種タスクに使用されます。

3.2.1 持続化の注釈

永続化に使用される注釈については、「持続化オブジェクトとしてのクラスのマーキング」の項を参照してください。

3.2.2 タスクの注釈

ワークフローにタスクが含まれている場合、特定の入力を求めるプロンプトがユーザに表示されます。ユーザが入力を求められるのは、タスクを表すクラスのフィールドに注釈を使用してマーキングが付けられている場合です。FormField の注釈によって、ユーザに表示する UI 入力フィールドのタイプ（テキスト フィールド、ドロップダウンリスト、チェック ボックス）が決まります。詳細については、「タスク」の項を参照してください。

3.3 値のリスト（LOV）

リストは、タスクに対する正しい入力値を簡単に取得できるようにユーザに対して表示される、ドロップダウン式の値のリスト（LOV）を表します。既存のリストを再利用することも、タスク UI に表示する独自のリストを作成することもできます。

Cisco UCS Director には、多くの事前作成済みのリスト プロバイダーが定義されており、各モジュールでこれらを簡単に使用して、ユーザの入力を促すことができます。リスト プロバイダーの 1 つの使用方法を示した例については、「独自のリスト プロバイダーの定義」および「タスク」の項を参照してください。

3.3.1 独自のリスト プロバイダーの定義

独自のリスト プロバイダーを定義して、そのプロバイダーをシステムに登録するようにプラットフォーム ランタイムに要求します。
リスト プロバイダー クラスは LOVProviderIf インターフェイスを実装し、getLOVs( ) メソッドの実装を提供します。

*サンプル:**FormLOVPair** オブジェクトの配列を返します。*

```
class MyListProvider implements LOVProviderIf
{

       /**
        * Returns array of FormLOVPair objects.  This array is
what is shown
        * in a dropdown list.
        * A FormLOVPair object has a name and a label. While the
label is shown
        * to the user, the name will be used for uniqueness
        */
       @Override
       public FormLOVPair[] getLOVs(WizardSession session) {

              // Simple case showing hard-coded list values

              FormLOVPair http = new FormLOVPair("http", "HTTP");
               // http is the name, HTTP is the value
              FormLOVPair https = new FormLOVPair("https",
"HTTPs");

              FormLOVPair[] pairs = new FormLOVPair[2];
              pairs[0] = http;
              pairs[1] = https;
              return pairs;
       }
}
```

3.4 テーブル（表形式のレポート）

ユーザが選択して使用できる、既存の表形式レポートを使用します。表形式のレポートに関する詳細については、「表形式のレポート」の項を参照してください。

3.5 タスク

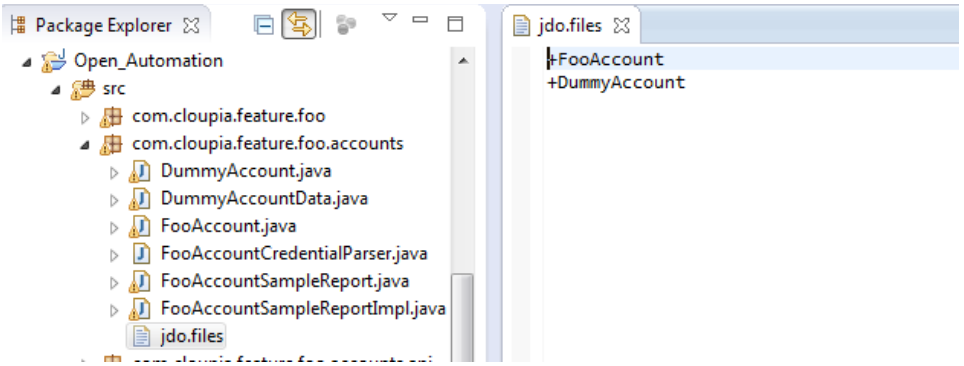
タスクはワークフロー定義で使用されます。タスクに関する詳細については、「タスク」の項を参照してください。

4 持続化オブジェクトとしてのクラスのマーキング

データベースで持続的に存在する必要がある POJO クラスは、JDO の適切な注釈を使用して定義し、マーキングする必要があります。JDO クラスの作成後、以下を行う必要があります。

- 永続化のための JDO 機能拡張を実装します。
- jdo.files に config クラスを追加します。

注：jdo.files は、タスクが作成されたときと同じパッケージである必要があります（次の図を参照）。



サンプル：持続化オブジェクトをマーキングします

```
@PersistenceCapable (detachable = "true", table =“sampleAccount”)
public class  SampleAccount
{
    @Persistent
    private String AccountName;
      @Persistent
    private String userName;
      @Persistent
    private String password;
}
```

4.1 永続可能オブジェクトの保存/取得

データベースへのオブジェクトの持続化を必要とするモジュールは、主にオブジェクト ストア API を使用してすべての CRUD 操作を実行するものとします。プラットフォーム ランタイムは、DataNucleus が提供する Java Data Object（JDO）ライブラリを利用して、すべての SQL 操作およびオブジェクト クエリ表現を抽象化します。

4.1.1 データを保存するコード

```
SampleAccount obj = new SampleAccount();
obj.setAccountName("openauto-account");
obj.setUserName(“Admin”);
obj.setPassword(“password”);
ObjStore store = ObjStoreHelper.getStore(SampleAccount.class);
store.insert(obj);
```

4.1.2 データを取得するコード

```
//retrieving all objects
ObjStore store = ObjStoreHelper.getStore(SampleAccount.class);
List list = store.queryAll();

//retrieving objects with query
ObjStore store = ObjStoreHelper.getStore(SampleAccount.class);
String query = "dcName == 'Default Datacenter'";
List filerList = store.query(query);
```

5 作業（Task）

ワークフロー タスクは、Cisco UCS Director が保持するタスク ライブラリへの貢献に必要なアーティファクトを提供します。タスクはワークフロー定義で使用されます。

5.1 タスクのコンテンツ

最低でも、タスクは次のクラスでなければなりません。

- TaskConfigIf インターフェイスを実装するクラス。
- AbstractTask クラスのメソッドを拡張および実装するクラス。

5.1.1 TaskConfigIf

TaskConfigIf インターフェイスを実装するクラスには、ユーザにプロンプトを表示するために注釈が付けられたすべての入力フィールド定義が含まれています。

サンプル：タスクを実装します。

```
@PersistenceCapable(detachable = "true", table =
"foo_helloworldconfig")
public class HelloWorldConfig implements TaskConfigIf {
       // you can define your task implementation
}
```

5.1.2 AbstractTask

タスクの実装によって、AbstractTask の抽象クラスが拡張され、すべての抽象化メソッドの実装が提供されます。この抽象クラスは、タスクに関連するすべてのビジネス ロジックを処理するメイン クラスです。ビジネス ロジックの実装が記述されるこのクラスで最も重要なメソッドは、executeCustomAction() です。その他のメソッドは、プラットフォーム ランタイムでタスクをオーケストレーション デザイナ ツリーに表示したり、ワークフロー内でタスクのドラッグ アンド ドロップを可能にしたりするための十分なコンテキストを提供します。

サンプル：タスクを抽象化します。

```
public class HelloWorldTask extends AbstractTask
{
@Override
```

```
public void executeCustomAction(CustomActionTriggerContext
context, CustomActionLogger  actionLogger) throws Exception
{
	long configEntryId =
context.getConfigEntry().getConfigEntryId();
//retrieving the corresponding config object for this handler
	HelloWorldConfig config = (HelloWorldConfig)
context.loadConfigObject();
                        }
	@Override
	public TaskConfigIf getTaskConfigImplementation() {
		return new HelloWorldConfig();
	}
	@Override
	public String getTaskName() {
		return HelloWorldConfig.displayLabel;
	}
	@Override
	public TaskOutputDefinition[] getTaskOutputDefinitions() {
		return null;
	}

}
```

