Level: All
Works with: Notes/Domino 6
Updated: 03-Sep-2003

by David Byrd
and Timothy Speed

In Part 1 of this article series, we presented a brief description of how Notes templates work and what you can do with them. We covered controlling design changes and setting ACLs and explained how you can retrieve design information with a tool we provided with the article (available for download from the Sandbox). In Part 2, we conclude with a discussion of template best practices, troubleshooting tips, and new features, such as Single Copy Template and Seamless Mail Upgrade.

We assume you have some familiarity with Notes/Domino programming, although novices should also find much of this information useful. Also, we suggest reading the *LDD Today* article "Enhancements to Notes/Domino 6.0.1 and 6.0.2" for descriptions of all the latest template features.

Template best practices

This section offers a few recommendations for template best practices, including domain structure, naming conventions, and revision tracking.

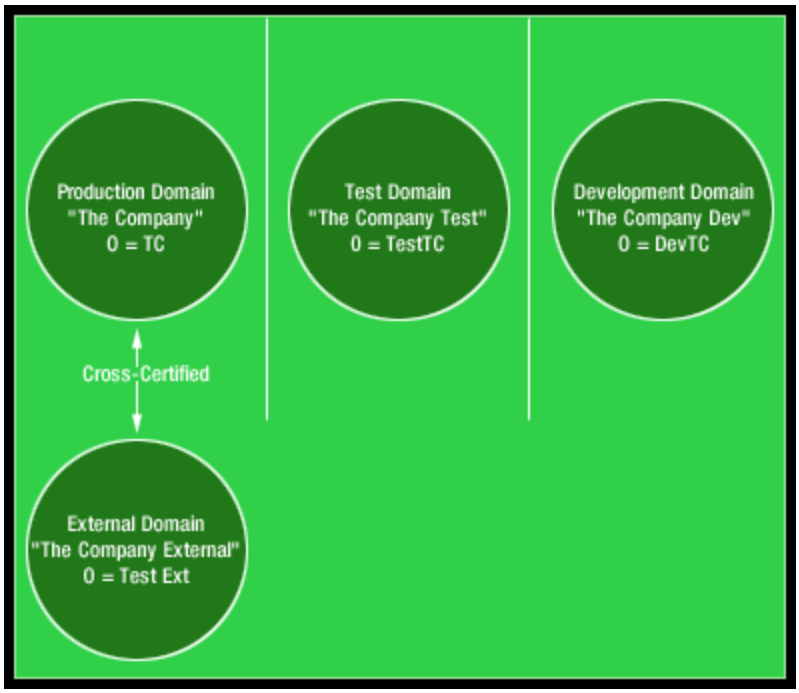
Domains

An important component of good template management is the efficient use of domains. A *domain* in Lotus Notes is a group of servers and clients that share the same Domino Directory, and thus share configuration information. This includes user, server, and group definitions. The Domino Directory controls the domain's internal and external Notes communications.

A Notes domain can also be used to control a logical collection of applications. Many companies have several domains, organized along corporate functional lines, for example, Production, Development, and Test domains. This type of structure helps you manage applications, making it easier to secure and deploy applications, as well as manage design changes.

We have often seen a production application experience some type of unidentified problem. After some forensics, we quickly discover that a well meaning developer went into the production copy of the application and made a change to one of the design elements, causing the application to break. This simple developer's change can quickly replicate throughout the domain, causing the application to stop working. This problem can be easily avoided with adequate controls in place to test the application in a "roped-off" environment—in other words, a test domain. This is why we recommend you keep your production templates isolated from your test or development templates. Also, remove any ACL entries for developers in the production domain to avoid unauthorized or unexpected changes to your production applications.

Here is an example of a recommended structure for testing and deployment of new applications:



In our hypothetical organization, the Production domain is isolated from the Test and Development domains. Each domain has its own organization-level certifier. The Production domain communicates with the External domain. Each domain has a defined responsibility. The Production domain is where the applications live for your users, customers, and/or general public. The Development domain is used for developers to generate new applications or to update existing applications. The Test domain is for testing the new or updated applications.

