

Build4Xpages

# Table of Contents

Getting Started .....	1
Tutorials .....	1
Building an NSF .....	2
BuildXPages Ant Tasks .....	10
NSF Deployment Related Tasks .....	10
Plugin Related Tasks .....	14
Designer Related Tasks .....	18
Server Related Ant Tasks .....	22

Build4XPages is a collection of tools which aid in the Building and Deployment of XPages NSFs and XPages plugins.

## Getting Started

In order to use the BuildXPages there is a little bit of setup to be done. This section will hopefully guide you through the process. Please let me know via an Issue on the Github Project page if anything was left out.

### First Time Setup Tasks

- Install IBM Notes/Designer (This is assumed!)
- Install a JDK
- Install Ant
- Install the BuildXPages ANT Task Library
- Install Eclipse
- Setup Environment Variables
- Install the Headless Designer Plugin to Notes/Designer
- Remove the Password from the Notes ID

## Tutorials

These Tutorials are designed to demonstrate the different tasks that are available. It is intended that these tutorials can be used to verify that everything is working correctly, and you can then take the build files for these tutorials and customise them for your own projects.

# Building an NSF

## IMPORTANT

Before doing this tutorial make sure you have performed necessary setup detailed in the **gettingstarted** page

In this tutorial we will build a Simple XPages Application from an on On-Disk Project, using Domino Designer Headless Plugin + BuildXPages Ant Tasks.

After the successful build, the nsf will be available in your local workspace.

The files for this Demonstration are in the **demo/buildnsf** folder of the Build4XPages distribution files. The sample On-Disk Project is located within the tutorial folder as **sampleodp**. The Application has a Custom Control and an XPage which uses that custom control.

## build.xml

Our 'recipe' for the build is contained in the **build.xml** file located in the **demo/buildnsf** directory. **build.xml** is the default name for an Ant build file, you can call it another filename but if so you would need to specify extra command line argument **-b myfile.xml**

Within the build.xml file are things called *targets*. A target is a bit like a code block. Targets can be configured to depend on another target, so that means executing a *target* will cause the other targets that it depends on to be executed first.

Within this build.xml file are 2 *targets* that are used to setup the demo **clean** and **init**

**clean** is used to clean up any previous attempt at this tutorial, and **init** is used to initialise a new attempt.

You can see that **init** has been set to depend on **clean** so that before the **init** target is executed, the **clean** target will be executed.

```

<target name="clean">
    <delete dir="odp"/>
    <delete dir="odperror"/>
</target>

<target name="init" depends="clean">

    <copy todir="odp">
        <fileset dir="sampleodp"></fileset>
    </copy>

    <copy todir="odperror">
        <fileset dir="sampleodp"></fileset>
    </copy>

    <!-- Delete the Custom Control from the odp that we are using for the error
demo -->
    <delete>
        <fileset dir="odperror/CustomControls"></fileset>
    </delete>

</target>

```

## Run a Successful Build

Within this **build.xml** file is a target called **build**. This will build the NSF that should not have any problems. This target is listed as the default target in the root element of the build.xml file (not shown here).

This target makes use of the **bxp:buildnsf** task which will connect to the Headless Designer Plugin, and instruct it to build the nsf as specified by the task attributes.

```

<target name="build" depends="init">

    <bxp:checkDesignerRunning/>

    <!-- We assign the .project file path to a property using the 'location'
attribute,
this way the property will be expanded to the absolute path of the .project
file-->
    <property name="odp.projectfile" location="odp/.project" />

    <bxp:buildnsf project="${odp.projectfile}" projectname="${odp.projectname}"
nsf="${nsf.filepath}" />

</target>

```

You can see that this target depends on **init**. **init** depends on **clean**, so when we execute this **build**

target what will really happen is the following order of execution:

```
clean -> init -> build
```

## To run the Build

1. Open your command line
2. Change directory to the the **demo/buildnsf** folder
3. issue the command **ant**

Ant knows to look for **build.xml** and the default target (which is our **build** target). The build process will then begin, and your nsf will (hopefully) be built into your local workspace.

