CHAPTER 10: DEPLOYMENT

Objectives

The objectives are:

- Get Steering Committee approval.
- Establish the right configuration.
- Perform data conversion.
- Go live.
- Do a project evaluation.
- Perform ongoing support.

Introduction

The deployment phase marks the end of the implementation phase. During deployment, the final preparations are made and the solution is installed at the client site.

There are a number of preconditions for this phase:

- The new solution must be completely developed and tested.
- Any errors must have been identified and resolved.
- The various analysis and design documents must be updated for any changes.
- The infrastructure must be in place.
- The users must be trained and knowledgeable about the new module they will be using after deployment.
- The user documentation must be prepared and ready for use.

Once these preconditions are in place, the seminar modulecan be deployed.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

10-1

Deployment Tasks

Before going live with the new module at the client site, there are a few tasks that must be accomplished.

Steering Committee Approval

Conduct a final walk-through of the new module with the Steering Committee to obtain their final approval for delivery.

Configuration Checklist

Go through a checklist of all the configuration elements that must be in place before the module can be used. For the Seminar module, be sure that the various new number series have been created and that the seminar setup is properly completed.

Data Conversion

During data conversion, perform the transfer of existing customer data into the new seminar module. This includes master file data as well as ledger entries and transaction documents.

Importing Document and Ledger Data

There are generally three types of data that are imported into Microsoft™ Dynamics NAV:

- Master or supplemental table data
- Transaction document data
- Ledger data

Importing the master data is fairly straightforward, and usually does not require any code to be written.

Importing transaction documents is more complicated. Because of the complex nature of the header and line data, and because the amount of data per line can sometime

s be beyond the limits of a dataport, it is not recommended to import documents. Historical data stored in the ledger entries is usually sufficient. However, if it is necessary to import open orders, use a dataport to import two files, one for the headers and one for the lines, making sure to validate the fields. If some orders are partially shipped, only import the portion of the order that is still open.

If necessary, import historical, or posted, documents. The process for doing this is simpler because the fields do not need to be validated since they are for viewing only, and line numbers are entered sequentially, since the user cannot insert new posted document records.

NOTE: When importing ledger or balance data, the most important thing to remember is to never import directly into a Ledger table. Instead, import the data into an appropriate Journal table. Either leave the entries in the Journal table to be reviewed and posted later or, if more testing is performed up front to ensure that the data is correct, post the Journal entries as they are imported.

Note that the validation of some fields changes the values of other fields. For instance, when the **Account No**. field is validated, the validation code fills in the **Description** field. If the **Description** had already been imported, the imported value is lost. To solve this problem, create a global variable and put that variable's identifier into the SourceExpr property of the **Dataport** field rather than the actual field name. Then, after calling the validation of the **Account No**. field, transfer the **Description** from the variable to the field.

Always fill in the **Source Code** field with a unique value for each import routine. This makes it possible to see where certain data in the journals and ledger entries came from.

When importing document or ledger data, write code to check that the data follows Microsoft Dynamics NAV business rules, to initialize the records, and to fill in any missing data.

Dataport Event Triggers

The events for dataport triggers revolve around the processing of data items and the import and export of records. The initial triggers that run before the data items are processed are:

- OnInitDataPort
- OnPreDataPort
- OnPreDataItem

OnPreDataport can be used to validate options entered by the user on the request form. OnPreDataItem can be used to insert code that is run once for the entire Dataport.

To write code that fires when records are exported, use OnBeforeExportRecord and OnAfterExportRecord triggers.
To work with importing records, use the import triggers:

- Use **OnBeforeImportRecord** to initialize the record to be imported.
- Use **OnAfterImportRecord** to validate the information being imported, fill in missing data, and to either store the record or post it.

To write code that fires after a data item has been processed, use the **OnPostDataItem** trigger, and to write code to fire after all data items have been processed, use the **OnPostDataPort** trigger.