Template naming
Now that we have the domains defined, we need to start developing our applications. First, we need to create some simple naming standards:

- *File names*
  We recommend that the file name follow this structure:

  [first eight characters of the application or template name] plus [application version number] plus [NTF or NSF extension]

  For example, the file name for the second version of the Accounting Tracking template is Accounti002.ntf, where Accounti is the first eight characters of the template name and 002 indicates the version.
- *Template titles*
  This should consist of the application name plus the version number. For instance, in the previous example, the template name is Accounting Tracking 002.
- *Template inheritance name*
  This can follow the structure:

  [department code] plus [first eight characters of the application name]

  Using the previous example, the template inheritance name is V123Accounti, where V123 is the department code.

Template tracking
You can use your domain structure and naming conventions to help track template revisions. This ensures that each template belongs to the appropriate domain and is isolated from the others. In our example, the production

template is Accounti002.ntf, but the developer is working on Accounti003.ntf. In this case, the production template is in the Production domain, and the template that is being updated is in the Development or Test domain. The workflow for this process may be similar to the following:

1. A change to the application is requested via a change control system.
2. The change is approved, and a developer is assigned.
3. The developer, using a template management system, checks out Accounti002.ntf from the archive.
4. The developer makes the changes and saves the template as Accounti003.ntf.
5. The developer creates some test scenarios. The developer then requests that the template be placed in the Test domain for testing. The test team uses the test scenarios to test the changes to the application.
6. If all tests are successful, the application is staged for final acceptance. A complete review of an acceptance process is outside the scope of this article, but at a minimum the acceptance checklist should include signing the application (for ECL implementations; see also this *Iris Today* article), checking file name and template inheritance name, updating the change control system, including special instructions for whatever existing data needs to be updated or changed, and creating a "backout" plan in case the application fails in production.
7. After the acceptance process and change control are completed, the application template is moved to the production server. At the designated time (typically a low-traffic date/time, such as Sunday at 1:00 AM), the production application is refreshed with the updated template.

Note that many administrators remove the Design task from the Notes.ini variable ServerTasks on their servers. In this case, a designated server (for example, an application "master" server) is used to refresh the application design. This master server contains all applications and associated templates. The template only exists on this server and is refreshed only on demand (via a workstation), so the Design task is never executed. One big advantage of this method is that you can block designers from both the template's ACL and the application's ACL. After the application has been moved to the production server, you can remove ACL access for the developer.

Here is an example tracking form that you can use to track application updates.

Application information:
Application name: _____
Version number: _____
Application description: _____
Template name (if this application is a template): _____
Name of template this application inherits from: _____
Template file name: _____
Release date: _____
Type of release (New, Bug Fix, Enhancement): _____
Database owner: _____
Server name: _____
Directory name: _____
Developer name: _____

Application Status:

- Is this a design replace or a refresh?_____
- Is this action approved by change control? Y/N
- Has the new release passed acceptance testing? Y/N
- Have you verified that the application is off-line and/or that users are not using the application? Y/N

Questions:

- Is the "Do not allow design refresh/replace to modify" option in Design Document Properties box deselected? Y/N
- Is the application version number updated in the elements (icon, navigators, help, and Using This Database document)? Y/N
- Are there any documents that need to be copied to the application after refresh or replace? Y/N

- Has the developer provided instructions for configuration and/or profile documents? Y/N
- Are there any agents that need to be executed after the application has been updated? Y/N
- Have all test forms/views/fields been removed? Y/N
- Does the ACL and/or roles need to be modified (per developer instructions)? Y/N
- Has the current release of the application been backed up? Y/N
- Is there an application upgrade backout plan? Y/N
- Have users been notified about the new release? Y/N
- Has the developer name (or group) been removed from the ACL of the template and application? Y/N
- Has the template been signed by the required signature ID? Y/N

As with any development resource, managing templates can be a chore if you let it. Templates themselves should be treated just as you would any other body of source code. They can easily be versioned for change control purposes. Templates can most effectively be managed by including version numbers into the template and file names as discussed earlier in this article. The ACLs should be locked down to prevent unauthorized access and in some cases, should be removed completely after the update has been performed. In addition to these manual management techniques, there are also third-party tools available (such as TeamStudio CIAO! from Ives Development) that can help you track template revisions. You may want to explore these tools for use at your own site.

