



BixChange in Five Minutes Flat

There really isn't much to BixChange. This guide is dedicated to providing you with a quick introduction to the system.

Although BixChange is written in Perl, you really do not need to know much Perl to use it. You, however, will need to learn to think of problems in the BixChange 'way'.

In order to understand BixChange it may be necessary to build a token program with it – something in-between “Hello World” and a “Duwamish Books”. Since the goal of this document is to provide you with a tutorial in five minutes or less, we'll have to consider a relatively simple problem that arises often – the need to show users, on-demand, a list of files in a particular directory. We'll assume that there is not enough time to give each user access to any type of shared directory and, besides, you need to provide the users will some other information along with the files, so a file-share solution is inadequate.

The rest of this tutorial assumes you have already installed BixChange. If you have not done so, please visit www.bixchange.com, extract the archive in a script-executable directory underneath your webserver, and follow the INSTALL.txt directions.

The Zero-Code Solution

If you cannot program, or you really are just too busy, you can opt for this solution. It involves the bare-bones, non-procedural solution with minimum benefits.

First, in order to do anything in BixChange you will need an INI file and a template. Here's a very simple INI file (store it in /data/f/fi/fiveminute.ini):



BixChange in Five Minutes Flat

```
[GENERAL]
PAGETYPE=FORM
SHOWDIR=<<EOF
```

```
FILE1.DBM
FILE2.DBM
FILE3.DBM
FILE4.DBM
```

```
EOF
```

Here is the associated template (/tmpl/fiveminute.tpl):

```
<HTML>
  <BODY>
    <h2>List of Files</h2>
    Okay people, here are the files. Periodically, these
files will be updated by DEPARTMENT X.
    <PRE>
      <TMPL_VAR NAME="SHOWDIR">
    </PRE>
  </BODY>
</HTML>
```

In order to test, you can run this on the command line by calling BixChange.cgi in the following manner:

```
perl -T bixchange.cgi f=fiveminute tpl=fiveminute
```

What's the advantage? First off, the template can be reused for *any* number of INI files. If you wanted to have a different file for department Y, you could change your INI file and associated template like so:

```
[GENERAL]
PAGETYPE=FORM
DEPARTMENT=Y
SHOWDIR=<<EOF
```



BixChange in Five Minutes Flat

```
FILE1.DBM  
FILE2.DBM  
FILE3.DBM  
FILE4.DBM
```

EOF

```
<HTML>  
  <BODY>  
    <h2>List of Files</h2>  
    Okay people, here are the files. Periodically, these  
files will be updated by DEPARTMENT <TMPL_VAR  
NAME="DEPARTMENT">.  
    <PRE>  
    <TMPL_VAR NAME="SHOWDIR">  
    </PRE>  
  </BODY>  
</HTML>
```

All very simple. The tags in the template are dynamically swapped out whenever a user calls BixChange. This also makes for an expandable solution since you can add more INI files and reference them in all sorts of ways. Going back to the multi-department scenario, you can add a different INI file for each department and one INI file listing all of the departments. In order to give users the option to select their own department's data, we simply add an additional section to our web template (multiple INI files can provide output to a single template in BixChange).

Here's the `five_minutedeps.ini` file which quickly lists the various departments:

```
[GENERAL]  
PAGETYPE=FORM
```



BixChange in Five Minutes Flat

```
[LOOP DEPTS1]
DEPARTMENT=X
```

```
[LOOP DEPTS2]
DEPARTMENT=Y
```

```
[LOOP DEPTS3]
DEPARTMENT=Z
```

And here is, again, the modified `fiveminute.tpl` template:

```
<HTML>
  <BODY>
    <h2>List of Files</h2>
    Okay people, here are the files. Periodically, these
    files will be updated by DEPARTMENT X.
    <PRE>
    <TMPL_VAR NAME="SHOWDIR">
    </PRE>
    <P />
    <TMPL_LOOP NAME="DEPTS">
      <a href="bixchange.cgi?f=fiveminutedept<TMPL_VAR
NAME="DEPARTMENT">;tmpl=fiveminute">DEPARTMENT <TMPL_VAR
NAME="DEPARTMENT"></a>
      <BR />&nbsp;  <BR>
    </TMPL_LOOP>
  </BODY>
</HTML>
```

This is called in the following way (now, assuming that you have graduated up to using your local web server):

```
http://localhost/cgi-
bin/bixchange/bixchange.cgi?f=fiveminute;f1=fiveminutedepts
;tmpl=fiveminute
```

Quick note: The above INI and template utilize a quick-and-dirty trick of BixChange – the LOOP feature. The INI sections have the LOOP



BixChange in Five Minutes Flat

<NAME><x> syntax, where NAME is some identifier and x is a number. Without adding a sort, the items are not guaranteed to appear in the same order in the resulting web page. For more on sorting INI data see the BixChange Development Guide at www.bixchange.com.