タスクをモジュール クラスに次のように登録します。

```
public class FooModule extends AbstractCloupiaModule {
@Override
	public AbstractTask[] getTasks() {
		AbstractTask task1 = new HelloWorldTask();
		AbstractTask[] tasks = new AbstractTask[1];
		tasks[0] = task1;
		return tasks;
	}
...
}
```

6 レポート

Cisco UCS Director では、オープン オートメーション レポートがデータを表示したりデータを取得するためにアップロードされたモジュールの UI で使用されます。レポートを作成する 2 つの方法は次のとおりです。

- Plan Old Java Object（POJO）と注釈のアプローチ。詳細については、「POJO と注釈のアプローチ」の項を参照してください。
- TabularReportGeneratorIf インターフェイスの実装。詳細については、「表形式のレポート」の項を参照してください。

6.1 レポートの登録

レポートを作成後、システムにレポートを登録する必要があります。

サンプル：レポートを登録します。

```
public class FooModule extends AbstractCloupiaModule {

@Override
      public CloupiaReport[] getReports() {
                    CloupiaReport[] reports = new
CloupiaReport[2];
                    reports[0] = new SampleReport();
                    reports[1] = new FooAccountSampleReport();
                    return reports;
}
}
```

6.2 POJO と注釈のアプローチ

次のクラスを使用して、POJO ベースのレポートを作成できます。

- CloupiaEasyReportWithActions
- CloupiaEasyDrillableReport

レポートを作成するには、持続化のために開発された Java Data Object（JDO）POJO を使用して、注釈をいくつか追加します。レポートは、UI に表示できます。

レポートを作成するには、次の手順を実行します。

手順 1：データ ソース POJO に **com.cloupia.service.cIM.inframgr.reports.simplified.ReportableIf** インターフェイスを実装します。レポートに表示したくない POJO の任意のインスタンスを除去するには、ReportableIf インターフェイスで **getInstanceQuery** メソッドを使用して、フレームワークで使用される述語を返します。

手順 2：レポートに表示する必要がある POJO のフィールドごとに、@ReportField 注釈を使用して、レポートに含めるフィールドとしてマーキングします。

*サンプル：注釈付き**POJO***

```
public class SampleReport implements ReportableIf{
@ReportField(label="Name")
@Persistent
private String name;

public void setName(String name){
this.name=name;
}
```

```
public String getName(){
      return this.name;
}

@Override
      public String getInstanceQuery() {
            return "name  == '" + name+  "'";
}
}
```

この POJO はデータ ソースと呼ばれることもあります。

手順 3: **com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaEasyReportWithAction** または **com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaEasyDrillableReport** クラスを拡張し、レポート名（このレポートを一意に識別するためのもの）、このレポートの**ラベル**（ユーザに表示されるもの）、およびデータ ソース（今作成した POJO）を提供します。

- CloupiaEasyReportWithActions：レポートするアクションを割り当てる必要がある場合は、このクラスを使用します。
- CloupiaEasyDrillableReport：ドリル ダウン レポートを実装する必要がある場合は、このクラスを使用します。

両方のクラスは、POJO と注釈のメソッドを使用してレポートを作成するために使用されます。

---

*サンプル:**POJO** ベースのレポート*

手順 1：ReportableIf の実装

ユーザが述語を返す getInstanceQuery メソッドを使用して、そのメソッドがレポートに表示したくない POJO の任意のインスタンスを除去するためにフレームワークで使用される場合、*DummySampleImpl* クラスは ReportableIf インターフェイスを実装します。

```
@PersistenceCapable(detachable = "true")
public class DummySampleImpl implements  ReportableIf {
      @Persistent
      private String accountName;
      @ReportField(label="Name")
      @Persistent
      private String name;
}
```

手順 2：CloupiaEasyReportWithActions の拡張

CloupiaEasyReportWithActions クラスを拡張し、レポート名（レポートを取得するために一意なもの）、データ ソース（pojo クラス）、およびレポート ラベル（UI に表示されるもの）を提供してレポートを取得します。

getActions() メソッドからアクション オブジェクトを返すことによってアクションをこのレポートに割り当てることができます。

```
public class DummySampleReport extends
CloupiaEasyReportWithActions {

//Unique report name that use to fetch report, report label use
to show in UI and dbSource use to store data in CloupiaReport
object.
	private static final String name =
"foo.dummy.interface.report";
	private static final String label = "Dummy Interfaces";
	private static final Class dbSource =
DummySampleImpl.class;

	public DummySampleReport() {
		super(name, label, dbSource);
	}

	@Override
	public CloupiaReportAction[] getActions() {
// return the action objects,if you don't have any action then
simply return null.
	}
}
```

UI でレポートを表示するには、システムに *DummySampleReport* レポートを登録します。詳細については、「レポートの登録」の項を参照してください。

6.3 表形式のレポート

TabularReportGeneratorIf インターフェイスを使用して表形式のレポートを作成します。
アプローチは、UI に表示するすべてのデータを含む TabularReportInternalModel のインスタンスを作成することです。
表形式のレポートを作成するには、次の手順を実行します。

手順 1: TabularReportInternalModel を作成し、ヘッダーにデータを移入し、各列のテキスト値を追加して行に記入します。

手順 2: **com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaReportWithActions** または **com.cloupia.service.cIM.inframgr.reports.simplified.DrillableReportWithActions** クラスを拡張します。

- CloupiaReportWithActions: アクションの有無にかかわらず、ドリルダウンなしでオープン オートメーション レポートを作成するには、このクラスを拡張します。

- DrillableReportWithActions：アクションの有無にかかわらず、ドリルダウンでオープン オートメーション レポートを作成するには、このクラスを拡張します。

手順 3：データ ソースの実装を指定し、***isEasyReport()*** メソッドが false を返すことを確認します。

サンプル：表形式のレポート

DummyReportImpl クラスは TabularReportGeneratorIf インターフェイスを実装しています。レポートに表示するデータについてより細かなコントロールが必要な場合は、このアプローチを使用して、TabularReportGeneratorIf インターフェイスを実装してレポートを作成します。

```
public class DummyReportImpl implements TabularReportGeneratorIf
{

private static Logger logger =
Logger.getLogger(DummyReportImpl.class);
             @Override
public TabularReport getTabularReportReport(ReportRegistryEntry
reportEntry, ReportContext context) throws Exception {
       TabularReport report = new TabularReport();
// current system time is taking as report generated time,
setting unique report name and the context of report
       report.setGeneratedTime(System.currentTimeMillis());
       report.setReportName(reportEntry.getReportLabel());
       report.setContext(context);

//TabularReportInternalModel contains all the data you want to
show in report
       TabularReportInternalModel model = new
TabularReportInternalModel();
        model.addTextColumn("Name", "Name");
       model.addTextColumn("VLAN ID", "VLAN ID");
       model.addTextColumn("Group", "Assigned To Group");
       model.completedHeader();
       model.updateReport(report);
       return report;
       }

}

public class DummySampleReport extends CloupiaReportWithActions {

       private static final String NAME = "foo.dummy.report";
       private static final String LABEL = "Dummy Sample";

        //Returns the implementation class
       @Override
       public Class getImplementationClass() {
```

```
                return DummyReportImpl.class;
            }
        //Returns the report label use to display as report name in
UI
            @Override
            public String getReportLabel() {
                return LABEL;
            }
        //Returns unique report name to get report
            @Override
            public String getReportName() {
                return NAME;
            }
        //For leaf report it should returns as false
            @Override
            public boolean isEasyReport() {
                return false;
            }
        //For drilldown report it should return true
            @Override
            public boolean isLeafReport() {
                return true;
            }

}
```

UI でレポートを表示するには、システムにレポートを登録します。詳細については、「レポートの登録」の項を参照してください。

6.4 ドリル可能なレポート

他のレポート内にネストされているレポートやドリル ダウンでのみ到達可能なレポートは、ドリル可能なレポートと呼ばれます。ドリル可能なレポートは、表形式のレポートのみに適用できます。

