Microsoft Dynamics NAV

Merging Application Object Source Files

Artifact Type
White Paper

July 2014

Contents

Merging Application Object Source Files

When you create or modify application objects, you use the Microsoft Dynamics NAV Development Environment, and you import and export objects as .fob files. But you can also export objects as text files and use other tools, such as the Microsoft Dynamics NAV 2013 R2 Development Shell, to make general changes to the objects. For example, you can use Windows PowerShell cmdlets to merge changes from an upgrade with your Microsoft Dynamics NAV solution.
You can export all application objects to a single text file. Optionally, you can split the large text file into separate text files for each application object. You can also use an external source control system to store the text files, such as Visual Studio Team Foundation Server, but this is not required by the Microsoft Dynamics NAV cmdlets.
The application merge utilities that are available in the Microsoft Dynamics NAV 2013 R2 Development Shell install when you choose the Developer option in Microsoft Dynamics NAV 2013 R2 Cumulative Update 9 Setup, or if you add the development environment to another installation option.

Comparing and Merging Application Objects

Microsoft Dynamics NAV includes Windows PowerShell cmdlets that help you apply changes to Microsoft Dynamics NAV solutions. You can use Microsoft Dynamics NAV cmdlets to modify application object source files in the Microsoft Dynamics NAV 2013 R2 Development Shell, or by importing the Microsoft.Dynamics.NAV.Model.Tools.psd1 module into the Windows PowerShell Integration Script Environment (ISE).
When Microsoft releases a new version of Microsoft Dynamics NAV, you want to upgrade your solution. Similarly, you want to update your solution with cumulative updates and other smaller changes to the application. Each time you want to update your solution, you have to compare the original version to the new version, and then you have to apply the difference to your own solution. Or you compare the new version to your own solution and merge the difference into the new solution. In both cases, you compare different versions of application objects to calculate and apply the difference. The Merge-NAVApplicationObject cmdlet compares the changes that have been made to application objects between two sets of Microsoft Dynamics NAV application objects, and applies the difference to a third set of application objects, the target of the merge. The result of the application object merge is a fourth set of application objects that make up your new, updated solution.
In the Microsoft Dynamics NAV 2013 R2 Development Shell, you can also use other Windows PowerShell cmdlets that help you manage application object files. But the recommended way to update your application is to use the Merge-NAVApplicationObject cmdlet. The other option is a two-step process using the Compare-NAVApplicationObject and Update-NAVApplicationObject, cmdlets.

Object Types supported by the Application Merge Utilities

The following object types can be merged using the new cmdlets:

- Tables
- Pages
- Reports
- Codeunits
- MenuSuites
- Queries
- XMLPorts

The following object types cannot be merged:

- Dataports
- Forms
- Reports with Classic report sections

Scenario: Updating your solution to the next cumulative update

In the following example, you have built a solution that is based on Microsoft Dynamics NAV 2013 R2, MySolution. Microsoft then releases Microsoft Dynamics NAV 2013 R2 Cumulative Update 9 that includes hotfixes and two regulatory features for your country/region. To help you apply the cumulative update to your solution, you use the new Microsoft Dynamics NAV cmdlets. The cmdlets calculate the modifications that Microsoft made between the original release and the cumulative update, and apply these to your solution. Many of the modifications are automatically merged, but a set is left for you to manually resolve. Conflicts are part of the merge process, such as when you and Microsoft have changed the same line of code, or modified the same property on a page. You must also validate the automatically applied modifications through testing and maybe code review. The following table describes the three versions of the Microsoft Dynamics NAV application that you want to compare and merge.

Version	Description
ORIGINAL	The baseline of the application merge. For example, the Microsoft release of Microsoft Dynamics NAV 2013 R2.
MODIFIED	The updated version of the original. For example, this can be Microsoft Dynamics NAV 2013 R2 Cumulative Update 9. Alternatively, it can be a small add-on. In many cases, the modified application is the version that contains fewer changes to the original than the version that is the target of the merge. This is because you want to apply fewer changes to a large application rather than applying a large change to a small application.
TARGET	The version of the application that you want to apply the difference between the original and the modified application to. For example, this can be your solution that you want to apply a cumulative update to. Alternatively, it can be a new major release from Microsoft that you want to apply your modified solution to.

