

# Make 処理のカスタマイズ

---

この機能は、Release10.0 以降で利用可能です。

J2EE に精通した開発者にとっては、NXJ によって生成されたアプリケーションアーカイブに変更を行なう必要がある場合もあります。その場合、ejb と Web アプリケーションファイル構造にファイルの追加や再配置を行なって、生成された構成ファイル（例：ejb-jar.xml, web.xml, その他）を変更します。

このドキュメントでは、必要な変更を作成するために Make Hook を記述して、Make 処理をカスタマイズする方法を説明します。この Java クラスは、実行コマンドの Make フェーズ中に NXJ によって呼び出されます。

---

**警告** – NXJ は、このファイルに行う変更に対して検証を行わないので、アプリケーションサーバや、NXJ コントロールセンタと互換性のないアーカイブに変更することが可能です。

---

## Make Hook の作成方法

Make Hook を作成する場合は、**com.unify.nxj.ide.make.NXJMakeHook** インタフェースを実装する Java クラスを記述します。このインタフェースは、以下の 1 つのメソッドを宣言します。

```
public void makeHook(  
    java.io.File earContentsDir,  
    java.util.Properties properties,  
    com.unify.nxj.ide.make.NXJMakeHookCallback callback  
    ) throws Exception;
```

---

NXJ が Make Hook を使用する構成では、クラス（クラスは 0 引数構造を持つ必要があります）のインスタンスを作成します。その後、makeHook メソッドが呼び出されます。makeHook メソッドがエラー無しで終了した場合、NXJ は適切なアーカイブ中にファイルをパッケージして、アプリケーションを配備します（アプリケーションを配備または実行している場合）。

makeHook メソッドは、3 つの引数を渡されます。

- java.io.File オブジェクト

最初の引数は、NXJ がパッケージされるファイルをアセンブルしたディレクトリを示す java.io.File オブジェクトです。このディレクトリは、3 つのその他のディレクトリがあります。META-INF ディレクトリには、J2EE アプリケーション用のディスクリプタである application.xml ファイルがあります。jar-contents ディレクトリには、アプリケーションの EJB 部分のファイルがあります。EJB ディスクリプタは、jar-contents/META-INF/ejb-jar.xml です。war-contents ディレクトリには、アプリケーションの Web アプリケーション部分のファイルがあります。Web アプリケーションディスクリプタは、war-contents/META-INF/web.xml です。

---

**注** – ear-contents ディレクトリ構造は、NXJ の今後のリリースでは変更の予定がありません。

---

- java.util.Properties オブジェクト

makeHook メソッドに渡される 2 つ目の引数は、java.util.Properties オブジェクトです。このオブジェクトは、Make Hook を構成するときに定義されたプロパティを含みます。（下の Make Hook の構成を参照してください。）

- com.unify.ide.nxj.make.NXJMakeHookCallback オブジェクト

makeHook に渡される最後の引数は、com.unify.ide.nxj.make.NXJMakeHookCallback オブジェクトです。このオブジェクトでは、アプリケーションデザイナーのコンパイラウィンドウでメッセージを表示可能な 3 つのメソッドがあります。

```
public void displayMessage(String msg);
public void displayWarning(String msg);
public void displayError(String msg);
```

---

最初の2つのメソッドは、コンパイラウィンドウにメッセージを表示して処理を続けます。表示エラーにより表示されるメッセージは、コンパイラウィンドウに赤で表示されます。makeHook メソッドによってエラーが表示されると、NXJ は makeHook メソッドの処理を中止して戻ります。

## サンプル Make Hook

下記のコードは、EJB ディスクリプタファイルに環境エントリを追加する Make Hook の例です。この例では、一行ずつファイルを読み込んで、ファイルの新しいバージョンに書き込む作業をします。ファイルの適切な位置に達すると、プロパティオブジェクトで渡される各プロパティの <env-entry> 宣言をすべて書き出します。

プロパティ名は、エントリの名前として使用されます。プロパティ値は、エントリに割り当てる値を含む文字列となります。また、値の後にセミコロンを加えることで、エントリのタイプとエントリ（デフォルトは、java.lang.String）のクラスタイプを宣言することができます。最後に、2つ目のセミコロンと説明を追加することができます。タイプと説明はオプションです。

その後、Make Hook は、ファイルの残りの書き出しを継続して終了します。

```
import java.io.*;
import java.util.*;
import com.unify.nxj.ide.make.*;

public class EJBEnvHook implements NXJMakeHook
{
    public void makeHook ( File earContentsDir,
        Properties props, NXJMakeHookCallback callback)
    {
        File orig = new File( earContentsDir.getPath() +
            "/jar-contents/META-INF/ejb-jar.xml" );
        if ( orig.exists() )
        {
            File backup = new File( orig.getPath() + ".bu" );
            if ( !orig.renameTo( backup ) )
            {
                callback.displayWarning("Cannot rename the xml file. Make will
                    continue with the unmodified file.");
            }
        }

        BufferedReader in = null;
```