Lab 10.1 - Creating Contact Dataport

The client wants to import contacts into the new seminar module. Create a simple Dataport by following these steps:

1. Create a dataport called Import Contact (123456700) with the **Contact** table as the **DataItem**.

2. In the Dataport fields, choose to import the following fields:

   - No.
   - Name
   - Type
   - Address
   - Address 2
   - City
   - Country Code
   - Phone No.
   - Company No.

3. Write code to update the **Currency Code** based on the **Country Code**. The countries that are set up are detailed in the following table.

Country Code	Currency Code
AT	EUR
BE	EUR
CA	CAD
DK	EUR
NL	EUR
US	USD

Go Live

At go live time, the system becomes available to users for normal daily transactions. After the system is up and running, pass the administration of the module over to a client manager.

Project Evaluation

As the final deployment task, perform a project evaluation with the project team to assess the key aspects of the project planning and implementation. Some of the factors to evaluate are:

- Cooperation with the client and within the project team
- Utilization of the project team skills
- Project planning
- "Highlights" and "lowlights" of the different project phases
- Client satisfaction

Ongoing Support Phase

After deployment, the project enters the ongoing support phase. As a Microsoft Dynamics NAV developer, there are two primary activities to perform during this phase:

- Implementing new customer requirements
- Updating for new releases

Each new development project should follow the Microsoft Dynamics NAV Implementation methodology used in developing the seminar module. Examine what is involved in the upgrade process.

Reasons to Upgrade

There are two important reasons to upgrade a customer's Microsoft Dynamics NAV installation.

The Customer has already paid for it: Many customers have an upgrade agreement in effect from the time of sale. Thus, many customers have paid for and expect an upgrade.

The Customer will pay for it: Some upgrades are important enough that the customer wants them even if they have not paid for them in advance with an upgrade agreement. Also, even with an upgrade agreement, the customer is only purchasing the product upgrades and not the installation of the product upgrades. The installation fees could be worthwhile, if upgrades are planned for and implemented properly.

The customer benefits from upgrading for these reasons:

- **Gain access to new features**: Customers can gain access to new features, such as record links and SMTP support.

- **Improvements to existing features**: In version 5.0, there are improvements to some features If customers want any of these improvements, they need to upgrade.

- **Improved performance and removal of problems**: Performance is improved and problems are removed with each upgrade.

- **Allow upgrades to new operating systems and hardware**: For example, when Microsoft Dynamics NAV Financials version 2.60 was released, it added support for Microsoft® Windows® 2000, did not include support for Microsoft® SQL® 2000.

- **Better support when on the current version**: Customers tend to receive better support when they are on the current version. As versions become obsolete, fewer people have expertise in that version.

A solution provider benefits from upgrading for these reasons::

- **Better customer relations**: Customer relations are built on providing services to the customer. An upgrade is an opportunity to provide a service to an existing customer that is NOT in response to a customer complaint. It also provides a chance to talk with customers and find out what new needs they may have.

- **Easier to support when the customer is using the current version**: In many cases, the solution to a problem a customer has is found in a newer version. In other cases, it may be hard to support an old product when nobody at an organization can remember it.

- **Remove problems before the customer notices them**: When customers find a problem that turns out to be caused by a bug, they often want the problem to be fixed for free, and even if they get that, they may be irritated with the solution center and with the product. If an upgrade fixes a bug that customers have not run into yet, they pay for it with the upgrade, and there is one less opportunity for them to be dissatisfied.

- **Opportunities for additional sales**: Microsoft Dynamics NAV often includes new features that the customer may be interested in using. However, to get these new features, the customer must upgrade, which generates service revenue.

Definitions

Before discussing the types of upgrades, it is important to understand the following terms:

- **Executables**: The Microsoft Dynamics NAV programs that run under the operating system and include the client, the server, the C/ODBC drivers, and so on. They can be identified in the Windows or NT Explorer because they are files that end with ".exe" or ".dll".