You should see some output on the console like so:

```
Buildfile: V:\Projects\BuildXPages\demo\buildnsf\build.xml

clean:
  [delete] Deleting directory V:\Projects\BuildXPages\demo\buildnsf\odp
  [delete] Deleting directory V:\Projects\BuildXPages\demo\buildnsf\odperror

init:
  [copy] Copying 20 files to V:\Projects\BuildXPages\demo\buildnsf\odp
  [copy] Copying 20 files to V:\Projects\BuildXPages\demo\buildnsf\odperror

build:
[bxp:buildnsf] Attempt to Create Socket
[bxp:buildnsf] Socket Created
[bxp:buildnsf] CONNECTED TO DESIGNER! What can we do for you?
[bxp:buildnsf] Issuing Refresh Import Build Command for
V:\Projects\BuildXPages\demo\buildnsf\odp\project
[bxp:buildnsf] BUILD JOB ABOUT TO RUN
[bxp:buildnsf] BUILD JOB RUNNING
[bxp:buildnsf] BUILD JOB STATUS: SUCCESS
[bxp:buildnsf] PROBLEMS STATUS: SUCCESS
[bxp:buildnsf] No ProblemMarkers found after Building

BUILD SUCCESSFUL
Total time: 2 seconds
```

Also, in your domino Designer workspace you should see a new NSF with filepath temp\BuildXPagesDemo.nsf

## Run a Build with Errors

Let's see an example of running an NSF Build that results in some errors. To do this we will use the same sample On-Disk Project, but before we run the build we will sneakily delete the custom controls. This should cause an error when Designer tries to build the XPages that use those missing

custom controls.

The *target* that I have written to do this is called **buildfail**.

Note it is pretty much the same as **build** but is pointing to a different location for the on-disk project, and specifies different NSF and projectname.

```
<target name="buildfail" depends="init">

    <bxp:checkDesignerRunning/>

    <!-- We assign the .project file path to a property using the 'location'
attribute,
    this way the property will be expanded to the absolute path of the .project
file-->
    <property name="odp.project" location="odperror/.project" />

    <bxp:buildnsf project="${odp.project}" projectname="BuildXPagesDemoNSFError"
nsf="temp\BuildXPagesDemoError.nsf" />

</target>
```

Since it is not the default target, we will tell ant that that is the target we want to build by including the target name in the command line arguments.

1. issue the command `ant buildfail`

You should then see an output like so:

Buildfile: V:\Projects\BuildXPages\demo\buildnsf\build.xml

clean:

```
[delete] Deleting directory V:\Projects\BuildXPages\demo\buildnsf\odp
[delete] Deleting directory V:\Projects\BuildXPages\demo\buildnsf\odperror
```

init:

```
[copy] Copying 20 files to V:\Projects\BuildXPages\demo\buildnsf\odp
[copy] Copying 20 files to V:\Projects\BuildXPages\demo\buildnsf\odperror
```

buildfail:

```
[bpx:buildnsf] Attempt to Create Socket
[bpx:buildnsf] Socket Created
[bpx:buildnsf] CONNECTED TO DESIGNER! What can we do for you?
[bpx:buildnsf] Issuing Refresh Import Build Command for
V:\Projects\BuildXPages\demo\buildnsf\odperror\project
[bpx:buildnsf] BUILD JOB ABOUT TO RUN
[bpx:buildnsf] BUILD JOB RUNNING
[bpx:buildnsf] BUILD JOB STATUS: SUCCESS
[bpx:buildnsf] PROBLEMS STATUS: FAIL
[bpx:buildnsf] The following ProblemMarkers were present after building
[bpx:buildnsf] Error: Home.xsp: The unknown tag xc:ccCustomControl cannot be used as a
control.
```

Notice that the 'PROBLEMS STATUS' is fail and the last line is telling us the exact problem. If you were to look in Designer you would see the problem there too.

The screenshot shows the IBM Bluemix Designer interface. On the left is a project explorer with a tree view containing 'Buildxpagesdemo' and 'Buildxpagesdemoerror'. The main editor area displays the source code of 'Home - XPage'. The code includes XML headers and several XSP tags, with the tag `<xc:ccCustomControl></xc:ccCustomControl>` highlighted in blue. Below the code editor, there are tabs for 'Design', 'Source', 'Properties', 'Events', and 'Problems (1 error, 0 warning:'. The 'Problems' tab is active, showing a table with one error:

Description	Resource	Path
Errors (1 item)		
The unknown tag xc:ccCustomControl cannot be used as a control.	Home.xsp	BuildXPa



## Deploying an NSF

Building an NSF is one thing, but it is no good if you don't get it into the live system! Let's have a look at how we can achieve that.