ドリル可能なレポートを作成するための重要なポイントは次のとおりです。

- レポート データ ソースは、POJO と注釈のアプローチによって実装される必要があります。レポートは **com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaEasyDrillableReport** クラスを拡張する必要があります。
- レポート データ ソースは、TabularReportGeneratorIf インターフェイスを使用して実装する必要があります。レポートは **com.cloupia.service.cIM.inframgr.reports.simplified.DrillableReportWithActions** クラスを拡張する必要があります。

どちらのクラスの場合も、ユーザがベース レポートをドリル ダウンすると表示されるレポートのインスタンスを指定する必要があります。getDrillDownReports() メソッドが呼び出されるたびに、同じインスタンスが返される必要があります。この場合の最善の方法は、レポートのアレイを初期化することです。詳細については、「レポートの登録」の項を参照してください。

サンプル:ドリル可能なレポート

```
public class DummyOneSampleReport extends
DrillableReportWithActions  {
      ...
            @Override
            public CloupiaReport[] getDrilldownReports() {
                  CloupiaReport[] ddReports = new
CloupiaReport[1];
                  ddReports[0] = new SampleReport();
                  return ddReports;
      }
}
```

6.5 アクション付きのレポート

レポートを作成する際、それらのレポートで動作するアクションも含めることができます。

次の項目はレポート アクションの開発に必要です。

- フォーム オブジェクト:フォーム オブジェクトは JDO とフォーム フィールド注釈付きの POJO です。フォーム オブジェクトは、レポート用に開発した設定オブジェクトと類似している必要があります。
- フォーム ハンドラ:レポート アクション用のロジックをハンドラに配置する必要があります。

com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaPageAction クラスを拡張し、次のメソッドをクラス内でオーバーライドします。

- definePage:フォームのレイアウトを定義します。
- loadDataToPage:フォームでデータのロードを処理します。
- validatePageData:ユーザが送信ボタンをクリックしたときに実行するあらゆるロジックを処理します。

サンプル:アクション付きのレポート

```
Public class DummyForm{
      @FormField(label = "Name", help = "Name")
      private String name;

      @FormField(label = "Id", type =
FormFieldDefinition.FIELD_TYPE_NUMBER)
      private boolean id;
      }
```

すべてのフォームにおいて、CloupiaPageAction を拡張することによって **Action** クラスが実装される必要があります。

```
public class DummyAction extends CloupiaPageAction{
private static Logger logger =
Logger.getLogger(DummyAction.class);
```

```
            private static final String formId =
"foo.simple.dummy.form";
      private static final String ACTION_ID =
"foo.simple.dummy.action";
            private static final String label = "Dummy Action";
// it returns the action id when you will click the action button
           @Override
      public String getActionId() {
            return ACTION_ID;
      }

// the form id will return when you will submit the form.
      public String getFormId()
      {
            return formId;
      }

//Returns report label to display in UI as report name
      @Override
      public String getLabel() {
            return label;
      }

//it returns the action type like how the form should appear from
UI.
      @Override
      public int getActionType() {
            return ConfigTableAction.ACTION_TYPE_POPUP_FORM;
      }
//if this returns false then no need to select report to appear
action button if it returns as true then
//We need to select the report to appear the action button in UI
in the registred context.
      @Override
      public boolean isSelectionRequired() {
            return false;
      }
//if this method returns as false then no need to do double click
to get report if it returns true then
//we have to do double click for getting report.
      @Override
      public boolean isDoubleClickAction() {
            return false;
      }
 // If this returns true then after clicking on action button it
will navigate to next level of the report.
      @Override
      public boolean isDrilldownAction() {
            return false;
      }
//define the form object like how the form should appear into UI.
      @Override
      public void definePage(Page page, ReportContext context) {
            page.bind(formId, DummyForm.class);
```

```
        }

//Loading data that you might want to show in the form when it is
first displayed to the user.
        @Override
public void loadDataToPage(Page page, ReportContext context,
WizardSession session) throws     Exception {
                String query = context.getId();
                        SimpleDummyForm form = new DummyForm();
                form.setName("dummy name - hardcoded for demo
purposes");
                session.getSessionAttributes().put(formId, form);
                page.marshallFromSession(formId);
                String dir =
FileManagementUtil.getDir(page.getSession());
                page.setSourceReport(formId + ".uploadFileName",
dir);
        }

// validating the data when the user presses the submit button .
@Override
public int validatePageData(Page page, ReportContext context,
WizardSession session) throws Exception {
//for page data validation put your code .
        }
        }
//Returns the label of action
        @Override
        public String getTitle() {
                return label;
        }
```

レポート クラスへのアクションの登録については、「表形式のレポート」の項を参照してください。

6.6 改ページ レポート

改ページ表形式のレポートを実装するには、次の 3 つのクラスを実装します。

1. CloupiaReportWithActions を拡張するレポート クラス
2. テーブルに表示するデータを提供するソース ソース クラス
3. 改ページ レポート ハンドラ クラス

改ページ レポートを実現するには、次の手順を実行します。

手順 1: レポート ファイルに CloupiaReportWithActions.java を拡張し、getPaginationModelClass および getPaginationProvider メソッドをオーバーライドします。

```
//Tabular Report Source class which provides data for the table
@Override
public Class getPaginationModelClass() {
```

```
return DummyAccount.class;
}

//New java file to be implemented for handling the pagination
support.
@Override
public Class getPaginationProvider() {
return FooAccountReportHandler.class;
}

Override the return type of the isPaginated method as true.
@Override
public boolean isPaginated() {
return true;
}
```

手順 2: getReportHint メソッドの戻りタイプを **ReportDefinition REPORT_HINT_PAGINATED_TABLE** としてオーバーライドし、改ページ レポートを取得します。

```
@Override
public int getReportHint(){
return ReportDefinition.REPORT_HINT_PAGINATED_TABLE;
}
```

手順 3: FooAccountReportHandler ハンドラに PaginatedReportHandler.java を拡張し、appendContextSubQuery メソッドをオーバーライドします。

- Reportcontext を使用して、コンテキスト ID を取得します。
- ReportRegistryEntry を使用して、レポートの管理列を取得します。
- QueryBuilder を使用して、クエリを形成します。

```
@Override
public Query appendContextSubQuery(ReportRegistryEntry
entry,TabularReportMetadata md, ReportContext rc, Query query)
{
      logger.info("entry.isPaginated():::::"+entry.isPaginated())
;
      String contextID = rc.getId();
      if (contextID != null && !contextID.isEmpty()) {
            String str[] = contextID.split(";");
            String accountName = str[0];
            logger.info("paginated context ID = " + contextID);
            int mgmtColIndex = entry.getManagementColumnIndex();
            logger.info("mgmtColIndex :: " + mgmtColIndex);
            ColumnDefinition[] colDefs = md.getColumns();
            ColumnDefinition mgmtCol = colDefs[mgmtColIndex];
            String colId = mgmtCol.getColumnId();
            logger.info("colId :: " + colId);
```

```
//sub query builder builds the context id sub query (e.g. id =
'xyz')
             QueryBuilder sqb = new QueryBuilder();
//sqb.putParam()
             sqb.putParam(colId).eq(accountName);
//qb ands sub query with actual query (e.g. (id = 'xyz') AND
((vmID = 36) AND //(vdc = 'someVDC')))
             if (query == null) {
//if query is null and the id field has actual value, we only
want to return //columnName = value of id
       Query q = sqb.get();
             return q;
                    } else {
                    QueryBuilder qb = new QueryBuilder();
                    qb.and(query, sqb.get());
                    return qb.get();
                    }
             } else {

                    return query;
             }
             }
```

6.7 レポートの場所の指定

UI でレポートが表示される正確な場所を指定するには、次の情報を提供する必要があります。

- UI メニューのロケーション ID
- 場所のレポート コンテキストに対応するコンテキスト マップ ルール

これらの情報を収集するには、Cisco UCS Director が提供するメタデータを使用します。メタデータには、レポートを表示したい場所に最も近いレポートのデータが含まれており、このデータを使用して必要なレポート仕様を作成できます。