- **Application**: The programs that run within the Executables, more specifically within the client, which is known as C/SIDE. The Application is made of Microsoft Dynamics NAV objects.

- **Functional Area**: A major division within the application, like General Ledger or Inventory, that can be identified by the fact that there is a Navigation Pane Menu item that identifies it and by the sub-menu that appears when this button is pressed.

- **Granule**: A set of objects that is purchased separately which contains a set of features.

- **Feature**: A small set of functions that are part of the application. The General Ledger is a functional area within the application, Account Schedules is a granule, and Date Comparison is a feature of Account Schedules.

- **Bug**: A piece of the program that does not work as intended. Note that if a piece of the program is not working as expected, that does not make it a bug. Only if it does not work as designed by Microsoft Dynamics NAV is something considered a bug.

- **Enhancement**: An enhancement is when an existing feature is made to work better in some way: easier to use, faster, performing additional functions, and so on. If a feature is merely made to work as it was originally intended, then that is a bug fix, not an enhancement.

Types of Updates

There are different categories of software updates which may need to be installed:

- **Hotfix**: A single cumulative package composed of one or more files used to address a problem in a product. Hotfixes address a specific customer situation and may not be distributed outside that customer organization.

- **Release**: Normally scheduled product release that includes the upgrade toolkit.

- **Update**: A broadly released fix for a specific problem addressing a non-critical, non-security related bug. This can include critical updates and feature packs.

- **Critical Update**: A broadly released fix for a specific problem that addresses a critical, non-security related bug.

- **Service Pack**: A tested, cumulative set of all hotfixes, security updates, critical updates, and updates, as well as additional fixes for problems found internally since the release of the product. Service packs may also contain a limited number of customer-requested design changes or features.

- **Feature Pack**: New product functionality that is first distributed outside the context of a product release, and is usually included in the next full product release.

Planning for Upgrades

The most important step in the upgrade process is to plan for it in advance. When Microsoft Dynamics NAV is installed on a customer's system, the customer will upgrade eventually, so plan for it.

Start this planning when making the initial implementation plans. Many times, an upgrade may be made very difficult because of a decision made when first implementing the customer's system. The largest area is in customizations. Other areas are also important. For example, an upgrade of the executables usually requires changes on every client machine. Throughout the implementation process, keep customers informed of upgrade considerations. For example, if they do not know that upgrades will bring their system down for a while, they will be irritated when an upgrade is done. If they know in advance, they can help plan by indicating when might be the best time to upgrade.

There is no overstating the importance of keeping good records. It is important to know what is on every customer installation, so that:

- The appropriate decisions can be made about whether to install an improvement.

- The correct plans can be made about how long an upgrade will take.

- Customers can be asked about whether they want certain new features.

- It is known when an upgrade will override an existing customization.

Customizations

The most important planning for upgrading involves customizations. Consider how upgrade customizations will be performed whenever the customer's software is modified. The best way to do this is to use the recommended methodology.

It is recommended that considerations for upgrading with every design specification be included. In addition to helping with planning, this also helps customers to make intelligent decisions. If they know that a particular design will mean extensive upgrade problems every time, they may opt for an alternate design, even if it costs more money at first.

Also, using low impact programming techniques are part of the Attain Development course. Only make changes to the base application when necessary, and then make them in a way that is easy to upgrade. In addition, document every change. This is not optional – it is essential. A smooth upgrade is virtually impossible without knowing what is on the customer's system.

If the customer has development tools, that can lead to additional problems. Teach the customer to use the same documentation and modification techniques learned when programming.

Scheduling

When it is time to perform the upgrade, the planning starts with scheduling. Once it is decided that an upgrade will be done, start planning a schedule with the customer. There are several reasons for this.