There is more than one way to deploy an NSF, not everyone is going to do it the same way. This tutorial will hopefully demonstrate some of the tasks you could use to achieve your desired deployment.

### Scenario 1 - Super Simple

After building an NSF, you simply want to make it available to someone who will perform manual deployment.

Automated deployment is not for everyone and maybe manual deployment

### Scenario 2 - Refresh an Existing Template

This is my preferred option.

- I have 'local' nsfs located on the headless build machine
- I have a Production Template
- I have one or more Production Applications

I set the 'master template' name of the built nsf, and the inherit of the template I refresh the production template I either allow the design task to run overnight, or if doing an immediate deployment, I issue the load design task console command

## Building Features and Plugins

In this sample we build the plugins for a feature, and prepare an Update Site that can be used to deploy to Domino or Domino Designer

### First Time Setup

Before we build plugins, we need to make sure we are set up properly

- Install Eclipse for RCP and RAP Development
- Install Ant
- Install Build4XPages Ant tasks

### Configure Your Notes and Domino Program Directories

In order to run the build steps, we need to know where Notes and Domino are

- Set them up as Environment Variables
- Specify them in the build.properties file

### Configure Eclipse Properties

The build script needs to know 3 things in order to build your plugins

*What is the base directory of Eclipse?*

This may be something like `C:\eclipse`

*Where can I find the PDE Build.xml file?*

This is a file which provides the actual recipe to build plugins. In older versions of eclipse it will be under `<eclipse>/plugins/org.eclipse.pde.build_<version>/scripts/build.xml` In Newer versions of eclipse it will be located under your user home directory's p2 pool of plugins `<userhome>/p2/pool/plugins/org.eclipse.pde.build_<version>/scripts/build.xml`

*Where can I find the equinox launcher jar?*

This is an executable jar which launches the plugin build process. In older versions of eclipse it will be under `<eclipse>/plugins/org.eclipse.equinox.launcher_<version>.jar` In Newer versions of eclipse it will be located under your user home directory's p2 pool of plugins `<userhome>/p2/pool/plugins/org.eclipse.equinox.launcher_<version>.jar`

These 3 things should be the same for every project you build on the computer, so you can set these up as Environment Variables as well

If you don't want to set them up as environment variables, you can specify them in the accompanying build.properties file

### Run the Build!

1. Open your console and navigate to the demo/buildfeature folder
2. Type `ant`

### 3. Cross your fingers

After the build is finished you should see a zip file in the `demo/buildfeature/BuildXPages` directory.

## Deploying Plugins

TODO

## BuildXPages Ant Tasks

### What is Ant?

Apache Ant is a tool which you can use to automate repetitive steps. Ant has access to many commonly used steps such as manipulation of files in the filesystem (copy, move, delete, unzip etc.) and more complex tasks such as compiling java, running programs.

BuildXPages provides Ant with some custom tasks that are useful in the Build and Deployment process of XPages applications and XPages Plugins.

BuildXPages provides tasks related to

[Deploying NSF's to a Server](#)

[Building and Deploying Plugins](#)

[Interacting with Domino Designer for Building NSF's](#)

[Controlling a Domino Server](#)

## NSF Deployment Related Tasks

The following tasks are usually useful after you have built NSF's and need to move them around, refresh templates etc. to achieve deployment of an NSF that has been successfully built.

### copynsf

Copies an NSF from one place to another using the Notes C API's 'NSFDbCreateAndCopy' function.

#### NOTE

Currently this task only copies the Note class 'ALLNONDATA' which means design elements only. It could easily be modified to include all documents if you like. Let me know if you want this done.

### Attributes

*srcserver*

The Server of the Database to be copied. Optional - defaults to Local

*srcfilepath*

The Filepath of the Database to be copied

*dstserver*

The Destination Server for the newly created database. Optional - defaults to Local

*dstfilepath*

The Filepath the Database to be created.