レポートの場所を指定するには、次の手順を実行します。

手順 1: セッション用の開発者メニューを有効にします。

1) Cisco UCS Director で右上にあるログイン名をクリックします。
2) [ユーザ情報（User Information）] ダイアログボックスで、[詳細設定（Advanced）] タブをクリックします。
3) [[開発者]メニューの有効化(このセッション)（Enable Developer Menu (for this session)）] チェック ボックスをオンにして、[ユーザ情報（User Information）] ダイアログボックスを閉じます。

セッション中に開いたレポート ビューで [レポートメタデータ（Report Metadata）] オプションが使用可能になります。

手順 2: レポートを表示したい場所にある表形式レポートに移動して、[レポートメタデータ(Report Metadata)] をクリックし、[情報(Information)] ウィンドウを表示します。このウィンドウの上部にある [レポートコンテキスト(Report Context)] セクションを確認します。

1) **uiMenuTag** に割り当てられた整数値を探します。
   uiMenuTag はレポートの *getMenuID* が返すべき内容を示します。
2) **type** に割り当てられた値を探します。
   タイプには、コンテキスト マップ ルールを構築する必要がある UI メニューのロケーション ID が提示されており、レポートの *getMapRules* が返すべき内容が示されています。

手順 3: レポートのメタデータからコンテキスト マップを作成するのに必要なコンテキスト マップ ルールを取得します。最初の列にレポート コンテキストのタイプが示され、2 つ目の列にレポート コンテキストの名前が示されます。**type** がわかっていれば、名前を見つけることができます。たとえば、0 は「global」に対応します。両方の情報(コンテキスト名とコンテキスト タイプ)がわかっていれば、コンテキスト マップ ルールを作成できます。

手順 4: 次のサンプル コードと同様の詳細情報を使用してコンテキスト マップ ルールをインスタンス化します。

```
ContextMapRule rule = new ContextMapRule();
rule.setContextName("global");
rule.setContextType(0);

ContextMapRule[] rules = new ContextMapRule[1];
rules[0] = rule;
```

✎ 注:このサンプルでは、引数なしコンストラクタが使用されています。他のコンストラクタは使用しないでください。この場合は、引数なしコンストラクタが適切であり、これらの値は明示的に設定します。

6.8 表形式以外のレポート

表形式以外のレポートを実装するには、次の 2 つのクラスを実装します。

1) CloupiaNonTabularReport を拡張するレポート クラス
2) レポート ソース クラス

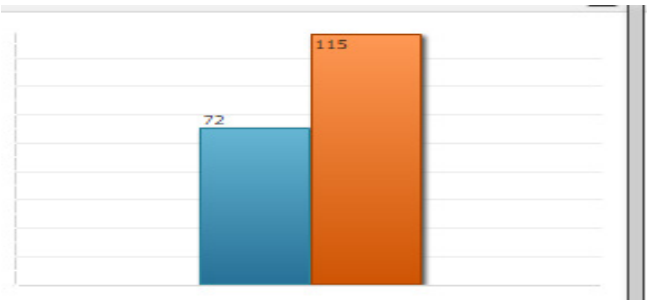
6.8.1 棒グラフ レポート

棒グラフを作成するには、次の手順を実行する必要があります。

1) CloupiaNonTabularReport クラスを拡張します。
   a) getReportType を使用して、REPORT_TYPE_SNAPSHOT を返します。
   b) getReportHint を使用して、REPORT_HINT_BARCHART を返します。

2) SnapshotReportGeneratorIf を実装します。
   a) 棒グラフ レポートの場合は、データはソース クラスによって提供することができます。getSnapshotReport メソッドをオーバーライドし、データ ソースを提供します。詳細については、次のサンプルを参照してください。

次の図は、UI の棒グラフ レポートをグラフで示しています。



サンプル：棒グラフ

```
public class SampleBarChartReportImpl implements
SnapshotReportGeneratorIf {

//In this example ,  defines the number of bars should be in
chart as bar1 nd bar2 like shown in above snapshot
       private final int NUM_BARS = 2;
       private final String BAR_1 = "bar1";
       private final String BAR_2 = "bar2";


       @Override
 public SnapshotReport getSnapshotReport(ReportRegistryEntry
reportEntry, ReportContext              context) throws Exception
{
SnapshotReport report = new SnapshotReport();
             report.setContext(context);
             report.setReportName(reportEntry.getReportLabel());
             report.setNumericalData(true);
             report.setValueAxisName("Value Axis Name");
             report.setPrecision(0);

             // setting the report name value pair for the bar
chart
             ReportNameValuePair[] rnv1 = new
ReportNameValuePair[NUM_BARS];
             rnv1[0] = new ReportNameValuePair(BAR_1, 5);
             rnv1[1] = new ReportNameValuePair(BAR_2, 10);
             // setting category of report
        SnapshotReportCategory cat1 = new
SnapshotReportCategory();
        cat1.setCategoryName("cat1");
        cat1.setNameValuePairs(rnv1);
```

```
        report.setCategories(new SnapshotReportCategory[]
{ cat1 });

            return report;
        }

}
```

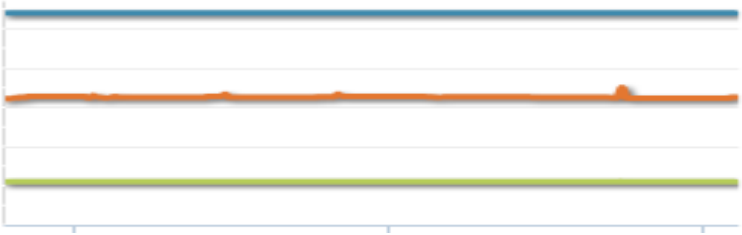
レポート クラスは CloupiaNonTabularReport を拡張し、getReportType() および getReportType() メソッドをオーバーライドしてレポートを棒グラフとして作成します。

```
public class SampleBarChartReport extends CloupiaNonTabularReport
{

private static final String NAME = "foo.dummy.bar.chart.report";
        private static final String LABEL = "Dummy Bar Chart";

// returns the implementation class
        @Override
        public Class getImplementationClass() {
            return SampleBarChartReportImpl.class;
        }

//The below two methiods are very important to shown as Bar cahrt
in the GUI.
//This method returns the report type  for bar chart shown below.
        @Override
        public int getReportType() {
            return ReportDefinition.REPORT_TYPE_SNAPSHOT;
        }

//This method returns the report hint for bar chart shown below
        @Override
        public int getReportHint()
        {
            return ReportDefinition.REPORT_HINT_BARCHART;
        }
 //bar charts will be display in summary if it returns true
        @Override
        public boolean showInSummary()
        {
            return true;
        }

}
```

UI でレポートを表示するには、システムにレポートを登録します。詳細については、「レポートの登録」の項を参照してください。

6.8.2 折れ線グラフ レポート

折れ線グラフは、次の図に示すように、レポートのグラフ表現です。



折れ線グラフ レポートを実装するには、次の手順を実行する必要があります。

1) CloupiaNonTabularReport クラスを拡張します。
   a) getReportType を使用して、REPORT_TYPE_HISTORICAL を返します。
2) HistoricalReportGeneratorIf を実装します。
   a) 折れ線グラフ レポートの場合は、データはソース クラスによって提供することができます。generateReport メソッドをオーバーライドし、データ ソースを提供します。詳細については、次のサンプルを参照してください。