The Microsoft Dynamics NAV Server (or SQL Server) must be taken down every time an upgrade is performed, no matter how small. Even if there is not an executables upgrade, there is an object cache on every client. In addition, a backup must be performed before every upgrade, no matter how small. Treat every upgrade as though a new system is being implemented.

Because of the need to take down the server for a period of time, it may be necessary to install the upgrade during off-hours, or even over a weekend, if there is a data conversion as part of the upgrade.

There are several things that you can do to relieve some of these time issues and make the upgrade less painful for the customer, but they require planning and scheduling. First of all, enlist the support of the customer's IT department for anything that they can do. For example, the customer's IT department can see that the server is taken down and that backups are performed in advance. Furthermore, they can also distribute executables changes, and communicate with the users about the upgrade. Second, do as much as possible before arriving. For example, upgrade all objects, write and test the data upgrade routines, and finally test the installation. When all this is done, bring everything to the customer's site for execution.

Upgrading the Executables

If there is an upgrade to the executables (the Microsoft Dynamics NAV programs that run on the operating system), this part of the upgrade is done first. It can be scheduled separately from any other steps. For example, the executables portion of a release can be done one weekend, and the upgrades for the objects or data might not be scheduled for another two or three months.

Upgrading the executables is very easy, as it normally just involves copying the new files over the old ones. However, when there are 50 or more client machines to upgrade, this can take a lot of time.

Sometimes, the internal database structure can change. When this happens, make a Microsoft Dynamics NAV backup under the old system, and then restore from the backup after installing the new system.

The first step is to determine what is being upgraded. This can be an upgrade of the Microsoft Dynamics NAV Server, the client, or both. It can also involve other executables, like N/ODBC. Sometimes, a new customer license is required. If so, be sure to order one before proceeding. Read the Upgrade Toolkit documentation carefully to determine what exactly is upgraded. Note that there are special instructions when upgrading the SQL Server Option.

Microsoft Dynamics NAV Server

Follow these instructions if an upgrade to the Microsoft Dynamics NAV Server is needed:

1. If a backup is needed, make the backup while the old system is still running. This must be a complete backup – all companies, data common to all companies, and application objects. Then have everybody log off the system, and bring down the Microsoft Dynamics NAV Server. Now, make a complete backup copy of all Microsoft Dynamics NAV files (executables and database) using tape or another fixed disk drive. This backup is used if anything goes wrong with the upgrade.

2. Install the new server components. This normally involves copying the new files over the old ones. Sometimes it requires a new installation, but just install the new version over the old version. Normally, there is a client also located on the server machine. If so, and if there is an upgrade to the client executables, then do this now. If there is a new license involved in the upgrade, copy it into both the server directory (the one which contains server.exe) and the client directory (the one which contains fin.exe).

3. If a Microsoft Dynamics NAV backup was required due to a database structure change, delete the old database, bring up the Microsoft Dynamics NAV client on the server machine, and create the new database (using the same name and location as before). Restore the complete Microsoft Dynamics NAV backup into this new database. Bring down the server machine client.

4. Bring up the server, and bring up the server machine client. Log on to the server from the server machine client. Run a few simple tests to make sure that the system has been properly restored and that basic functionality still runs.

5. If there are executable upgrades for clients, bring down the server and upgrade all of the clients. This is covered in the next section. Once this is done, bring the server back up.

Microsoft Dynamics NAV Clients

If there are executable changes for a client, the steps are quite simple:

1. Bring down the client (exit Microsoft Dynamics NAV). Copy the new client files into the client directory, or re-install the client, depending on the upgrade kit instructions.

2. If there are other executable upgrades, like N/ODBC, C/FRONT, and so on, then install them now as well. The upgrade kit includes specific instructions.

3. If a new customer license is required, copy it into the client directory at this time. Then bring the client up.

Repeat the above steps for each client in the installation. If there are many clients, enlist the customer's help in doing this part of the upgrade. The steps are simple, but must be repeated on each system.