### Example

```
<copynsf srcfilepath="demo\\GitTest.nsf" dstfilepath="demo\\DidItwork.nsf" />
```

## deletensf

Deletes an NSF using the Notes C API's NSFDbDelete method.

### Attributes

*server*

The Server of the NSF that you want to Delete. Optional - defaults to Local

*filename*

The filepath of the NSF that you want to Delete

### Example

```
<!-- Delete a Local NSF -->  
<deletensf filename="demo\\DidItwork.nsf" />
```

```
<!-- Delete an NSF on another Server -->  
<deletensf server="Domino02" filename="demo\\DidItwork.nsf" />
```

## settemplatenames

Updates the Template Inheritance settings of an NSF. You can set an NSF To be a master template, or you can set an NSF to Inherit from another Template, or Both! You can also clear the 'inherit from' or 'master template' settings.

### Attributes

*server*

The Server of the NSF that you are modifying template settings for

*database*

The filepath of the NSF that you are modifying template settings for

*clearinheritfrom*

Defaults to false. When set to 'true' will clear any inheritance settings if they exist

*inheritfrom*

The Name of the Master Template that you would like the NSF To inherit from

#### *clearmastername*

Defaults to false. When set to 'true' will clear the 'Is Master Template' settings of the nsf if they exist

#### *mastername*

This is the Template Name that you would like this NSF to be known as.

### **Example**

```
<!-- Set an NSF to inherit from 'DemoTemplate' -->
<bxp:settemplatenames database="MyFolder\MyNSF.nsf" server="Domino01"
inheritfrom="DemoTemplate">
</bxp:settemplatenames>
```

```
<!-- Set an NSF to be a Master Template called 'DemoTemplate' -->
<bxp:settemplatenames database="MyFolder\MyNSF.nsf" server="Domino01"
mastername="DemoTemplate">
</bxp:settemplatenames>
```

```
<!-- Clear the Inherit From settings so the NSF will no longer inherit from a template
-->
<bxp:settemplatenames database="MyFolder\MyNSF.nsf" server="Domino01"
clearinheritfrom="true">
</bxp:settemplatenames>
```

```
<!-- Clear the Master Template settings so the NSF will no longer be a Master template
-->
<bxp:settemplatenames database="MyFolder\MyNSF.nsf" server="Domino01"
clearmastername="true">
</bxp:settemplatenames>
```

### **scxd**

Sets the Single Copy XPage Design settings of an NSF. You can set both the relevant properties of the SCXD path, and the Flag which determines whether to use the SCXD template or not.

#### **Attributes**

##### *server*

The server of the NSF that you would like to set SCXD settings for

##### *database*

The filepath of the nsf that you would like to set SCXD settings for

### *scxdpath*

The path of the XPages database which you would like to use as the SCXD

### *scxdflag*

true/false determines whether the 'Use Single Copy XPage Design' checkbox is will be ticked in the Database Properties

## Example

```
<!-- Set an NSF to use 'scxd\Awesome.nsf' as it's single copy xpage design -->  
<bxp:scxd database="MyFolder\MyNSF.nsf" server="Domino01" scxdpath="scxd\Awesome.nsf"  
scxdflag="true" />
```

```
<!-- Set an NSF to not use any SCXD -->  
<bxp:scxd database="MyFolder\MyNSF.nsf" server="Domino01" scxdpath="" scxdflag="false"  
/>
```

## refreshdbdesign

Refreshes an NSF's design from a server using the Notes C API DesignRefresh method.

The console output shows all the design elements that are modified which is great to verify what has changed.

This task uses existing template settings of the NSF so if you need to change them you should use the settemplatenames task before using this task.

### Attributes

#### *server*

The server of the NSF that you would like to Refresh the Design of

#### *database*

The filepath of the NSF that you would like to refresh the Design of

#### *templateserver*

The server which has the Template that will be refreshed from

## Example

```
<!-- Refresh MyNSF.nsf from the Domino03 server -->  
<refreshdbdesign server="Domino02" database="MyFolder\MyNSF.nsf"  
templateserver="Domino03" />
```

# Plugin Related Tasks

The following tasks are all related to the building and deployment of OSGi plugins for XPages and Notes/Designer.

