Bind 9 Dynamically Loadable Zone (DLZ) Project Plan

Objectives:

To add capabilities to Bind 9 that will allow Bind backend databases to support adding and removing zones without interrupting normal server operation.

History:

This project began because of a personal need. A larger personal project of my own requires the ability to dynamically add and remove zones from a running DNS server without the downtime associated with zone reloads, restarts etc.

I investigated several DNS server implementations (DJB DNS, Dents, DNS Java, etc) and found them all lacking or unsuitable for one reason or another. I finally began investigating Bind 9.

During the course of investigating Bind 9, I posted several messages to the Bind9-workers mailing list. My messages to the mailing list caught the attention of Teus Hagen, chairman and general director of NLnet.

NLnet is a non-profit organization committed to network (Internet) related research and development. As part of this commitment, they sponsor the development of software that would benefit the Internet and make that software available as open source.

Teus posed an interesting proposition: NLnet would sponsor (fund) the development of this capability in Bind 9 if I made its results open source and available to the world. The result of this proposition is this project.

Requirements:

1. **It will be open source.**
   Open source designates software for which the source is freely available for reuse. For the definition, see http://www.opensource.org here. This is an essential condition of funding, as the NLnet Foundation supports network software projects that are open for reuse. The source will be released under a BSD style license.

2. **It will use / re-use the existing Bind 9 database interface.**
   Bind 9 already has a database interface to support DNS record lookup in an external database. There is no need to replicate this functionality, and we can simply make use of it.

3. **It will not modify the existing Bind 9 database interface.**
   Bind uses the existing interface for the default in memory database. If we caused modification to the existing database interface, it would require even more changes to Bind's existing code base. This would require even more development

effort and debugging. Since it is possible to add the functionality we need to Bind without modifying this interface, we should do so. Also, minimizing the changes to Bind's existing code will make our changes even more likely to be merged into Bind's core.

4. **It will be merged into Bind 9 public source.**
   The goal of this project is to develop code that will improve Bind's functionality. Ideally, these changes will become part of Bind 9 and not need to be maintained as a separate patch. To that end, source code enhancements will always attempt to follow the current Bind 9 coding style and standard. However, we cannot guarantee that our modifications will become part of Bind. Therefore, this is not a requirement of funding because it is out of our control. The best we can do is to develop the code and present it for inclusion into Bind 9.

5. **(Optional) If possible, drivers should allow "dual mode" use.**
   Since the database drivers will be re-using the existing Bind database interface, it should be possible to have the driver operate in "two modes." Mode 1 would support the current Bind database interface without any extensions. Mode 2 would support our extended interface. This would eliminate the need to duplicate any effort between a dynamically loadable zone driver and a standard Bind database driver. This capability needs to be investigated further and is not a requirement of funding.

   By supporting two modes of operation, Bind administrators can choose to implement the full functionality of the driver and gain support for dynamically loaded zones, or only use a portion of the driver's ability and reduce the overhead involved with looking up a zone (By using Bind's existing in memory zone table for statically loaded zones).

6. **To the extent possible, the drivers should support multiple operating systems.**
   This will be limited to the developer's ability to test the drivers on multiple operating systems. However, every effort should be made to allow the drivers to operate on any system that the database it supports operates on.

7. **Dynamically loadable zone drivers may only implement limited functionality.**
   Dynamically loadable zone drivers re-use Bind's existing database interfaces. The standard Bind database interface supports both lookup and updates allowing RFC 2136 DNS update queries to make modifications to zone records. The SDB (Simple Database Interface) available in Bind 9 does not provide for DNS record updates.

   Generally, DLZ drivers will be implemented using the SDB interface and thus not allow RFC 2136 updates against zones using the driver. This is a limitation of the current SDB implementation.