If the server executables are also being upgraded:

1. Bring down all clients.

2. Upgrade the server.

3. Upgrade all clients.

4. Bring the server back up.

5. Bring all clients back up.

Upgrading the Objects

The Object Upgrade is the most difficult part of any upgrade, no matter what the category of software update. This is because the following occurs:

1. Microsoft Dynamics NAV in Denmark makes worldwide changes to the application objects.

2. The NTR makes changes to these same application objects.

3. You make custom changes to these same application objects. In fact, if customers have the design tools, they can make their own changes to these same application objects.

Each NTR is responsible for merging Microsoft Dynamics NAV changes with their changes (if applicable). This is why it takes time between the worldwide release of a version and the local NTR release of that same version. Merge these changes with the changes made during customization. Following the rules for documenting customizations is always a good idea. Version tags and modification flag rules are reviewed as the upgrade steps are followed. (For a review of versioning, consult the Microsoft Dynamics NAV Development I manual).

The basic process is to go through each individual application object to be reviewed. For this example, use the **Customer Table** table (18) as a stand-in for any application object.

For each object, there are three potential versions.

- The old base version, which is on the CD from which the installation was performed. It is important to document what version was installed.

- The new base version and also the old customized version, which is what is currently running on the customer's system.

- What is desired when the upgrade is finished, is a new customized version of the object, which includes all of the customizations, plus all of the changes made for this upgrade.

For each version issue, there are four options:

- There are no changes to this object. In this case, the object on the customer's system does not change during this upgrade.

- The object has been customized, but do not change it for this upgrade. Again, in this case, the object on the customer's system remains unchanged for this upgrade.

- The object for the upgrade is changed, so there is a new base version, but this object was not customized. In this case, the new object replaces the object on the customer's system.

- The possibility that causes the problems – customizations were made to the object, and the object as part of the upgrade was also changed. In this case, first determine which version changed the object the most, and use that version. Then, change that object so that it includes all of the changes that were made by the other version.

  1. For example, the upgrade obtained contains more changes than were made during customization. Therefore, get the object that obtained from the upgrade, re-do the customizations on that object, and then replace the object on the customer's system with the new, customized object.

  2. Another example is when the customizations made contain more changes than those for the upgrade. When this happens, get the object from the customer system, re-do all of the upgrade changes on that object, and then replace the object on the customer's system with the new customized object.

A Typical Upgrade

Go through the typical steps to upgrade objects. In this situation, a few small changes where made for the customer.

Assume that all of the recommendations for version tags have been followed –all modified objects have been tagged with an additional version tag (use CR01), and that all modification flags are turned off. In addition, the customer has no design tools; therefore, it is not necessary to go to the customer site to pick up the objects, since the customer has not changed them. A copy of the customer's database has been kept in-house. This copy just has objects, with either no data or minimal (Cronus) test data. There may also be a copy of the target version application.

Step by Step

1. Make a working copy of the customer's database (objects only). Create a Microsoft Dynamics NAV backup, create a new database, and then restore application objects only.

2. Get the improved objects from the target version database. To do this, go into the Object Designer and click the **All** button. Then put the cursor in the **Version Tag** column, and set a filter to the NTR's code for the new version. This filters out all but the improved objects. Then select all objects (CTRL+A) and export. Export as a Microsoft Dynamics NAV object file, with an extension of .fob.

3. Go back into the working copy database and import the Microsoft Dynamics NAV object file into the Object Designer. Normally, there will be conflicts, but even if there are not, bring up the Object Import Worksheet anyway. Check all of the objects, find the conflicts, and write down the object type and ID in a conflict list. Skip any objects that have a conflict for this step. For the remaining objects, there is no conflict, and the new objects simply replace the old ones since the old ones were not customized.

4. Review the four possibilities again:

   – First, an object might not have changed in either the customized version or the new base version. In this case, only the changed objects were imported, so the object would not have imported. These do not show up in the Import Worksheet.