## buildfeature

### CAUTION

This Task and Documentation needs a little bit of love before it is truly re-usable. If you are keen to use this then let me know and I will improve it!

This task will build all the Plugins listed in a feature, using the headless eclipse PDE build system.

Before using this task, you will need to have prepared the build directory with the source code of the plugins that are to be built, and the the feature that is to be built.

### Properties that are used in this task

This task is actually a macro, and currently relies on some properties to have already been set in ant

#### *featureId*

The feature to be built

#### *eclipseBase*

Root directory of eclipse

#### *pdeBuildVersion*

PDE Build version

#### *equinoxLauncherVersion*

Equinox Launcher Version

#### *buildId*

Usually the build number

#### *buildLabel*

Project Name

#### *buildConfigDir*

Directory of build.properties

#### *buildDir*

The working directory in which the plugins will be built

### Attributes

#### *pluginPath*

This is a semi-colon delimited string of the target platform plugin paths



## Example

### CAUTION

You really need a lot more information than this to use this task but I have just put this all here as a starting point. Please feel free to contact me to make me provide more information.

```
<!-- Plugin path is supplied by a build.properties file -->
<bxp:buildfeature pluginpath="${pluginPath}" />
```

## importplugins

Imports plugins/features from an update site on the filesystem into an [Open Eclipse NSF Update Site](#)

### Attributes

#### *server*

The server that the **Open Eclipse NSF Update Site** is located on

#### *database*

The filepath of the **Open Eclipse NSF Update Site**

#### *sitexml*

The location of the site.xml of the **Filesystem update site** that you want to import

#### *deletefirst*

true/false, defaults to false. If set to true, it will clear the updatesite nsf of all existing plugins/features

## Example

```
<bxp:importplugins server="Domino02" database="UpdateSite\MyUpdateSite.nsf"
sitexml="C:\MyFolder\MyProject\com.my.updatesite\site.xml" />
```

## copyplugintobuilddir

This task is used during preparation for a headless eclipse plugin build. It is used to copy the source code of a plugin, into the *build directory* which is a special working directory used to build the plugins and features.

### Properties used

#### *pluginsDir*

The Destination directory, usually the eclipse/plugins folder of the build Directory

## Attributes

### *plugin*

the folder name of the plugin to copy

## Example

```
<bxp:copyplugintobuilddir plugin="com.acme.mypugin" />
```

## unpackplugin

Usually you update Domino Designer with new plugins by going through the normal installation process. But there is another way in which you can build plugins into a particular folder, and Domino Designer always picks up new versions.

When deploying plugins this way, some plugins need to be 'unpacked' which means they don't exist as a jar file, but the contents are unpacked into a folder.

This task will unpack a plugin to a folder. You just need to give the plugin name, and it will find the version information automatically

## Attributes

### *dir*

The directory that the plugin is in

### *pluginid*

the plugin id that you want to unpack

## Example

```
<bxp:unpackplugin dir="eclipse/plugins" pluginid="com.acme.superplugin" />
```

## generatesitexml

After building your plugins headlessly with eclipse PDE, you will then have a directory of plugins and features, but before importing plugins into an [Open Eclipse NSF Update Site](#) you will also need a **site.xml** file.

Newer versions of eclipse use a *p2 repository* format which does not require a site.xml, but with Notes/Domino running on a much older version of eclipse, it still uses an older format which uses the **site.xml** file.

The build process does not generate one for us so we do it ourselves with this task, which scans the features directory and builds the necessary xml.

## Attributes

### *eclipsedir*

The root folder that contains the plugins and features directory. I call it eclipse directory because often you will find it in the structure ../something/**eclipse**/plugins and ../something/**eclipse**/features

## Example

```
<bxp:generatesitexml eclipsedir="testgen" />
```

## clearupdatesite

### NOTE

This task does not refer to clearing an NSF update site, this task is for clearing and update site on the filesystem

After building plugins, sometimes you want to put them in an update site on the filesystem. You may want to make sure the update site is empty before you put your plugins there, so this task can clear out an existing update site.

