

Building the Firebird distribution of InterBase

By Paul Reeves

InterBase became an open source project back in July. It was disappointing, however, to find that the code was virtually unbuildable outside of Scott's Valley. Whatever else had gone on in the preceding six months, little had been done to prepare the code for compilation by ordinary(ish) users. Some capable developers quickly started to rectify this and soon the Firebird project was born. As a result, it is now possible to visit the project at Sourceforge, download the source, follow the build instructions and have your own custom built version of InterBase running in less than an hour. I'm impressed.

Most of the work on Firebird seems to be happening around the Linux platform. This is probably due to its open source roots. The folk with the most experience of this sort of thing use Linux as a matter of routine. So, unless stated otherwise, pretty much all of this discussion is founded on using Linux as the operating system. Work has also been done to make building on Win32 feasible, but we will come to that later. Additionally, there have been successful ports to FreeBSD, AIX and the new Mac OS X. A project is also underway to develop a port for SCO.

Of course, despite the fact that the build is now straightforward and actually works, there are still a few issues to consider when it comes to building Firebird. This article will take a look at a few of them, with a particular emphasis on the Linux version.

Getting started - a five minute guide

If you want to delve into the mysteries of building InterBase the Linux version is the easiest place to start. The build procedure is well-documented, straightforward, works, and has been followed by the greatest number of developers. If you run into a problem, you can be sure you won't be the first and, equally, you will find a quick solution. So, what has to be done?

Firstly, a decent install of Linux is required. Recent versions of Red Hat (6.1,6.2 and, I believe 7.) have been tested. SuSE versions 6.3, 6.4 and probably 7.0 also work. My preferred distributor is SuSE and my testing has been under 6.3. I haven't looked too deeply into this issue, but given the 'cheapness' of Linux, the basic rule of thumb should be that any distribution released in the last year or so will have everything you need to work with. Don't waste time with an old version. If you have RedHat 7.0 don't forget that the initial release had a busted version of glibc. You need to get the patch for this.

Secondly, a good chunk of disc space is in order. You need to install the Gnu C development environment, along with a variety of other tools you may never have heard of. (I even found that I had to install 'ed'. This is a line-oriented text editor. Not the sort of thing I want cluttering up my hard disk, but required by part of the build process.) I find, these days, that my Linux install requires a good Megabyte of disc space before I have even started using it. The InterBase source itself takes up about 130Mb, and I find that copies of it seem to proliferate easily. Building is also easier if an existing version of InterBase 6 is already installed. Even if it isn't, you will need a special boot kit, so make sure you have plenty of disc space.

Thirdly, you need to get it all up and running and configured correctly. As part of this you will need some sort of communication with the outside world - either to the rest of your network, or directly via a modem to the Internet.

Once you have the machine set up, things are straightforward. You connect to the the Firebird CVS page - http://sourceforge.net/cvs/?group_id=9028 and follow the instructions to checkout the source. The checkout process takes me about 20 minutes using a 64K ISDN line. Once you have the source downloaded go to `download_path/interbase/build` and take a look at 'ibbuild_linux_cvs_build.html'. This has step-by-step instructions on what to do. They are clear and, at the time of writing, nearly 100% accurate. Anything up to forty minutes later should then see you with a working copy of InterBase built by your own hands.

So, that's it, really. Well, not quite. There are pitfalls along the way. I found a few and no doubt you will find others. There is nothing like a user to confound the best-written instructions.

My problems

The first challenge was using CVS to actually get the code. You need CVS running on your machine. CVS needs to be able to see the sourceforge host. If you have a direct connection to the InterNet, no problem. If, like me, your Linux box is just one of several on an internal network and external access is via an NT server, then you probably won't be able to see Sourceforge from the target. After a few fruitless hours extending my understanding of networking I still couldn't ping Sourceforge.

Onto plan B. My NT box has the Cygwin stuff installed. This is basically the Gnu development environment built for windows. It's pretty good and includes CVS. Getting it downloaded is a bit long-winded, but it installs fine. So, I made my dialup connection to Sourceforge from the NT box and followed the CVS instructions. In my case, I placed the code directly on my Linux box using Samba, but I could just as easily have stored it locally.

This presents the next problem. By introducing a Windows box to the process every line feed gets a carriage return prepended, which automatically means that any attempt to build on Linux will fail. So we have to remove the CRs. Fortunately, this is a path previously trodden. In particular, Reed Mideke, a former InterBase engineer who now contributes to Firebird, has solved the problem with this little script:

```
#!/bin/sh
# Description: I (Reed Mideke) created this
# script because my windows box is the only one
# with direct internet access, but I need to do
# testing on my linux box (which is on a
# private lan)
# I copy my CVS tree from Windows using Samba,
# and then run this script.
```

```

echo removing execute permissions
find . -type f -exec chmod a-x {} \;
echo converging to unix textmode
find . -type f -not \( -name "*.gbk" -o -name
    "*.gbak" -o -name "*.ico"
    \) -exec recode ibmpc..lat1 {} \;
echo restoring execute permissions to select files
chmod a+x setup_dirs setup_build builds/
    original/buildRefDatabases

```

It works like a charm, even if how it works is a little unclear (have a look at the man page for 'recode'). It also handles the way that execute permissions are messed up by the transfer.