Generally, it is undesirable for dynamically loadable zones to support RFC 2136 update or zone transfer capabilities. Updates to dynamically loadable zones generally take place through the backend database, not through DNS update queries. Also, DNS data replication of dynamically loadable zones should be handled through the backend database, and not by Bind zone transfers.

With the architecture we have in mind, it should be possible to add the same capabilities to dynamically loadable zones that standard zones have. However, this project's timeline does not allow for implementing anything more than basic lookup functionality of dynamic zones.

To be more precise, it is the goal of this project to enable records to be looked up in dynamically loadable zones (i.e. to be able to answer standard DNS queries to support domain name resolution). Further functionality such as zone transfer and support for RFC2136 Dynamic Update protocol will not be implemented as part of this project.

8. **It will be properly documented for both users and future developers.**

   Good documentation is essential for everyone. Unfortunately, Bind 9's source is not very well documented, and that makes understanding the code and modifications to it difficult. The modifications to Bind will be documented in the source as code comments in an effort to allow future developers to understand what is going on. Where possible, comments may be added which explain relevant parts of Bind 9's existing code.

   Additionally, this modification adds a significant new feature to Bind's capabilities. Ideally, these new features will be documented in Bind's administrator reference manual (ARM). Documentation of the new features will be created and forwarded to the appropriate persons for possible inclusion into Bind's ARM. Funding is not dependent upon the inclusion of the documentation into Bind's ARM.

   As an alternative, several "how to" documents could be created. The first of these documents would explain the dynamically loadable zone software architecture. The second would document the use of dynamically loaded zone functionality. Additional documentation will be created for each of the DLZ drivers we implement.

9. **At least 1 production level driver will be implemented.**

   This addition to Bind would be useless without a driver that supports the new functionality. This driver is required to interface between Bind and the external database.

   The PostgreSQL driver will be the first one developed. The PostgreSQL driver will make use of the new DLZ interface and Bind's existing SDB interface. Therefore, it will not support or allow RFC 2136 DNS dynamic update queries.

The driver will allow looking up zones in the database, as well as looking up zone records in the same database. Ideally, this driver will be included into Bind's distribution as a built in driver.

10. (Optional) An ODBC database driver will be implemented.

ODBC provides an interface to many different databases through a single interface. By developing an ODBC driver, we can support many databases with one driver. This driver will be of most use on Windows based systems where ODBC is widespread but can also be used on *nix machines that have ODBC support. The ODBC driver will make use of the new DLZ interface and Bind's existing SDB interface. Therefore, it will not support or allow RFC 2136 DNS dynamic update queries.

The driver will allow looking up zones in the database, as well as looking up zone records in the same database. Ideally, this driver will be included into Bind's distribution as a built in driver.

11. (Optional) A LDAP driver will be implemented.

Many larger ISP's may be interested in support for Bind's new dynamically loadable zone feature. Many of these larger ISP's use LDAP for login and other configuration information. An LDAP driver would allow Bind 9 to easily integrate with existing systems for these large ISP's. The LDAP driver will make use of the new DLZ interface and Bind's existing SDB interface. Therefore it will not support or allow RFC 2136 DNS dynamic update queries.

The driver will allow looking up zones in the database, as well as looking up zone records in the same database. Ideally, this driver will be included into Bind's distribution as a built in driver.

12. (Optional) A MySQL Driver will be implemented.

MySQL is another popular open source database. Evidence of interest in supporting DLZ drivers for MySQL has been witnessed on the Internet. Development of a MySQL driver would round out and more fully complete the built in drivers for Bind and DLZ. The MySQL driver will make use of the new DLZ interface and Bind's existing SDB interface. Therefore, it will not support or allow RFC 2136 DNS dynamic update queries.

The driver will allow looking up zones in the database, as well as looking up zone records in the same database. Ideally, this driver will be included into Bind's distribution as a built in driver.

13. (Optional) A Berkeley DB Driver will be implemented.