## Attributes

### *eclipsedir*

The root folder that contains the plugins and features directory. I call it eclipse directory because often you will find it in the structure ../something/**eclipse**/plugins and ../something/**eclipse**/features

## Example

```
<bxp:clearupdatesite eclipsedir="myplugins" />
```

## copypluginstoupdatesite

### NOTE

This task does not refer to copying an NSF update site, this task is for copying to an update site on the filesystem

After building plugins, sometimes you want to put them in an update site on the filesystem. This task will copy the plugins to your target update site, and also generate the site xml.

## Properties

### *builtPluginsZip*

the zip file of the built plugins after headless build

## Attributes

### *eclipsedir*

The root folder that contains the plugins and features directory. I call it eclipse directory because often you will find it in the structure ../something/**eclipse**/plugins and ../something/**eclipse**/features

## Example

```
<bxp:copypluginstoupdatesite eclipsedir="my/target/updatesite/eclipse" />
```

# Designer Related Tasks

## startDesigner

### CAUTION

Using this task to start designer will cause designer to shutdown at the end of the ant build process. Due the the process being a sub-process. I need to do this a better way such as the powershell script way but I haven't got around to it yet. I actually in real life just leave designer running all the time so I don't need to start/close it.

Before building an NSF with the Headless Designer Plugin, you need to make sure designer is running. You can use the **checkDesignerRunning** task to check, and if it is not running you can use this **startDesigner** task.

## Properties

### *notesProgDir*

The Notes program directory on this machine

## Example

```
<bxp:startDesigner />
```

## buildnsf

The buildnsf task is used to build an ODP into an NSF. To do this it connects to the Designer Headless Server, and instructs it to build. If the NSF that is specified does not yet exist, it will be created. You

## Attributes

### *project*

This is the .project file that is in the root directory of the On-Disk Project

### *projectname*

When the on disk project is imported it must be given a 'name'. This name needs to be unique, so you can specify it here. Note: Do not put your ODP under the Notes Workspace.

### *server*

This is the server of the NSF that you are building

### *nsf*

This is the filepath of the nsf that your building with

### *port*

This specifies the port that your Designer Headless Server is running on (Default 8282)

### *failonerror*

If Errors are found against the nsf after building, fail the task

## **Example**

```
<target name="testbuildnsf" description="Build an NSF">
  <bxp:buildnsf project="D:\\workspaces\\testworkspace\\WahWah\\.project"
projectname="FromAnt" nsf="stageing\\fromant2.nsf">
  </bxp:buildnsf>
</target>
```

```
<target name="testbuildnsf2" description="Build an NSF">
  <bxp:buildnsf ondiskproject="V:\\eclipse-kepler\\runtime-
DominoDesigner\\DoraHeadless\\.project" targetfilename="fromant2.nsf" port="8283">
  </bxp:buildnsf>
</target>
```

## **closedesigner**

This task connects to the Headless Designer Plugin and instructs it to shutdown.

### **Attributes**

#### *port*

The port that the Headless Designer Plugin is running on. Default = 8282

## **Example**

```
<bxp:closedesigner />
```

## markersreport

This task connects to the Headless Designer Plugin, and requests a report of all the problem markse (e.g. Error, Info, Warning etc.) for a particular NSF.

If you like , you can cause the build to fail if there are errors present on the NSF, to prevent further tasks like deployment.

### Attributes

#### *ondiskproject*

The On Disk Project file (.project) of the NSF that you want a report for.

#### *failonerror*

When set to true (Default) the build will fail if errors are present on the NSF

#### *port*

The port which Headless Designer Plugin is running on (default = 8282)

### Example

```
<bxp:markersreport ondiskproject="C:\\workspaces\\neon-64\\runtime-HeadlessPlugin\\DeleteMe\\.project">
</bxp:markersreport>
```

## checkdesignerrunning

**CAUTION** | Windows only!

This task checks if the nlnotes.exe process is running, if so it sets a property **designer.running**. You can then use this property to decide whether to take some other action.

### Example

```
<bxp:checkDesignerRunning />
<fail if="designer.running" message="Designer was running" />
```

## updateDesignerLink

**NOTE** | I am not sure this is a great way to deploy plugins so I haven't detailed this part yet.

This task is used to set up an update site that will always be loaded when Designer Is started up. This is only used when you are deploying plugins to Domino Designer using the sneaky 'permanent update site' method.