   – Second, the object might only have changed in the customized version, but not in the new base version. In this case, the object is not imported, since only new base objects are imported. Again, these do not show up in the Import Worksheet.

–   Third, the object might have changed only in the new base version, but not in the customized version. These objects are in the Import Worksheet, but they show up with no conflicts. They can just be imported.

–   Fourth, the object might have been changed in both the customized version and the new base version. These objects are in the Import Worksheet. In this case, they show up with a conflict. Skip these objects (do not import them) and write them down in a conflicts list.

5. Review the conflicts list. Look up each object in the conflicts list in the Change Log that came with the new version. Remember to check all the improvements, since it might have been modified more than once. Look at the Change Log in the Latest Improvements page, where they are all together in one log, and start at the bottom with the version in the Import Worksheet. This makes all of the changes easier to find.

6. For each object in the Conflicts List, consider if the upgrade modification is small. More to the point, is it smaller than (or even almost as small as) the change made for the customization?

    If the answer to this question is yes, then use the Change Log that came with the new version to implement the improvement on the customized object in the Working Copy database.

    If the answer to this question is no, then import the new base object into the working copy database, and then use the Change Log to implement the customization on the new base object.

    If there is no change log, use the Compare tool to create one. This makes it easier to decide which change is smaller.

    Either way, set the version tag to reflect both new versions. This shows both the new NTR Version Tag and the Customization Version Tag.

    There is one exception to these rules. If this is a table object, the upgrade may have added a new field in a number range that is not allowed. In this situation, if at all possible, import the object and then implement the customization on the new object using the change log. If this is not practical, the only alternative is to set the action to "Merge Existing<-New" and import the new base object. This brings in the new field(s). Then, apply the remainder of the Change Log.

7. Complete step 6 for every object in the conflicts list.

8. As an optional step, change the version that is displayed on the Help About screen. Note that this step is optional only for an improvement. If this is a Service Pack or a minor or major release, this step must be performed.

    In the Object Designer, open codeunit 1 and look for the function called **ApplicationVersion**. Simply change the displayed version to the new version to display, reflecting the latest base version tag in all the objects. Change the text between the single quote marks to this text. After updating codeunit 1 and saving it, update the version tag of this object as well, to the same version as put in those quotes.

9. The next step is to compile all objects. Do this to make sure there are no conflicts with existing objects that show up. In the Object Designer, click the **All** button, select all objects, and press F11 to compile them. When done, any objects that did not compile are marked. These can be viewed with View, Marked Only.

10. Now, create the Upgraded Objects file. In the Object Designer, click the **All** button, and then filter the **Version Tag** column. Once this is done, clear all the **Modification Flags**.

11. Select all the filtered objects (use CTRL+A) and export them as a Microsoft Dynamics NAV Object File, a file that has the extension .fob. This is the Upgraded Objects file.

12. Install the Upgraded Objects on the customer site. First, bring this file to the customer site. When the objects are loaded, especially table objects, sometimes tables are automatically converted or re-keyed, which takes a long time when there is a lot of data.

    At the customer site, bring down the server. Make a backup copy of the database (enlist the support of the customer's IT department for this step). Start up the client that is located on the Server Machine, and open the customer's database in Local mode. Now go into the Object Designer and import the Upgraded Objects .fob file.

    Remember, even if there are now conflicts, they have already been resolved. Therefore, when the Import Worksheet is displayed, just click the **Replace All** button, and then do the import.

    Once this is completed, compile all objects, then bring down the local client. Bring the server back up, and the customer can continue working on the newly upgraded system.

Tools for Upgrading the Objects

Microsoft Dynamics NAV provides several tools to help with performing upgrades. Each one may be best for different circumstances.

Import Worksheet

The Import Worksheet can be a useful tool. It is the best tool for rapidly determining object conflicts. It can also be used to add new objects and fields in the base number ranges that there are no permissions for. It is the best tool to use to install the upgraded objects on the customer's system.