Evidence of interest in supporting DLZ drivers for Berkeley DB has been witnessed on Bind's mailing lists. The BDB driver will make use of the new DLZ interface and Bind’s existing SDB interface. Therefore, it will not support or allow RFC 2136 DNS dynamic update queries.

The driver will allow looking up zones in the database, as well as looking up zone records in the same database. Ideally, this driver will be included into Bind’s distribution as a built in driver.

Resources:

- A computer running Linux.
- A computer running Windows NT/2000.
- PostgreSQL database
- Berkeley Database (optional)
- ODBC Database (optional)
- MySQL Database (optional)
- LDAP server (optional)
- 1 monitor. Monitors shall review the design document and provide immediate feedback.
- Contact with the ISC, or a developer with commit authority to the Bind 9 source tree and documentation.
- Beta testers on a variety of operating systems. Beta testers will assist in finding and reporting bugs.
- Website and mailing list.

Existing resources available:

- At least 1 computer running Windows 2000.
- At least 1 computer running Linux.
- PostgreSQL database (open source)
- Berkeley Database (open source)
- ODBC Database (MS Access or Microsoft Data Engine)
- LDAP server (openLDAP open source)
- MySQL database (open source)
- 1 monitor (Andreas Gustafsson)
- 1 contact with ISC / developer with commit authority (Andreas Gustafsson)
- Website and mailing list. (Bind website or another project specific website, Bind9-workers and Bind9-users mailing lists.)

Development Timeline: (20 man weeks total, 40 hours / week. Approx 800 hours.)

Week 1:

Research existing Bind 9 code and determine how Bind currently operates. Determine if dynamically loadable zone functionality is possible and create preliminary design.

Week 2-3:

Using preliminary design, begin code changes to implement dynamically loadable zone functionality. Simultaneously, refine design of dynamically loadable zone (DLZ) interface.

Week 4:

Finalize design of DLZ interface. Finish adding DLZ code to Bind 9. Test Bind operation with new code, without using a DLZ driver to verify correct operation. Fix any bugs.

Week 5:

Create basic DLZ stub driver. Use stub driver to test operation of new interface and Bind operation with new driver. Fix any bugs in Bind.

Week 6-7:

Discuss and begin implementing DLZ driver internal architecture.

Week 7-8:

Finalize DLZ driver internal architecture and test the "frameworks" functionality. This becomes the basic framework for all DLZ drivers we implement. It could also become the framework for future drivers written by others. Begin implementing database specific code for PostgreSQL driver.

Week 9:

Finalize and alpha test / debug PostgreSQL driver.

Week 10:

Fix bugs in Bind or driver. Begin development of Berkeley DB driver.

Week 11:

Finalize and alpha test / debug Berkeley DB driver.

Week 12:

Fix bugs in Bind or drivers. Begin development of ODBC driver.

Week 13:

Finish and alpha test / debug ODBC driver.

Week 14:

Fix bugs in any code. Begin development of LDAP driver.

Week 15:

Finish and alpha test / debug LDAP driver.

Week 16:

Fix bugs in any code. Begin development of MySQL driver.

Week 17:

Finish and alpha test / debug MySQL driver.

Week 18:

Fix any bugs. Begin writing documentation on all parts.

Week 19-20:

Fix any bugs. Finish documentation. Fix any documentation errors.

The timeline above is in logical weeks. The actual development time will take much longer as the project will not be the primary focus of the developer.

Also, logical weeks 18 – 20 will be broken up and not completed at the end as detailed here. Documentation on each part of the project (core changes and each driver) will be created and delivered along with each part (core or drivers).

Optional drivers may not be developed in the order listed above. User interest, NLnet input, and the developer will determine the order of driver implementation.

Lastly, this project will be split into 2 parts. Part one will be the core changes to Bind and the development of the PostgreSQL driver. Upon the successful completion of part one, part two may begin. Part two will be the development of any of the optional drivers listed above, or any driver with sufficient interest.