Troubleshooting templates

Template troubleshooting can be a fairly simple task. There are only a few things that can go wrong, and each has fairly simple fixes. Each situation described in this section involves a common problem that Lotus Notes professionals will likely run into sooner or later:

- *When the Design task ran on the server, elements just appeared or disappeared.*
  This can happen when the Design task runs nightly on the server. If you're not careful with template versions, unexpected design changes can occur. Also remember that design templates replicate just like any other database. Watch out for old copies causing design elements to come back from the dead.

- *The design elements are not updating correctly.*
  A change was made in the template, but that change is not being made in the parent database. This can happen if:
  - The design element is locked from changes.
  - The database level Inherit from template name setting does not match the base template.
  - The design element level Inherit from template name setting does not match the base template.
  - Multiple database templates share the same template name.

- *Documents are showing up in my database after I created it from the template.*
  This can happen if the developer left documents in the template. There are some cases in which this is desired. For instance, if a keyword list view needs to be pre-populated in your application, then those documents can be added to the template.

- *The ACL for the database created from the template ACL is not set correctly.*
  This can happen if the template designer did not place brackets around each ACL entry to be carried to the new database.

- *The ODS for the templates are 20 (R4 format) and not 41 (R5 format) or 43 (Notes/Domino 6 format) like the rest of the databases.*
  ODS20 is the default On-Disk Structure for templates. The main reason for this is space savings compared to the new R5 or Notes/Domino 6 ODS format. It does not cause any loss of functionality beyond having a 4 GB size limit. This is generally not a problem with most templates.

- *Duplicate design elements appear in the application after the Design task runs.*
  This can happen if you copy and paste design elements into a template. The Design task updates design elements based on the element name and a matching unique identifier (UNID). When you paste a design element, the UNID is changed. (Note that in general, copying/pasting design elements is a valid

development technique. However, you do need to exercise caution in some cases. For example, suppose you have a form named "Document" in your template, and you need to make a large number of design changes to it, so many that you don't want to make them simultaneously within the template. So to test, you make a number of these changes in a copy of the template. Then you delete the form "Document" from your template and paste the updated "Document" form from the test database into the template. Now you have the same-named form in the template, but it has a different UNID from the original template.)

Note that the best place to debug template update issues is in the status bar of the Notes client (for local refr eshes) or the server log (log.nsf) for nightly design updates. Each design element added, updated, or removed is displayed along with the template these changes originated in.

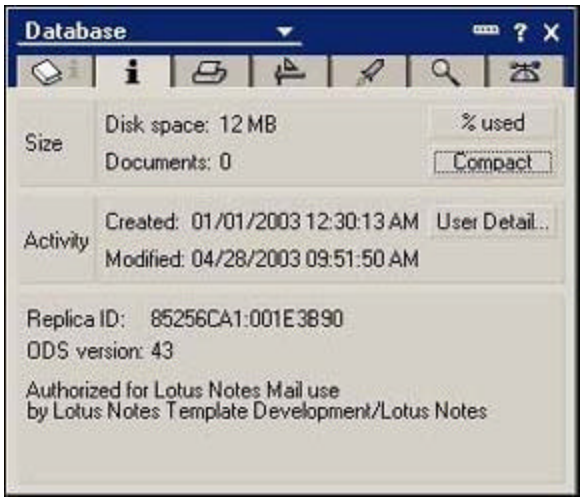
Of course, the best form of troubleshooting is to prevent problems from occurring in the first place. Follow these simple rules to remain out of trouble:

- Use naming standards for your templates.
- Keep test and development templates separate from production templates.
- Consider using tools to manage your templates and applications.
- Use change control to manage your application environment.
- Know which properties do and do not refresh from a template.
- Back up your design templates—you never know when you need the original copy!
- Make changes to existing design elements. Do not copy elements and delete the original element.