サンプル：折れ線グラフ レポート

```
public class SampleLineChartReportImpl implements
HistoricalReportGeneratorIf {

       @Override
public HistoricalReport generateReport(ReportRegistryEntry
reportEntry, ReportContext        repContext,String durationName,
long fromTime, long toTime)
                    throws Exception {
              HistoricalReport report = new HistoricalReport();
        report.setContext(repContext);
        report.setFromTime(fromTime);
        report.setToTime(toTime);
        report.setDurationName(durationName);
        report.setReportName(reportEntry.getReportLabel());

        int numLines = 1;
        HistoricalDataSeries[] hdsList = new
HistoricalDataSeries[numLines];

        HistoricalDataSeries line1 = new HistoricalDataSeries();
        line1.setParamLabel("param1");
        line1.setPrecision(0);
         // createDataset1() this method use to create dataset.
        DataSample[] dataset1 = createDataset1(fromTime, toTime);
        line1.setValues(dataset1);
        hdsList[0] = line1;
        report.setSeries(hdsList);
        return report;
       }
```

```
 //implementation for method createDataset1()
private DataSample[] createDataset1(long start, long end) {
	long interval = (end - start) / 5;
	long timestamp = start;
	double yValue = 1.0;
	DataSample[] dataset = new DataSample[5];
	for (int i=0; i<dataset.length; i++) {
		DataSample data = new DataSample();
		data.setTimestamp(timestamp);
		data.setAvg(yValue);

		timestamp += interval;
		yValue += 5.0;

		dataset[i] = data;
	}
	return dataset;
}
}
```

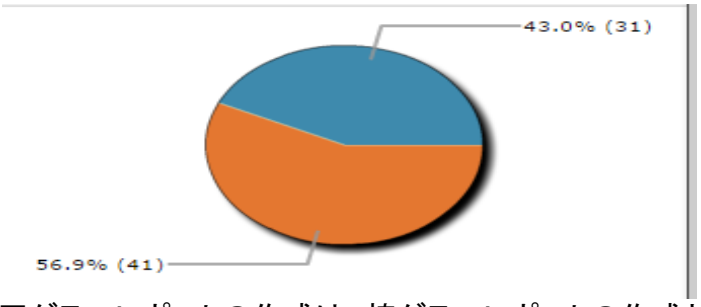
折れ線グラフレポートは CloupiaNonTabularReport クラスを拡張し、getReportType() メソッドをオーバーライドします。

```
public class SampleLineChartReport extends
CloupiaNonTabularReport {
// report name and report label is defined.
	private static final String NAME =
"foo.dummy.line.chart.report";
	private static final String LABEL = "Dummy Line Chart";
	 //Returns implementation class
	@Override
	public Class getImplementationClass() {
		return SampleLineChartReportImpl.class;
	}
	//This method returns report type as shown below
	@Override
	public int getReportType() {
		return ReportDefinition.REPORT_TYPE_HISTORICAL;
	}

}
```

UI でレポートを表示するには、システムにレポートを登録します。詳細については、「レポートの登録」の項を参照してください。

6.8.3 円グラフ レポート

円グラフは、次の図に示すように、レポートのグラフ表現です。



円グラフ レポートの作成は、棒グラフ レポートの作成と同様であり、同じ基本手順に従う必要があります。詳細については、「棒グラフ レポート」の項を参照してください。

円グラフを作成する上での重要な違いは次のとおりです。

1) CloupiaNonTabularReport クラスを拡張します。
   a) getReportType を使用して、REPORT_HINT_PIECHART を返します。

サンプル：円グラフ レポート

```
public class SamplePieChartReportImpl implements
SnapshotReportGeneratorIf {

       @Override
       public SnapshotReport getSnapshotReport(ReportRegistryEntry
reportEntry,
        ReportContext context) throws Exception {
        SnapshotReport report = new SnapshotReport();
        report.setContext(context);
        report.setReportName(reportEntry.getReportLabel());
        report.setNumericalData(true);
        report.setDisplayAsPie(true);
        report.setPrecision(0);

        //creation of report name value pair goes
        ReportNameValuePair[] rnv = new ReportNameValuePair[5];
        for (int i = 0; i < rnv.length; i++)
        {
            rnv[i] = new ReportNameValuePair("category" + i,
(i+1) * 5);
        }
        //setting of report category goes
        SnapshotReportCategory cat = new
SnapshotReportCategory();
        cat.setCategoryName("");
        cat.setNameValuePairs(rnv);
        report.setCategories(new SnapshotReportCategory[]
{ cat });
        return report;
       }

}
```

CloupiaNonTabularReport クラスを拡張し、getReportType() および getReportHint() メソッドをオーバーライドします。

```
public class SamplePieChartReport extends CloupiaNonTabularReport
{
//Returns implementation class
@Override
	public Class getImplementationClass() {
		return SamplePieChartReportImpl.class;
	}
	//Returns report type for pie chart as shown below
	@Override
	public int getReportType() {
		return ReportDefinition.REPORT_TYPE_SNAPSHOT;
	}
	//Returns report hint for pie chart as shown below
	@Override
	public int getReportHint()
	{
		return ReportDefinition.REPORT_HINT_PIECHART;
	}

}
```

UI でレポートを表示するには、システムにレポートを登録します。詳細については、「レポートの登録」の項を参照してください。

6.8.4 ヒート マップ レポート

Open Automation を使用すれば、ヒート マップなどの表形式以外のレポートを作成することができます。ヒート マップは、値をサイズや色で表したセルまたは領域を使用してデータを表現します。単純なヒート マップは情報の瞬間的概要表示を可能にします。
ここで示す手順は、3 つのセクションからなるヒート マップ レポートの作成方法を示します。3 つのセクションのそれぞれが 4 つの等しい子セクションに分割されます。ここでは、i が最大 25 のサイズを設定します。開発者は、ここに示すアプローチを拡張することにより、セクションをさらに分割することができます。
ヒート マップ レポートの作成は、単純な表形式レポートの作成と同様であり、同じ基本手順に従う必要があります。詳細については、「表形式のレポート」の項を参照してください。

ヒート マップ レポートを作成する上での重要な違いは次のとおりです。

1) CloupiaNonTabularReport を拡張します。
   a) getReportType を使用して、REPORT_TYPE_HEATMAP を返します。

サンプル:ヒート マップ レポート

```
public class DummyHeatmapReport extends CloupiaNonTabularReport
{
private static final String NAME = "foo.dummy.heatmap.report";
private static final String LABEL = "Dummy Heatmap";
@Override
                    public int getReportType() {
                           return
ReportDefinition.REPORT_TYPE_HEATMAP;
       }
}

public class DummyHeatmapReportImpl implements
HeatMapReportGeneratorIf{

@Override
       public HeatMapReport getHeatMapReportReport(ReportRegistryEntry
       reportEntry, ReportContext context) throws Exception {
              for (int i=0; i<3; i++) {
  String parentName = "parent" + i;
  HeatMapCell root = new HeatMapCell();
  root.set.Label(parentName);
  root.setUnUsedChildSize(0.0);

  //create child cells within parent cell
  HeatMapCell[] childCells = new HeatMapCell[4];
  for (int j=0; j<4; j++) {
    HeatMapCell child = new HeatMapCell();
    child.setLabel(parentName + "child" + j);
    child.stValue((j+1)*25); //sets color, the color used
//for each section is relative, there is a scale in the //UI
    child.setSize(25);  //sets weight
    childCells[j] = child;
  }
  root.setChildCells(childCells);
  cells.add(root);
}

}

}
```

6.8.5 サマリー レポート

Open Automation を使用すれば、サマリー レポートを作成することができます。サマリー レポートは、表形式以外のレポートと考えられます。これはサマリー レポートという 1 つの機能ですが、このレポートをサマリー パネルに表示するかどうかを指定できます。

サマリー レポートの作成は、単純な表形式レポートの作成と同様であり、同じ基本手順に従う必要があります。詳細については、「表形式のレポート」の項を参照してください。

サマリー レポートを作成する上での重要な違いは次のとおりです。

1) CloupiaNonTabularReport を拡張します。
   a) getReportType() をオーバーライドして、REPORT_TYPE_SUMMARY を返します。
   b) getReportHint() をオーバーライドして、REPORT_HINT_VERTICAL_TABLE_WITH_GRAPHS を返します。
   c) isManagementReport が True を返すように設定します。