For Some, This is Pretty Much It

Believe it or not, for many developers this is the end of the road for BixChange learning. Their use of BixChange is limited to rendering the contents of INI files to the masses as they utilize various scripts, database stored procedures, or other utilities to create INI files for long-term archival. This was actually the original purpose of the BixChange system – to allow for web-enabled reading of archived laboratory data.

The early 'typical' BixChange developer managed enterprise systems and had little time (or patience) to learn much HTML. The desire was to simply locate a quality HTML template online or to take one from the corporate Intranet that already met company standards and 'hack' it with some kind of tagging mechanism.

The system has come along way from it's original roots. Today, the soon to be released 1.8b version of BixChange supports Apache::Registry, a mod_perl module for greater performance; allows you to run code inside of INI files; does not hinder you from connecting to databases; and more.

Wrapping Up the Non-Code Solution

The more INI and template files you have the more of a pain your solution will become. Such is life. If your INI files make up a single project there are ways to to keep things in BixChange tidy and easy to load into Emacs, a powerful cross-platform text editor. First, you'll need a .dat file. This file contains the paths to the relevant files of your project that are located underneath the BixChange root folder as shown here in fiveminute.dat:



BixChange in Five Minutes Flat

```
./data/f/fi/fiveminute.ini
./data/f/fi/fiveminutedepts.ini
./tmpl/fiveminute.tpl
fiveminute.dat
```

The last entry in the .dat file is the name of the file itself. The .dat file is stored in the same directory as BixChange itself. There is a reason for all of this. BixChange comes with an E-macs script that loads an entire BixChange module with a single command. For more details on that, click on the E-macs tutorial at www.bixchange.com.

Zip your module with Tar:

```
tar -cvf fiveminute-v0.1.tar -T fiveminute.dat
```

The 0.1 is the version number for your module. Whenever you make significant change to your files, make another version.

Okay, Add Some Code

The good thing about moving up to a coded solution for our example module is that you do not have to dump what you have done already. This is why BixChange is so useful as a prototyping tool. Your web templates can be thoroughly tested before you write a single line of code, proving usability with hard-coded, but easily modifiable, INI values.

Anyway, let's see some code:

```
[GENERAL]
PAGETYPE=FORM
SHOWDIR=<<<EOF
%%
my $result = ' ';
$result .= `dir /b .\\data\\*.dbm.*`;
$result || ' ';
```



BixChange in Five Minutes Flat

```
%%  
EOF
```

The %% symbols tell BixChange that you want to execute the contents as a command. The backticks tell Perl to run a shell command (a DOS command in our example).

Basically, this code is pretty boring and good only as a proof-of-concept. You will crave a better solution or else you'll start putting HTML inside of the code of the SHOWDIR tag. The good news is that, with a bit of tweaking, you can do better with relative ease.

Before we do that though, let's save this version:

```
tar -cvf fiveminute-v0.2.tar -T fiveminute.dat
```

Reusing the same code:

```
[GENERAL]  
PAGETYPE=FORM  
SHOWDIR=<<<EOF  
  %%  
  my $result = ' ';  
  my @files = ();  
  $result .= `dir /b .\\data\\*.dbm.*`;  
  foreach(split /\n/, $result)  
  {  
    push @files, {'FNAME'=>$_};  
  }  
  BixChange::_bankdeposit({FILELOOPER=>\@files});  
  $result || ' ';  
  %%  
EOF
```

We can now change the template thus:



BixChange in Five Minutes Flat

```
<HTML>
  <BODY>
    <h2>List of Files</h2>
    Okay, people, here are the files. Periodically, these
    files will be updated by DEPARTMENT X.

    <HR />
    <TMPL_LOOP NAME="FILELOOPER">
      <TMPL_VAR NAME="FNAME"><BR />
    </TMPL_LOOP>
    <P />

  </BODY>
</HTML>
```

Huh? How does this work?

Using a bit of Perl wizardry, the script now chops up the output of the directory command into single elements that it pushes into an array. From there we use BixChange's `bankdeposit()` function to hand-off the results to the system for placement on the actual web page.

BixChange comes with plenty of examples (data/m/ma/main*.ini files) like this one. Also, the BixChange website comes with a developer's guide to more fully explain many of the concepts briefly touched upon above.

Wrap-Up

That's it. In five minutes you have learned the following things:

- BixChange can create entire websites with separate data and web template elements
- BixChange is extensible (anything that can be called from Perl and respond via `Bankdeposit` can be 'BixChanged')
- Converting scripts and command line one-liners to web extensible



BixChange in Five Minutes Flat

- web applications is pretty painless
- All of this can be done with CGI for virtually any HTTP web server that runs CGI (Apache, THHTTPD, IIS, etc.)
 - Testing can be done on the command line without the webserver
 - There's a lot more to BixChange, but you'll have to read the material at www.bixchange.com to learn it.