An alternative suggestion, which I haven't tried, is to use the functionality of zip/unzip. There are (at least on the infozip variant) switches to add or remove the CR as part of the compression/decompression process. It can also convert a file between ascii and binary (more on this later), although it won't solve the perms issue.

With the source successfully placed on my Linux box I was ready to start the build. The next step is to create the reference databases. These InterBase databases store crucial information required by the build process. So I ran buildRefDatabases as per the instructions. Here everything failed, with GBAK having a permissions problem. Not having had much experience with Linux at that time it took me a while to work out what was happening. I was logged in as 'root'. InterBase was running as the 'interbase' user. User 'interbase' is a member of the 'root' group. But the buildRefDatabases script doesn't give write perms to 'group' or 'other' so GBAK can't restore the databases. This is easily remedied by amending the script and by the time this article appears that will have been done. Remedies are always easier once the cause of the symptoms has been identified.

I next ran into a problem attempting to build the messages database. Somehow, the trip across the network had turned the file from binary to ascii and added an extra few hundred kilobytes in file size. I wasn't in the mood to work out why this had happened, but the solution was straightforward. Sourceforge has a web interface into the CVS tree. It was simple to do a specific checkout of the one file straight to my Linux box. After that, the reference databases were created successfully, although restoring the forms.gdb still threw an error. As it is a deprecated part of InterBase, I didn't pay any attention to it.

That took me to the point where I could start the build. Everything ran smoothly until gpre tried to build the met.e file in the jrd directory. This failed and the error killed the whole build process. The fix is easy enough. You have to change to the appropriate directory, run the line that failed manually, change directories again and restart the build. A similar error will appear when building the Alice directory. This problem only occurs the first time you build InterBase from a source tree. Subsequent builds will succeed. This issue is documented, but not in the build instructions I mentioned above. Again, by the time this article appears I hope this problem will be documented more visibly.

Running the build for the third time did the trick. On my slow old Pentium the process took about half an hour but at the end were the magic words 'Build succeeded'. I shut down the production version of InterBase and started my new copy, with all the latest bug fixes. Connecting to it and executing some SQL presented no problems. Which is just how it should be.

Building On Win32

Building InterBase on Windows is slightly more complicated, or perhaps just complicated in a different way. Currently you will need tools from three compilers to succeed. In a nutshell, you need:

- The 'make' utility from Borland. The one supplied with the free C compiler will do.
- The Microsoft C compiler for the actual code compilation.
- The Cygwin or MKS Korn shell, to provide all the unix tools needed to join everything up.

In an ideal world all would be done with the cygwin stuff, but we are not in an ideal world. It is generally accepted that the MSVC compiler is the best optimised for Windows. Just about every cross-platform open source project discards GCC in its' favour. There are two issues when it comes to building InterBase. Firstly, a 'home-built' production version should not be slower than the 'standard' and, secondly, using GCC or BCC may introduce a different set of behaviours (or shall we say bugs?). I'm currently lacking the Microsoft component, so I haven't had a chance to test things out. Suffice to say that Reed Mideke has provided some good documentation for those in a position to try.

Conclusion

It is almost guaranteed that you will encounter problems when you first start. Aside from idle curiosity, the reasons for building from the source are usually either that you want to run on an unsupported platform or you have a bug fix/enhancement to implement. Either way, your circumstances are beyond that which can be satisfied by use of a standard build. Plus, the skills needed go beyond those usually required for database development work. For these reasons it must be expected that you will encounter problems that have not been foreseen or are undocumented. Don't worry. There is a good support network in place via the Firebird developers list.

We are a long way on from the situation last July. Building from the source was then almost impossible. Now, it is possible and increasing numbers of developers are doing it. We are moving to the point where bugs are rapidly getting fixed and the source code is being updated as a result. The ultimate expectation of an Open Source project is that developers everywhere combine to build a better wheel. We are now approaching the point where a core are beginning to understand how this wheel works, so expect to see some enhancements appearing in the coming months.

Links

<http://firebird.sourceforge.net/> has all the links you need to get started with building the code..

<http://www.ibphoenix.com> has pretty much everything else you need to know about InterBase(r), InterBase generally and Firebird.

<http://www.redhat.com> for cygwin.



Paul Reeves, the UK-BUG InterBase Group Leader, specialises in InterBase development and is now in charge of support at ibphoenix.

He can be contacted at paul@fleetriver.demon.co.uk