Merge Tool

The Merge tool is the best tool for automatically merging changes. The Merge tool is used only to merge the objects that have conflicts; in other words, only load the objects (from both the customized and the new base versions) into the Merge tool to compare and merge. The Merge tool is also useful to track and store the various versions that exist for various customers. For it to do this properly, disregard previous instructions and load all changed objects from both the customized and new base versions.

Editing Text Objects

C/SIDE allows the importing of text objects as well as Microsoft Dynamics NAV objects. This is the best if manually applied an automatic change log to some objects. It is also used when importing the Merge tool results.

Compare Tool

The Compare tool is the best tool to use when generating and printing an Automatic Change Log. It can also export an automatic change log, which is in a good format to use for manual updates. Remember that when an automatic change log is used, it is best to make the change in a text object file and import it as text. Also, the Compare tool helps find the relative size of each modification by reporting what percentage of each object changed.

Using the Change Log

Many times when an improvement is made, a manually generated change log is published. This is the best tool for documenting modifications that will be implemented many times. However, implementing the modification once or twice, it is too much work to create. Remember that for a manual change log, it is best to make the changes directly in C/SIDE.

A Major Release

If this is a major release with extensive customization, since there are more objects, there will be more chance of conflicts. Since the changes are more extensive, there is more of a chance that the customization will have to be re-implemented, rather than using a change log to re-implement the upgrade.

When planning, include sufficient time to test the upgraded system (after recompiling all objects but before the version tags are set), reset the modified flags, and export. Make sure that there are no obvious problems. Use the customer license to do the test, in case there is a problem with it. After the export, test the object upgrade on a copy of the complete customer database (including data).

One other possibility is that there could be a major functionality change. This usually means a change in the way something is done. In this case, the change is so major that the changes cannot be merged, either with the merge tool or manually. This is because there is a whole new way of doing something, not just a change in the details. If this is the case, re-implement the customization from scratch in any object where there is a conflict. In some cases, the customization may need to be changed even in unchanged objects, should it depend on the old way of doing things.

It is also important to re-evaluate the relevance of the customization when moving to a new version. Perhaps Microsoft Dynamics NAV has added a feature that could replace all or some of the customization. Or perhaps the customer's business processes have changed. Before upgrading, review the customization's functionality and discuss with the customer which parts will be upgraded.

Field Name Changes

If major field name changes were made, the problem is that these changes ripple down into every other object that refers to the changed field, at least in any automatic change log generated. This can create huge change logs where in fact there is no actual change to many objects. For example, suppose **Department** was changed to **Division**. Consider the effect on the **Customer Table**. When the field name is changed, it is changed automatically on the card form, on the list form, on the **Customer List** report, and so on. However, an automatic change log is generated, and it shows that the card form changed, the list form changed, the reports changed, and so on. This makes it harder to do upgrades.

The solution is to make the field name changes first. Before extracting the new base objects from the new database, make the customized field name changes to the tables only. For any object being updated, be sure the modified flag is checked, even if just changing the table name. When extracting the new objects, get all objects whose Version Tag or Modified Flag has been changed. Use the Marking facility to do this. First, filter all objects with a Modified Flag on, select all of them, and press CTRL+F1. Then, remove that filter, filter on all objects with the new Version Tag, select all of them, and press CTRL+F1. Now, remove that filter, use View, Marked Only, and then select all filtered objects.

Customer has Design Tools

If the customer has Design Tools, the problem is that the customer probably does not follow the Microsoft Dynamics NAV development rules. These include rules about setting the Version Tags and Modification Flags, internal object documentation, developing on line, and not using a project log, so one cannot tell why they made a change.

The solution has two parts; first, better planning is necessary. Try to teach customers the rules, explaining how it makes upgrading easier. Or, just get them to follow one rule – leave the Modification Flags alone. This rule is easy to follow, since it does not require them to do anything.