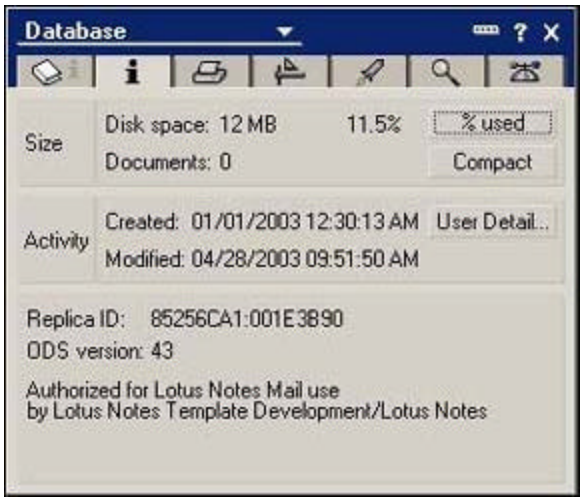
Single Copy Template

Lotus Domino 6.0.1 implements a new feature called Single Copy Template (SCT). This feature allows multiple databases to share design elements with a separate database. For example, in a classic Lotus Domino configuration each user's mail databases has a full set of design elements, along with mail data adding to the overall size of each database. Depending on the release of Lotus Domino, this design can add between 7 to 11 MB of space. SCT changes this by linking the design elements of all mail databases to a single database.

For example, suppose a mail database was created using the StdR6Mail template. The following shows the size of this example mail database with no documents.



For the space saving benefits of SCT to be fully realized, the database must first be compacted. Notice in the screen below that after the database is converted to a SCT-enabled mail database, only 11.5 percent of the space is used even though the database itself is still the original 12 MB:



We then compact the database, reducing its size to approximately 1.3 MB, saving 10.7 MB in storage space:

```
04/28/2003 09:53:43 AM Compacting mail\admin.nsf (Admin)
04/28/2003 09:54:02 AM Compacted mail\admin.nsf, 10752K bytes recovered (87%)
```

Multiply this by the number of mail files in your domain, and you have an idea of how much disk space SCT can save!

The steps to enable a mail (or any other) database to use SCT are as follows:

1. Create a copy of the current mail template and give the copy a new name, for example mail6sct.ntf or inotes60sct.ntf. It's also a good idea to change the database title to indicate this is a “special” template. For example, you can append "SCT" to both the file name and database title.
2. Enable the template as a Single Copy Template by opening the Database Properties box and selecting the Single copy template option.
3. Modify the design of the target database(s) to use this new template. You can do this with the Convert server task or by selecting File - Database - Replace Design. The following Convert statement updates the design of the mail database located under the mail subdirectory:

   ```
   >load convert mail\*.nsf StdR6Mail mail6sct.ntf
   04/28/2003 10:01:59 AM Mail Conversion Utility starting
   04/28/2003 10:02:29 AM Mail Convert: Started replacing design template 'StdR6Mail' with 'StdR6MailSCT' in 'mail\admin.nsf'
   04/28/2003 10:03:00 AM Mail Convert: Finished replacing design template in 'mail\admin.nsf'
   04/28/2003 10:03:00 AM Mail Conversion Utility shutdown
   ```

4. Execute the Design task on the server console. This removes the design elements from the mail database and replaces them with pointers to the Single Copy Template:

   ```
   >load design
   04/28/2003 10:04:05 AM Database Designer started
   04/28/2003 10:04:30 PM Database Designer shutdown
   ```

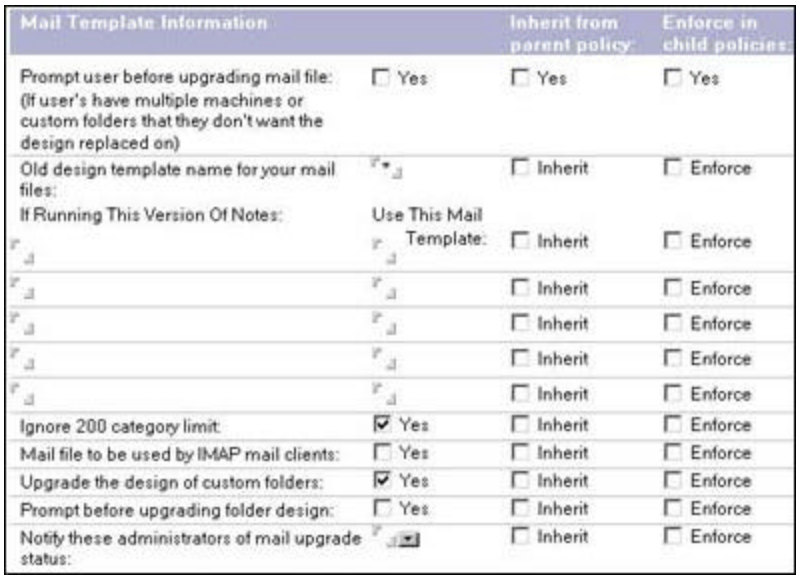
5. Compact the database to remove the additional white space.

