G11NToolKit Extensions Guide

by Bill Rich

Extensions Guide for the G11NToolKit.

Table of contents

1 How to Use Extensions......2
2 How to Install Extensions......4
3 How to Test Extensions......5
4 Sample Extensions......5
4.1 Perforce Extensions......6
4.2 XML File Extensions......7
4.3 Java ListResourceBundle File Extensions......8

1. How to Use Extensions

To use the extensions mechanism of the G11NToolKit all files associated with the extensions must be placed in sub-directories of the G11NToolKit `extensions` directory. This directory comes with some default files as place holders. Some of the files must be overridden with code for the extensions. Other files, like the `readme.html` files can be used for whatever purpose the user decides.

The easiest way to get started is to download the G11NToolKit distribution files that you need and unzip them into a base directory. The minimum you need are the `G11NToolKit-1.2.0-bin.zip` and `G11NToolKit-1.2.0-ext.zip` files. These contain all the executable files that are needed for the G11NToolKit. Unpacking the `G11NToolKit-1.2.0-bin.zip` file will create the `bin`, `etc`, `L10NProcess`, and `lib` directories. Unpacking the `G11NToolKit-1.2.0-ext.zip` file will create the `extensions` directory and all of its sub-directories. This split is intentional since the `G11NToolKit-1.2.0-ext.zip` file contains default files that will conflict with some of the extension files that a user would create. Unzipping the `G11NToolKit-1.2.0-ext.zip` file should be done with extreme care once extensions are added to the G11NToolKit for a particular installation.

The `bin` directory can contain any executable file that is needed for the extensions. This directory is not referenced in the G11NToolKit, it is provided as a convenience to the user.

The `Doc` directory can contain the documentation files for the extensions. There is a link in the G11NToolKit documentation navigation bar file to the `extensions/Doc/index.html` file. This file must be available in the `Doc` directory. A default version of `index.html` is provided in the `G11NToolKit-1.2.0-ext.zip` file. It should be replaced by a file that documents the extensions. The replacement file can contain any HTML coding to present the extensions documentation in any form the user desires. All files needed for the documentation of the extensions should be copied to the `extensions/Doc` directory or sub-directories so that they are easily accessible from the `index.html` file.

The `etc` directory should contain any helper files, such as XSLT files, that are needed by the extensions.

In the `L10nProcess` directory the `L10NProcess-extensions.xml` file is requried since it is referenced in the `L10NProcess.xml` file in the G11NToolKit. It contains targets that must be overriden to handle the extensions to the process. You can also import other Ant files in the `L10NProcess-extensions.xml` file as needed. File names, target names, and property names must not conflict with G11NToolKit names.

The targets that must be overridden are:

`checkin-ext`
Must implement any extensions to the `checkin` target. This would include any special processing for a particular file type or source code control system.

`clean-ext`
Must implement any extensions to the `clean` target. This would be for any special file type or directory.

`copysrctar-ext`
Must implement any extensions to the `copysrctar` target. This would be for any special file type or directory. `copysrctar` copies the source files into their target directories and file names but does not change the file content except where a class name is changed to match the target file name as required by Java.

`detok-ext`
Must implement any extensions to the `detok` target. This would be for any special file type or directory.

`get-ext`
Must implement any extensions to the `get` target. This would be for any special file type or directory or for any special requirements of the source code control system..

`init-ext`
Must implement any extensions to the `init` target. This would be for any properties required by the extensions.

`tok-ext`
Must implement any extensions to the `tok` target. This would be for any special file type or directory.

The `lib` directory contains any `jar` or `zip` file that may be neede in the path expression for the extensions. The `lib` directory is referenced in the path expression for the G11NToolKit. It includes any `jar` or `zip` file in the directory.

Return to: Top of page

2. How to Install Extensions

For illustrative purposes we will assume that you are installing the tool kit on a Windows machine, the directory you want to use for the G11NToolKit is named `myG11NToolKit`, and it will be placed in the root of the `C` drive on your machine. Of course, you can change the drive and directory name to anything you choose.

To Install the combination of the G11NToolKit and one extensions package on a Windows machine you will need to:

1. Use whatever mechanism is appropriate for your version of Windows to set the `G11NTOOLKIT-DIR` environment variable. From the command line it would look like this:
   ```
   set G11NTOOLKIT-DIR=C:\myG11NToolKit
   ```
   It is recommended that you do not use the same directory where you will be doing G11NToolKit development work. We have had some conflicts doing this where the build file tries to delete a directory and fails because it is being used or was used and not released.
2. Download the following files from the Files area at the G11NToolKit Project (these are the minimum number of files you must download, you can download more if you want)
   - `G11NToolKit-1.2.0-bin.zip`
   - `G11NToolKit-1.2.0-doc.zip`
   - `G11NToolKit-1.2.0-ext.zip`
3. Unzip the files to `C:\myG11NToolKit` (or whatever directory you set as `G11NTOOLKIT-DIR`).
4. Identify and locate the files needed for the extensions package.
5. Copy these files into the appropriate directories under `C:\myG11NToolKit\extensions` directory. You will have files to place into some or all of the following directories:
   - `C:\myG11NToolKit\extensions\bin`
   - `C:\myG11NToolKit\extensions\Doc`

- `C:\myG11NToolKit\extensions\etc`
- `C:\myG11NToolKit\extensions\L10NProcess`
- `C:\myG11NToolKit\extensions\lib`

6. The G11NToolKit and the extensions package are now ready to use.
7. Opening the `C:\myG11NToolKit\Doc\index.html` file will display the G11NToolKit documentation.
8. Clicking on the "G11NToolKit Extensions Guide" link in the navigation bar will display the extensions package documentation (if any is provided).