---

```

BufferedWriter out = null;

try
{
    in = new BufferedReader( new FileReader( backup ) );
    out = new BufferedWriter( new FileWriter( orig ) );
    String nextLine = null;
    boolean inBean = false;
    while ( (nextLine = in.readLine()) != null )
    {
        if ( inBean )
        {
            // Add our entries at the end of the declaration
            // for the bean
            if ( nextLine.indexOf("</session>") > 0 )
            {
                Enumeration propNameNames = props.propertyNames();
                while ( propNameNames.hasMoreElements() )
                {
                    String propName = (String)propNameNames.nextElement();
                    callback.displayMessage("Adding Env Entry: " + propName);
                    if ( propName.equals("test") )
                        throw new Exception("this is a test");
                    out.write("<env-entry>%n");
                    out.write("<env-entry-name>" + propName +
                        "</env-entry-name>%n");
                    String property = (String)props.getProperty(propName);
                    StringTokenizer prop = new StringTokenizer(property, ";");
                    out.write("<env-entry-value>" + prop.nextToken() +
                        "</env-entry-value>%n");
                    if ( prop.hasMoreTokens() )
                        out.write("<env-entry-type>" + prop.nextToken() +
                            "</env-entry-type>%n");
                    else
                        out.write("<env-entry-type>java.lang.String</env-entry-
                            type>%n");
                    if ( prop.hasMoreTokens() )
                        out.write("<description>" + prop.nextToken() +
                            "</description>%n" );
                    out.write("</env-entry>%n");
                }
                inBean = false;
            }
        } else {
            // This line indicates the start of the <session> bean

```

---

```

        // we want to modify
        if ( nextLine.indexOf("nxjFieldManager_ID") > 0 )
            inBean = true;
    }
    out.write(nextLine);
    out.newLine();
} // end while
} catch ( Exception e ) {
    callback.displayError(
        "An error occurred while executing the make hook: " +
        e.getMessage());
    callback.displayError("Make process will now stop.");
} finally {
    try {
        out.flush();
        out.close();
        in.close();
        backup.delete();
    } catch ( Exception e2 ) {}
}
}
} // end makeHook
}

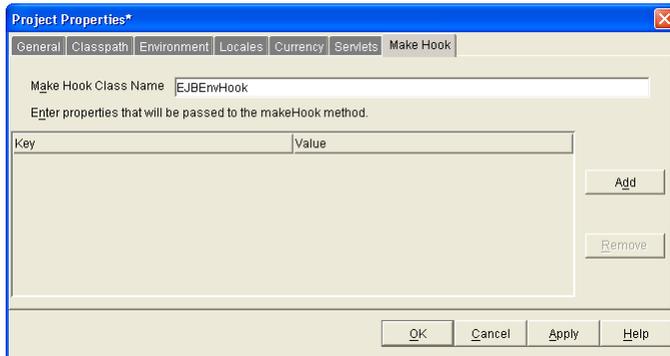
```

## Make Hook の構成

Make Hook を作成した場合、プロジェクトのクラスパスにそれを追加しなければなりません。Java ファイルがプロジェクトの一部である場合、デフォルトでクラスはクラスパス上にありますが、アプリケーションを構築するとき、クラスはアプリケーションアーカイブにも含められます。そのため、Make Hook の作成が終わったら、Jar ファイル中に Make Hook をパッケージして、**プロジェクト > プロパティ > クラスパスパネル** でクラスパスエントリとして追加することを推奨します。

クラスがアーカイブに含められないように、‘EAR に含める’チェックボックスをクリアにします。

Make Hook は、**プロジェクト > プロパティ > Make Hook** パネルで構成します。Make Hook クラス名は、Make Hook Class 固有の名前です。この例では、‘EJBEnvHook’です。



新しいエントリを追加するには **追加** ボタンをクリックして、ダイアログで必要数のプロパティを追加します。サンプル Make Hook がこれらのプロパティを使用する方法については、上記の説明を参照してください。

実行する場合は **OK** ボタンをクリックします。

## Make Hook による動作

サンプル Make Hook を試行する場合は、EJB 環境エントリを取得するために標準の J2EE メソッドを使用するフォームを作成します。プロジェクトにフォームを追加して、フォームにテキストフィールドを配置します。テキストフィールド名を 'envEntry' に変更します。フォームにエントリポイントを追加します。フォームの BEFORE FORM セクションでコードを追加します。'Test' という String プロパティでサンプル Make Hook を構成する場合、以下のコードで取得することができます。

```
BEFORE FORM
{
    Context jndiCtx = new InitialContext();
    Object obj = jndiCtx.lookup("java:comp/env/Test");
    String value = (String)PortableRemoteObject(obj,
        java.lang.String.class);
    envEntry = value;
}
```

---

アプリケーションを実行します。NXJ がアプリケーションを構築中に、各プロパティのためにコンパイラウィンドウに表示される ‘Adding Env Entry: Test’ というメッセージを確認する必要があります。アプリケーションの実行時、フォームは表示され、そのテキストフィールドは、Make Hook を構成したときに設定した値を含んでいます。