サンプル：サマリー レポート

```
public class DummySummaryReport extends CloupiaNonTabularReport
{

      private static final string NAME =
"foo.dummy.summary.report";
      private static final string LABEL = "Dummy Summary";

      //Override getReportType() and getReportHint(), using this
code snippet:
@Override
public int getReportType()
{
    return ReportDefinition.REPORT._TYPE_SUMMARY;
}

/**
* @return report hint
*/
@Override
public int getReportHint()
{
    return
ReportDefiniton.REPORT_HINT_VERTICAL_TABLE_WITH_GRAPHS;
}
@Override
      public boolean isManagementReport()
      {
            return true;
}
}

public class DummySummaryReportImpl implements
TabularReportGeneratorIf{

@Override
public TabularReport getTabularReportReport(ReportRegistryEntry
reportEntry, ReportContext context) throws Exception {
TabularReport report = new TabularReport();
```

```
  report.setContext(context);
 report.setGeneratedTime(System.currentTimeMillis());
 report.setReportName(reportEntry.getReportLabel());

 //showing how to add two tables to your summary panel
 //the tables in summary panel are always two column tables
 SummaryReportInternalModel model = new
SummaryReportInternalModel();

 //this will be my first table
 //two rows, column one and column two values shown, note the
last value
 //this is how you group what data is grouped together
        model.addText("table one key one", "table one property
one", DUMMY_TABLE_ONE);
        model.addText("table one key two", "table one property
two", DUMMY_TABLE_ONE);

        model.addText("table two key one", "table two property
one", DUMMY_TABLE_TWO);
        model.addText("table two key two", "table two property
two", DUMMY_TABLE_TWO);

        //you'll notice the charts that show in summary panels
aren't mentioned here
        //that's because you'll need to specify in the chart
report whether or not the
        //chart should be displayed in the summary panel or not,
look in the BarChartReport
        //example for more detail

        //finally perform last clean up steps
        model.setGroupOrder(GROUP_ORDER);
        model.updateReport(report);

        return report;
}
}
```

7 新メニューの開発

新しいメニュー項目を作成するには、次のような一連の関連作業が必要です。

- メニューを定義します。
- 新しいメニューを登録します。
- 新しいレポート コンテキストを登録します。
- 必要に応じて、左側のナビゲーション ツリー プロバイダーを作成します。
- レポートを作成して、新しいメニューの場所が適切なコンテキスト マップ ルールを使用して指定されていることを確認してください。

7.1 メニュー項目の定義

Open Automation を使用して新しいメニューを追加する唯一の方法は、menu.xml という名前のファイルを指定することです。この項で提示するコード例では、メニュー項目を定義するタスクに関するガイダンスを提供しています。
次のサンプル コードは、Open Automation SDK サンプルに含まれています。XML サンプルはメニュー項目を導入するための次の 2 つのオプションを表示します。

- 新しいメニュー項目を既存のフォルダ（Virtual など）に挿入する方法
- まったく新しいメニュー項目を UI に追加する方法

すべてのメニュー ノードに一意の整数 ID が関連付けられます。新しいメニュー項目を導入する場合は、表に示すように、次のメニュー ID が予約されていることに注意してください。

メニュー	ID
仮想	1000
物理	1001
組織	1002
ポリシー	1003
仮想/ハイパーバイザ ポリシー	1007
物理インフラストラクチャ ポリシー	1008
管理（Administration）	1004

メニュー項目のコンポーネントは次のとおりです。

- label：メニューの UI に表示するラベル。
- path：メニューに移動するときに URL で使用される値。

> ✐ 重要：この文字列の最後にバックスラッシュを含める必要があります。たとえば、dummy_menu_1/ or dummy_menu_2/ などです。

- op：このメニューにアクセスするためにユーザが持っていなければならない権限。no_check は誰でもアクセスできることを意味します。
- url：これは常に modules/GenericModules.swf でなければなりません。このフィールドは、メニュー項目であるがカテゴリではない場合にのみ入力する必要があります。
- leftNavType：有効な値は none または backend_provided です。lefNavType の詳細については、「メニュー ナビゲーションの定義」の項を参照してください。このフィールドは、メニュー項目であるがカテゴリではない場合にのみ入力する必要があります。

- children：メニューにサブ メニューがある場合、それらをここに追加する必要があります。ベスト プラクティスは、メニュー内に 4 つ以上のレベルを作成しないことです。

前述したメニュー項目のコンポーネントのサンプル値をサンプルコードで示します。

オプション 1：既存のフォルダの下に新しいメニュー項目を追加します。
たとえば、Virtual 呼ばれる既存のフォルダの下に新しいメニュー項目を追加するには、項目を追加するメニュー カテゴリの ID を検索する必要があります。Virtual フォルダの ID は 1000 です。記入したメニュー ID を持つ親メニュー項目をメモします。これは、既存のカテゴリにメニュー項目を入れていることを示すために必要なものです。新しいメニュー項目が子フィールドに配置されます。

サンプル

```
<menu>
       <!-- this shows you how to add a new menu item underneath
virtual -->
       <menuitem>
       <menuid>1000</menuid>
       <children>
              <menuitem>
                     <menuid>12000</menuid>
                     <label>Dummy Menu 1</label>
                     <path>dummy_menu_1/</path>
                     <op>no_check</op>
                     <url>modules/GenericModule.swf</url>
                     <leftNavType>backend_provided</leftNavType>
              </menuitem>
       </children>
</menuitem>
```

オプション 2：まったく新しいメニュー項目を UI に追加します。
まったく新しいメニュー項目を定義する場合は、サンプルに示すように、すべての詳細を指定します。メニュー カテゴリに関するすべての詳細を指定して、その下にすべての子メニュー項目を追加します。次の例は、メニューを 2 レベル下まで表示していますが、理論上は、必要な深さにすることができます。ベスト プラクティスは、3 レベル以下の深さのメニューを作成することです。

サンプル

```
<!-- entirely new menu -->
<menu>
  <menuitem>
          <menuid>11000</menuid>
          <label>Sample Category</label>
       <path>sample/</path>
                <op>no_check</op>
```

```
                    <children>
                            <menuitem>
                                    <menuid>11001</menuid>
                                    <label>Sample Menu 1</label>
                                    <path>Sample_menu_1/</path>
                                    <op>no_check</op>

<url>modules/GenericModule.swf</url>

<leftNavType>backend_provided</leftNavType>
                            </menuitem>
                    </children>
            </menuitem>
<menu>
```

7.2 メニュー項目の登録

Open Automation では、メニュー登録が自動的に処理されます。開発者が行う必要があるのは、メニューの xml ファイルを *menu.xml* として指定し、モジュールの一部として xml ファイルをパッケージすることだけです。menu.xml ファイルがモジュール jar ファイルの最上位レベルにあることを確認してください。

7.3 メニュー ナビゲーションの定義

Cisco UCS Director は、メニュー ナビゲーションを使用して UI に表示するレポートとフォームを決定します。レポートの場所に関する詳細については、「レポートの場所の指定」の項を参照してください。

leftNavType フィールドは、メニュー項目に使用するナビゲーションのタイプを指定します。

値 **none** の意味は次のとおりです。

- ナビゲーションが必要ない。
- メニュー項目に関連付けられたコンテキスト マップ ルールが type = 10、name = "global_admin" である。（重要）

leftNavType が **backend_provided** の場合は、左側のナビゲーション ツリーを生成する **com.cloupia.model.cIM.AbstractTreeNodesProviderIf** の実装を指定する必要があります。

ナビゲーション ツリーの各ノードは次の要素を提供する必要があります。

- ラベル（Label）
- UI に表示するアイコンへのパス（任意）
- コンテキスト タイプ（詳細については、『Cisco UCS Director Open Automation Developer Guide』のレポート コンテキストの登録の項を参照してください）。
- コンテキスト ID（これが表の生成時に使用されるレポート コンテキスト ID になる）