Return to: Top of page

3. How to Test Extensions

Following the assumptions from the Install section do the following steps to test the extensions:

1. Download the `G11NToolKit-1.2.0-test.zip` from the Files area the G11NToolKit Project (If you did this already you can skip it.)
2. Unzip the file to `C:\myG11NToolKit` (or whatever directory you set as `G11NTOOLKIT-DIR`) (If you did this already you can skip it.)
3. Identify and locate the test case files needed to test the extensions package.
4. Make the `C:\myG11NToolKit\test\ext` directory.
5. Copy the extensions test case files into the `C:\myG11NToolKit\test\ext` directory.
6. Run both the G11NToolKit test cases and the extension test cases to verify that both the G11NToolKit and the extensions package are working properly.

Return to: Top of page

4. Sample Extensions

There are some samples of extensions in the `G11NToolKit\Doc\ExtensionSamples`. There are three examples included in the files: some extensions for using Perforce as the source code management system, an extension to handle a variation of XML files, and an extension to handle a variation of the Java ListResourceBundle file.

4.1. Perforce Extensions

In adding support for using the Perforce source code control system we need to consider several important things:

- How to identify the key elements for Perforce
- How to get files from Perforce
- How to put files into Perforce

The key elements that are needed are the `P4User`, `P4Client`, `P4Depot`, and `P4Port` identifiers. The `P4User`, `P4Client`, and `P4Depot` identifies can vary for each user while the `P4Port` identifier will be the same for all users of the project.

To handle the user variables we create a `user.properties` file and add the Perforce identifiers to it. For the project level identifier we add this to the `project.properties` file.

To get files from Perforce requires some Ant tasks be added to the L10N Process files. To do this we create the file. In this file we add targets to synchronize the Perforce depot and copy the files from the depot to the project work area.

Putting files into Perforce is similar to getting files from Perforce. To do this we add some targets to the `ExtSamp-p4tasks.xml` file. These targets check to see if the target file already exists in the Perforce depot, and, if the file exists, issue the `P4 Edit` command, otherwise issue the `P4 Add` command. After the Perforce command is processed then the new target file is copied to the Perforce depot. When all the target files are processed a `P4 Revert` command is run for all unchanged files. This assures that only files that have been modified will be replaced on the Perforce server. It was decided that the user would be responsible for checking the change list and running the final `P4 submit` command to make sure the update is correct before committing it.

There is one last thing we need to make the Perforce extensions available. We need to add an import for the `ExtSamp-p4tasks.xml` file to the `ExtSamp-tasks.xml` file. If this is not added Ant will not be able to recognize our Perforce targets.

Return to: Top of page Sample Extensions

4.2. XML File Extensions

In our example to handle a variation of XML files we need to translate the the value of the `value` attribute when the `name` attribute has a value of `titles` for any `param` element in the file. The source file is also created by applying an XSLT file to the source file.

Since we need to apply the XSLT file to the source file first we will need to modify the Ant target that tokenizes an XML file. We add the `tokxm2files` target to the `ExtSamp-toktasks.xml` file. We also need to make sure this target only runs for files of type `xm2` so we limit it to processing only the `projXLFileList_XM2` file list.

Another variation for these XML files is that we need to describe what to extract. This description is in the form of a regular expression and is included in a special properties file. The `xm2.properties` file is used for this purpose. We use the file type identifier as the file name so we can easily identify the file later. The file is then added to the `tokxm2files` target as the `xmlprops` parameter value.

Since an `xm2` type file is a new file type we will need to provide a detokenizer for it also. There is no difference in the way an `xm2` file is detokenized so we can just copy the `detokxmlfile` task and change the file type that is handled. We do this in the file. Here again we limit the target to processing only the `projXLFileList_XM2` file list.

The project control file also needs a slight change to handle the `xm2` file type. This is shown in the `ExtSamp.slpd` file. It is just a matter of adding a file entry for any `xm2` type file that needs processing.

Return to: Top of page Sample Extensions

4.3. Java ListResourceBundle File Extensions

Making an extension to handle the `oddlrb` type files is similar to making the extension for the `xm2` type files with the added attraction that we need to actually write a tokenizer class to handle the parsing requirements.

We start by writing the tokenizer class that we will call `TokODDListResourceBundle`. We can subclass the `TokListResourceBundle` class and add the code needed to handle the parsing requirements. You can inspect the code to see how this was done.

Since we need the extended tokenizer class we need to execute it so we add the `tokoddlrbfiles` target to the `ExtSamp-toktasks.xml` file. We also need to make sure this target only runs for files of type `oddlrb` so we limit it to processing only the `projXLFileList_ODDLRB` file list.

Since an `oddlrb` type file is a new file type we will need to provide a detokenizer for it also. There is no difference in the way an `oddlrb` file is detokenized so we can just copy the `detoklrbfile` task and change the file type that is handled. We do this in the `ExtSamp-detoktasks.xml` file. Here again we limit the target to processing only the `projXLFileList_ODDLRB` file list.

The project control file also needs a slight change to handle the `oddlrb` file type. This is shown in the file. It is just a matter of adding a file entry for any `oddlrb` type file that needs processing.

Return to: Top of page Sample Extensions