## Properties

*notesProgDir*

The Notes Program Directory on this machine

## Attributes

*updateSiteLabel*

This is a short code to be used as a name for the update site. It will be used in a filename so it should be something simple e.g. 'extlib'

*updateSiteDir*

This is the Directory that contains the update site. It is expected to have a subdirectory called **eclipse**, and within that subdirectory should be the site.xml, **features** directory and **plugins** directory

## Example

```
<bxp:updateDesignerLink updateSiteLabel="extlib"  
updateSiteDir="C:\\Projects\\extlib\\updatesite" />
```

## initdesignersite

### NOTE

I am not sure this is a great way to deploy plugins so I haven't detailed this part yet.

This task is used to set up an update site that will always be loaded when Designer Is started up. This is only used when you are deploying plugins to Domino Designer using the sneaky 'permanent update site' method.

This task is just like the **updateDesignerLink** task, however it will create the update site folder structure within the framework directory of the Notes Installation.

You can then copy plugins to this update site, and they will be loaded by designer (if you have configured platform.xml using the **checkplatformxml** task)

## Properties

*notesProgDir*

The Notes Program Directory on this machine

## Attributes

*updateSiteLabel*

This is a short code to be used as a name for the update site. It will be used in a filename so it should be something simple e.g. 'extlib'

## Example

```
<bxp:updateDesignerLink updateSiteLabel="extlib"  
updateSiteDir="C:\\Projects\\extlib\\updatesite" />
```

## checkplatformxml

This checks that the IBM Notes Installation's platform.xml file is configured to automatically load plugins from the filesystem. The task will fail if the file is not configured properly.

### Properties

*notesProgDir*

The Notes Program Directory on this machine

## Example

```
<!-- Make sure notesDataDir property is set -->  
<property name="notesDataDir" location="H:\\Notes\\Data"/>  
  
<bxp:checkPlatformXml/>
```

## configuredynamicplugins

This task configures IBM Notes' **platform.xml** file so that IBM Notes will load plugins from the filesystem without the need to 'approve' them via the User Interface.

Usually you need to go through the whole provisioning process for plugin updates, but this task will put IBM Notes into a configuration that bypasses this.

### Properties

*notesDataDir*

the location of IBM Notes Data directory on this machine

## Example

```
<!-- Make sure notesDataDir property is set -->  
<property name="notesDataDir" location="H:\\Notes\\Data"/>  
  
<bxp:configureDynamicPlugins />
```

## Server Related Ant Tasks

These tasks are designed to interact with the domino server by issuing commands to the console.



## controlhttp

The controlhttp task is used to control a Domino Server's http tasks. It ultimately just issues a console command to the domino console.

### Attributes

*server*

The server that you are controlling

*action*

start | stop | restart

### Example

```
<target name="teststarthttp">
  <bxp:controlhttp server="Domino02" action="start" />
</target>
```

```
<target name="teststophttp">
  <bxp:controlhttp server="Domino02" action="stop" />
</target>
```

```
<target name="testrestarthttp">
  <bxp:controlhttp server="Domino02" action="restart" />
</target>
```

## maintenancewarning

The maintenance warning task issues a console command to the Domino OSGi console which tells current users that there will be a maintenance outage in a certain amount of time

The Maintenance warning relies on another plugin that has not yet been added to BuildXPages so this task won't be useful until that plugin is added to this project

### Attributes

*server*

The server that you are controlling

*minutes*

in how many minutes time is the outage going to occur

## Example

```
<target name="testmaintenancewarn">
  <bxp:maintenancewarning server="CameronWS" minutes="20" />
</target>
```

```
<target name="testmwcancel">
  <bxp:maintenancewarning server="CameronWS" cancelwarning="true" />
</target>
```

## loaddesign

Issues a 'load design' console command for a server

### Attributes

*server*

The server that you are controlling

*directory*

which directory should have it's designs refreshed

## Example

```
<target name="testloaddesign">
  <bxp:loaddesign server="CameronWS" directory="Cameron\Jogs\JobHub" />
</target>
```