ナビゲーション ツリーはメニュー ID に関連付ける必要があるため、ツリー プロバイダーの登録時には必ず、対応するメニュー ID を使用してください。

仮想アカウント用のシステム メニュー ID	
メニュー	**ID**
コンピューティング	0
格納形式	1
ネットワーク	2
物理アカウント用のシステム メニュー ID	
メニュー	**ID**
コンピューティング	50
格納形式	51
ネットワーク	52

8 トリガー条件の開発

特定の目的に使用するトリガーを作成するには、正しく定義されたトリガー条件を準備する必要があります。適切なトリガー条件がまだ存在しない場合は、それを実装する必要があります。適切で必要な条件の構成要素がまだ定義されていない場合は、ここに示す情報を使用してそれらを実装することができます。

[トリガーの作成（Create Trigger）] ウィザード（[ポリシー（Policies）] > [オーケストレーション（Orchestration）] > [トリガー（Triggers）]）の [条件を指定します（Specify Conditions）] 画面で、新しいトリガー条件を設定するときに使用可能なオプションを設定する必要があります。

トリガーは次の 2 つのコンポーネントから構成されます。

- com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredContextIf の実装。
- com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredParameterIf の 1 つ以上の実装。

MonitoredContextIf は、モニタされオブジェクトへの参照のリストを提供するオブジェクトを記述します。トリガーを選択し、[トリガーの編集（Edit Trigger）] をクリックし、トリガー編集ウィザードの [条件を指定します（Specify Conditions）] 画面に移動します。コントロールと関連オプションを使用して、オブジェクトとそのオブジェクトへの参照を選択します。たとえば、MonitoredContextIf を使用して、ダミー デバイスオブジェクトを監視し、使用可能なすべてのダミー デバイスのリストを返すことができます。

MonitoredParamterIf は、次のようなトリガー条件の定義で使用されます。

- 検査される特定のパラメータを提供するため。たとえば、MonitorContextIf で定義した特定のダミー デバイス（ddTwo など）のステータスを表すパラメータなどです。

- パラメータに適用できる演算を提供するため。一般的な演算には以下が含まれます。
  - より少ない
  - 次の値と等しい
  - より大きい

  （適切な演算は、実装によって異なります）。
- 値のリストを提供するには、その各値とパラメータを論理的に比較してトリガーをアクティブにします。

たとえば、**Dummy Device ddTwo Status is down** などのトリガー条件は 1 つの条件として論理的にテストすることができます。監視対象の Status パラメータがステートメント True を意味する場合に、トリガー条件が満たされます。

8.1 新しいトリガー条件の追加

新しいトリガー条件を追加するには、次の手順に従います。

1. MonitoredContextIf とすべての適用可能な MonitoredParameterIf を実装します。
2. システムにトリガー条件を登録します。

サンプル

手順 1：MonitoredContextIf とすべての適用可能な MonitoredParameterIf を実装します。

```
public class MonitorDummyDeviceStatusParam implements
MonitoredParameterIf {

	@Override
	public String getParamLabel() {
		//this is the label of this parameter shown in the
ui
		return "Dummy Device Status";
	}

	@Override
	public String getParamName() {
//each parameter needs a unique string, it's a good idea to
//prefix each parameter
//with your module id, this way it basically guarantees
//uniqueness
		return "foo.dummy.device.status";
	}

	@Override
	public FormLOVPair[] getSupportedOps() {
//this should return all the supported operations that can be
//applied to this parameter
		FormLOVPair isOp = new FormLOVPair("is", "is");
```

```
            FormLOVPair[] ops = { isOp };
            return ops;
      }

      @Override
      public int getValueConstraintType() {
            return 0;
      }

      @Override
      public FormLOVPair[] getValueLOVs() {
//this should return all the values you want to compare against
//e.g. threshold values
            FormLOVPair valueUP = new FormLOVPair("Up", "up");
            FormLOVPair valueDOWN = new FormLOVPair("Down",
"down");
            FormLOVPair valueUNKNOWN = new
FormLOVPair("Unknown", "unknown");
            FormLOVPair[] statuses = { valueDOWN, valueUNKNOWN,
valueUP };
            return statuses;
      }

      @Override
      public int getApplicableContextType() {
//this parameter is binded to MonitorDummyDeviceType, so it needs
//to return the same
            //value returned by
MonitorDummyDeviceType.getContextType()
            DynReportContext dummyContextOneType =
ReportContextRegistry.getInstance().getContextByName(FooConstants
.DUMMY_CONTEXT_ONE);
            return dummyContextOneType.getType();
      }

      @Override
      public String getApplicableCloudType() {
            return null;
      }

      @Override
      public int checkTrigger(StringBuffer messageBuf, int
contextType,
                  String objects, String param, String op,
String values) {
//you want to basically do if (objects.param op values)
{ //activate } else { not activate }

            //first step, you'd look up what objects is pointing
to, usually objects should be an identifier
            //for some other object you actually want
            //in this example, objects is either ddOne (dummy
device) or ddTwo, for simplicity's sake, we'll
            //say ddOne is always up and ddTwo is always down
```

```
            if (objects.equals("ddOne")) {
                  if (op.equals("is")) {
                        //ddOne is always up, so trigger only
gets activated when "ddOne is up"
                        if (values.equals("up")) {
                              return
RULE_CHECK_TRIGGER_ACTIVATED;
                        } else {
                              return
RULE_CHECK_TRIGGER_NOT_ACTIVATED;
                        }
                  } else {
                        return RULE_CHECK_ERROR;
                  }
            } else {
                  if (op.equals("is")) {
                        //ddTwo is always down, so trigger only
gets activated when "ddTwo is not up"
                        if (values.equals("up")) {
                              return
RULE_CHECK_TRIGGER_NOT_ACTIVATED;
                        } else {
                              return
RULE_CHECK_TRIGGER_ACTIVATED;
                        }
                  } else {
                        return RULE_CHECK_ERROR;
                  }
            }
      }

}


public class MonitorDummyDeviceType implements MonitoredContextIf
{

      @Override
      public int getContextType() {
            //each monitored type is uniquely identified by an
integer
            //we usually use the report context type
            DynReportContext dummyContextOneType =
ReportContextRegistry.getInstance().getContextByName(FooConstants
.DUMMY_CONTEXT_ONE);
            return dummyContextOneType.getType();
      }

      @Override
      public String getContextLabel() {
            //this is the label shown in the ui
            return "Dummy Device";
      }
```

```
        @Override
        public FormLOVPair[] getPossibleLOVs(WizardSession session)
{
                //this should return all the dummy devices that
could potentially be monitored
                //in this example i only have two dummy devices,
usually the value should be an identifier you can use
                //to reference back to the actual object
                FormLOVPair deviceOne = new FormLOVPair("ddOne",
"ddOne");
                FormLOVPair deviceTwo = new FormLOVPair("ddTwo",
"ddTwo");
                FormLOVPair[] dummyDevices = { deviceOne,
deviceTwo };
                return dummyDevices;
        }

        @Override
        public String getContextValueDetail(String
selectedContextValue) {
                //this is additional info to display in the ui, i'm
just returning a dummy string
                return "you picked " + selectedContextValue;
        }

        @Override
        public String getCloudType(String selectedContextValue) {
                // TODO Auto-generated method stub
                return null;
        }

}
```

手順 2:システムにトリガー条件を登録します。

```
// adding new monitoring trigger, note, these new trigger
components
// utilize the dummy context one i've just registered
// you have to make sure to register contexts before you execute
// this code, otherwise it won't work
MonitoringTrigger monTrigger = new MonitoringTrigger(
new MonitorDummyDeviceType(),new
MonitorDummyDeviceStatusParam());
MonitoringTriggerUtil.register(monTrigger);
menuProvider.registerWithProvider();
```