When performing an object upgrade, schedule a time to pick up the customer database from the customer site before upgrading. This is because the copy is not up-to-date. Just use a Microsoft Dynamics NAV backup. Make sure when picking up the backup from the client to inform customers that they should not make any changes until the upgrade is installed. Furthermore, reiterate that any changes they make will not be included in the upgrade. Also, consider identifying a customer contact who helped with the development and who can answer any questions about the changes made.

The second part of the solution is that certain steps in the upgrade process will be changed. Be sure to create a new database and restore the application objects only from the Microsoft Dynamics NAV backup obtained from the customer site. After this, go through and give the customer's changed objects (the ones with the Modification flag on) their own additional Version Tag, something different from the version tag. Use new letters, based on the customer name. For example, if Acme Tools is the customer, make their Version tag AT0, and then add an "AT01" at the end of every modified object with an existing version tag. If the customer created a new object, set the Version tag to "AT01."

For step 6, (updating each object on the conflicts list), take into account customer changes. At the least, generate a new change log from the working copy. After compiling all objects, but before exporting, schedule time for the customer contact to come in and check out the upgraded system. It is easier to see if a customer change was lost this way.

Finally, when exporting the objects, be sure to get all objects with either the new base Version Tag, or the customer's new Version Tag. Use the marking technique to do this. The rest of the steps are the same.

More Than One Upgrade to Do

Here is a common situation. The customer has not upgraded for a while, and has skipped a release and two service packs. Now, with the latest release, the customer sees a feature the customer wants, and the customer wants to be upgraded. The problem is that there is no direct upgrade path except from one version to the next version. The solution requires multiple upgrades at once. Simply do the upgrade steps 2 through 10 for each upgrade version in between. Once upgraded to the final version, remember to export all objects with any version tag higher than what the customer had, so the customer gets all changed objects.

However, data upgrades are usually too complex to combine. Therefore, schedule a separate upgrade for each data upgrade needed. This usually corresponds to a release. When combining object upgrades, export the modified objects for each data upgrade. Note that technically, all the data upgrades can be done at once, even if they are not combined. It is highly recommended that thorough testing be performed between data upgrades if the data upgrades at the customer site are to be combined.

Upgrading the Data

Data upgrade is usually necessary every time there is a release.

Data conversion routines (usually codeunits) are included in an Upgrade Toolkit which is generally released about one month after release of the product. Take advantage of this time lag to become familiar with the new product and any oddities about it. When the data conversion routines are available, try not to modify them. It is best if any data conversion routines needed can be written separately, and then run in series after ours. However, occasionally, it is necessary to modify the data conversion routines, because otherwise they will not work with the customizations. This step requires care. Write (or modify) data conversion routines at the same time as creating the upgraded objects. However, these routines must be run on the customer site, in order to upgrade their live data.

Preparation

There are some additional steps to prepare for a data upgrade. First, when making the working copy, make an additional copy to use to create the phase one conversion objects. Phase one objects must be created using the old, un-upgraded database. When the upgrade of the objects is complete, make another copy of the database, this one to be used to create the phase two conversion objects. Phase two objects must be created on a new, upgraded database.

Also, create a complete copy of the customer's database. This is for testing the upgrade process. Get the customer database from the customer site so that the data is complete and current.

The Conversion Process

The conversion process is highly dependent on the particular configuration of the system and the old and target versions involved. The Upgrade Toolkit provides the steps involved for each possibility.

Please note that all of these steps outlined in the Upgrade Toolkit can take a significant amount of time, especially if the customer has substantial data. Gauge the amount of time when doing the test run at the site. Be sure that the customer can be down long enough for the entire database and object upgrade. This work will probably be done only on weekends or holidays.

Conclusion

You have successfully deployed your seminar module and are prepared for the ongoing maintenance phase. Your client has identified some additional requirements, which can be found in Appendix A, Additional Exercises, but the project is now complete.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

10-21