Seamless Mail Upgrade

Another key template upgrade feature in Notes/Domino 6 is Seamless Mail Upgrade. This allows you to define base levels of mail database templates across your site by defining a Desktop policy. If a user accesses a server outside this baseline, Notes/Domino 6 automatically corrects the problem. (In R5, this requires a tremendous level of effort.) Policy and Setting construction are beyond the scope of this article, but are described

in the product documentation and help database in more detail.

To implement Seamless Mail Upgrade, create a Desktop setting document in the Domino Directory (or modify an existing one). In the Desktop setting document, populate the following fields as appropriate for your organization:



Mail Template Information		Inherit from parent policy:	Enforce in child policies:
Prompt user before upgrading mail file: (If user's have multiple machines or custom folders that they don't want the design replaced on)	☐ Yes	☐ Yes	☐ Yes
Old design template name for your mail files:		☐ Inherit	☐ Enforce
If Running This Version Of Notes:	Use This Mail Template:	☐ Inherit	☐ Enforce
		☐ Inherit	☐ Enforce
		☐ Inherit	☐ Enforce
		☐ Inherit	☐ Enforce
		☐ Inherit	☐ Enforce
Ignore 200 category limit:	☑ Yes	☐ Inherit	☐ Enforce
Mail file to be used by IMAP mail clients:	☐ Yes	☐ Inherit	☐ Enforce
Upgrade the design of custom folders:	☑ Yes	☐ Inherit	☐ Enforce
Prompt before upgrading folder design:	☐ Yes	☐ Inherit	☐ Enforce
Notify these administrators of mail upgrade status:		☐ Inherit	☐ Enforce

After completing the Desktop setting document, define a policy covering the users to whom you want the policy to apply. When users access the server, their mail template is automatically upgraded (if so configured).

The truth about templates

This concludes our two-part tour of Notes/Domino templates. We've introduced you to some basic template concepts, how they work, and what you can do with them. We provided a tool to help you retrieve design information and covered some template best practices. You should now have a good grounding in "template technology"—but there's still a lot we haven't covered. So the best course of action is to set up examples to see for yourself how the various new template features and options work. Good luck, and let us know your experiences!

ABOUT THE AUTHORS

David Byrd is a Consulting IT Architect with IBM Software Services for Lotus (ISSL) from Atlanta, GA. David is fluent in virtually all areas of Lotus products and technologies ranging from C/C++ API development and application, security, and messaging architectures with heavy focus on enterprise-level messaging and directories. He has worked with Lotus Notes and Domino since the early 1990's and holds numerous certifications from Lotus, IBM, Redhat, Microsoft, and Novell. You can email David at david_byrd@us.ibm.com.

Timothy Speed is an infrastructure and security architect for IBM Software Services for Lotus (ISSL). Tim has been involved in Internet and messaging security since 1992. He also participated with the Domino infrastructure team at the Nagano Olympics and assisted with the Lotus Notes systems for the Sydney Olympics. His certifications include MCSE©, VCA (VeriSign Certified Administrator), Lotus Domino CLP Principal Administrator, and Lotus Domino CLP Principal Developer. Tim has also co-authored four books: *The Internet Security Guidebook* , ISBN: 0122374711, February, 2001; *The Personal Internet Security Guidebook* , ISBN: 0126565619, October, 2001; *Enterprise Directory and Security Implementation Guide: Designing and Implementing Directories in Your Organization* , ISBN:0121604527; and *Internet Security: A Jumpstart for Systems Administrators and IT Managers,* ISBN 1555582982. You can reach Tim at Tim Speed@us.ibm.com.