9 CloudSense レポートの開発

各 CloudSense レポートは、次の 2 つのコンポーネントで構成されます。

- XML ディスクリプタ ファイル
- JavaScript によるレポートの実際の実装

XML ディスクリプタ ファイルには次のものが必要です。

- レポートを識別する一意の文字列
- UI への表示に使用するレポートの説明
- レポートの実装ファイルの名前
- レポートの生成時に使用する必要があるコンテキスト

JavaScript による CloudSense レポートの実装については、Cisco UCS Director のマニュアルに記載されています。

CloudSense レポートに関連するすべてのファイルは、cloudsense というフォルダに配置する必要があります。Open Automation を使用してアップロードする zip ファイルには、cloudsense フォルダが含まれている必要があります。フォルダ構造に関する詳細については、サンプル モジュールを参照してください。

10 アカウントの追加

システムにインベントリ収集のコレクタを検索させるようにするには、Cisco UCS Director でアカウントを追加します。アカウントを追加するには、特定のアカウントの AccountTypeEntry を参照して、新しいアカウント タイプとその詳細を定義します。レポートを配置する UI に一意のアカウント タイプ、アカウント カテゴリおよび場所を指定する必要があります。

サンプル アカウントの追加には、次のタスクが含まれます。

1. クレデンシャル クラスの作成
2. JDO 機能拡張のためのアカウントの登録
3. モジュールでのアカウントの登録

ユーザからデータを収集するための POJO クラスの作成

ユーザからデータを収集するための POJO クラスで入力を定義します。これは持続可能でなければならず、レポートする入力を読み込む必要があります。アカウントの追加時は、次のサンプルに示すように、適切な注釈で入力フィールドに注釈を付けます。

```
@persistent //this is used to persistence
@ReportField(label="Account Name") //this is used to populate the
inputs to report in the UI.
```

✎ 注：このサンプルは、ダミーのクラスからデータを収集します。

サンプル

```
@PersistenceCapable(detachable = "true")
public class DummyAccount {

        @ReportField(label="Account Name")
        @Persistent
        private String accountName;
```

```
        @ReportField(label="Status")
        @Persistent
        private String status;

        @ReportField(label="IP Address")
        @Persistent
        private String ip;
}
```

10.1 クレデンシャル クラスの作成

ConnectorCredential インターフェイスを実装する AbstractInfraAccount を拡張するクレデンシャル クラスを作成します。

ConnectorCredential インターフェイスの実装時に、InfraAccount toInfraAccount() メソッドをオーバーライドします。

サンプル

```
public class FooAccount extends AbstractInfraAccount implements
ConnectorCredential{
        //Pojo can implement for the account creation
        @Persistent
        @FormField(label = "Device IP", help = "Device IP",
mandatory = true)
        private String deviceIp;


        @Persistent
@FormField(label = "Protocol", help = "Protocol", type =
FormFieldDefinition.FIELD_TYPE_EMBEDDED_LOV, validate = true, lov
= {
                        "http", "https" })
        private String protocol="http";

@FormField(label = "Port", help = "Port Number", type =
FormFieldDefinition.FIELD_TYPE_TEXT)
        @Persistent
        private String port = "8080";

        @Persistent
        @FormField(label = "Login", help = "Login")
        private String login;

        @Persistent
@FormField(label = "Password", help = "Password", type =
FormFieldDefinition.FIELD_TYPE_PASSWORD)
                private String password;
        /**
         * @return the deviceIp
         */
        public String getDeviceIp() {
                return deviceIp;
        }
```

```
        /**
         * @param deviceIp
         *            the deviceIp to set
         */
        public void setDeviceIp(String deviceIp) {
               this.deviceIp = deviceIp;
        }

/**
         * @return the protocol
         */
        public String getProtocol() {
               return protocol;
        }

        /**
         * @param protocol
         *            the protocol to set
         */
        public void setProtocol(String protocol) {
               this.protocol = protocol;
        }

        /**
         * @return the port
         */
        public String getPort() {
               return port;
        }

        /**
         * @param port
         *            the port to set
         */
        public void setPort(String port) {
               this.port = port;
        }

        /**
         * @return the login
         */
        public String getLogin() {
               return login;
        }

        /**
         * @param login
         *            the login to set
         */
        public void setLogin(String login) {
               this.login = login;
        }

        /**
```

```
	 * @return the password
	 */
	public String getPassword() {
		return password;
	}

	/**
	 * @param password
	 *            the password to set
	 */
	public void setPassword(String password) {
		this.password = password;
	}

@Override
public InfraAccount toInfraAccount() {
		try {
			ObjStore<InfraAccount> store = ObjStoreHelper
				.getStore(InfraAccount.class);
			String cquery = "server == '"
				+ deviceIp + "' && userID == '"
+ login
				+ "' && transport == '" +
protocol + "' && port == "
				+ Integer.parseInt(port);
			logger.debug("query = " + cquery);

			List<InfraAccount> accList =
store.query(cquery);
			if (accList != null && accList.size() > 0)
				return accList.get(0);
			else
				return null;
		} catch (Exception e) {
logger.error("Exception while mapping DeviceCredential to
InfraAccount for server: "+ deviceIp + ": " + e.getMessage());
		}

		return null;
	}

}
```

10.2 JDO 機能拡張のためのアカウントの登録

POJO とクレデンシャル クラスを実装した後、Jdo 機能拡張用のアカウントを登録します。詳細については、『持続化オブジェクトとしてのクラスのマーキング』を参照してください。

10.3 モジュールでのアカウントの登録

アカウントの実装後、FooModule クラスでアカウントを登録します。

サンプル

```
private void createAccountType(){
AccountTypeEntry entry=new AccountTypeEntry();
// setting the credential class that holds the device credential
details
entry.setCredentialClass(FooAccount.class);
//There are three category of account type compute,storage and
network. we are
//setting the category as Storage shown below
entry.setCategory(InfraAccountTypes.CAT_STORAGE);

// we are setting type of account. There are two types of account
available virtual
//account and physical account.
entry.setAccountClass(AccountTypeEntry.PHYSICAL_ACCOUNT);

// we are setting the connection handler. Before adding account
this checks
//whether device is reachable or not by pinging the device ip.
entry.setTestConnectionHandler(new FooTestConnectionHandler());

//according to account type we developed inventory listener for
collecting credential
//details
entry.setInventoryListener(new FooInventoryListener());
//Registring of inventory object goes
registerInventoryObjects(entry);
//Registering of an account goes .
PhysicalAccountTypeManager.getInstance().addNewAccountType(entry)
;
}
```

11 アカウントのインベントリ収集

項目のインベントリを収集するには、FooModule クラスでルート名とそのハンドラクラスを登録します。

サンプル

```
//Registering in FooModule clas
private void createAccountType(){
try {
       ...
       // Adding inventory for root objects
       registerInventoryObjects(entry);

       PhysicalAccountTypeManager.getInstance().addNewAccountType(
entry);
}catch(Exception e){
}
```

RegisterInventoryObjects メソッドは、ルートまたは一意の項目のインベントリを収集するために使用されます。収集されたインベントリの項目ハンドラをパーサーに呼び出し、UI にデータを入力するためのオブジェクトへの応答をバインドします。

```
private void registerInventoryObjects(AccountTypeEntry
fooRecoverPointAccountEntry) {
ConfigItemDef fooRecoverPointStateInfo =
fooRecoverPointAccountEntry
.createInventoryRoot("foo.inventory.root",FooInventoryItemHandler
.class);
}

public class FooInventoryItemHandler extends
AbstractInventoryItemHandler {

	private static Logger logger =
Logger.getLogger(FooInventoryItemHandler.class);
	//this method use for clean up operation
	@Override
	public void cleanup(String accountName) throws Exception {
		// cleanup implementation goes .

	}

	@Override
	public void doInventory(String accountName,
InventoryContext inventoryCtxt)
			throws Exception {
		//inventory implementation goes
	}
```