The Microsoft Dynamics NAV cmdlets calculate the modifications made from **original** to **modified**, capture these, and then apply automatically as many as possible to the **target** application with a resulting application named **result**. Some application objects cannot merge without human intervention; these are called **conflicts** and described in separate text files that you can analyze.

Tip:

The name or the parameter **target** is carefully chosen to indicate a direction: You identify external modifications and apply them to your target solution. However, sometimes it may be advantageous to swap the MODIFIED and TARGET parameters, for example for version upgrades. Since the processing is so fast, typically few minutes, it is recommended to try out the two options. In most cases, it's easier if you place the set of application objects with the fewest modifications in MODIFIED.

Tip:

While the application merge utilities work well with cumulative updates as described, they can be used for any set of application objects. For more information, see Using the Windows PowerShell Cmdlets with Other Versions of Microsoft Dynamics NAV.

Application Object Source Files

The input to and output from the application merge utilities are text files that have been exported from a Microsoft Dynamics NAV database. For more information about exporting text files, see How to: Export Objects in the MSDN Library.

When you export your application objects as text files, you can – but do not have to - store the files in external source control systems. That way, you have source control and version control. You can also use other tools to bulk edit the files, such as if you want to set a version number.

If you are using a version control system for your application objects, you can use the text files directly from there. For more information, see Using the Application Merge Utilities with a Version Control System.

If you are merging cumulative updates, Microsoft ships these both as text files and as .fob files and you can use the files from there.

You can export all objects into a single file, or you can export objects into separate files per object. Microsoft Dynamics NAV includes Windows PowerShell cmdlets that can split a large text file into separate files per object, and join separate files into a single text file. Most of the cmdlets support text files with one file of application objects, list of separate files, and folders that contain text files.

Using the Application Merge Utilities with Version Control Systems

Using a version control system such as Visual Studio Team Foundation Server for your application source text files is always highly recommended, and it makes the code upgrade scenarios using the Windows PowerShell cmdlets simpler because you have easy access to all text files that you must pass to the cmdlets. However, this is in no way required for using the AMU, since it is completely text file and folder based.

Using the Windows PowerShell Cmdlets with Other Versions of Microsoft Dynamics NAV

The Windows PowerShell cmdlets that are introduced in Microsoft Dynamics NAV 2013 R2 Cumulative Update 9 are not version-specific. However, they work better with application objects from Microsoft Dynamics NAV 2013 and later because the metadata type changes are fewer compared to objects from Microsoft Dynamics NAV 2009, where forms

became pages, and so on. For those obsoleted types, and the upgrade and transformation of those, see Upgrading to Microsoft Dynamics NAV 2013 R2 in the MSDN Library.
In general, you can use the cmdlets to upgrade tables and codeunits across versions with a high percentage of automatic merge.

Merged Files – with Conflicts

When you merge application changes by running the Merge-NAVApplicationObject cmdlet, the differences are applied automatically if possible. However, when conflicts are detected, such as conflicting sections of code, or conflicting object IDs, those conflicts are captured in CONFLICT files that you must manually resolve. CONFLICT files are plain text files that you can open in text editors such as Notepad. CONFLICT files are helpful because they clearly identify where two parties such as you and Microsoft have changed the same object or parts of it. Conflicts are to some extent avoidable by following a few simple best practices.

Compare and Update Files – with Deltas

As an alternative to the Merge-NAVApplicationObject cmdlet, you can use the Compare-NAVApplicationObject and Update-NAVApplicationObject cmdlets together. This introduces a set of DELTA files that capture the differences between two application objects. DELTA files are plain text files that you can open and explore in text editors such as Notepad. Generally, we recommend that you use the Merge-NAVApplicationObject cmdlet. For more information, see How to: Use the Compare/Update Cmdlets.

Special Handling Required

The application merge utilities handle all object types, but some better than others. Also, some parts of some object types require special attention from the tool or from you as described in the following list:

- Application object properties.
  Each of the application object properties in the application object files often calls for hands-on handling. You can run the cmdlets with the default parameters values, but you can also overrule the default behavior by setting parameters specific for application object properties, or you can run other cmdlets for post-processing. Two cmdlets are available specifically for post-processing: Get-NAVApplicationObjectProperty and Set-NAVApplicationObjectProperty. For more information, see Handling Application Object Properties.
- Documentation triggers
  In each object, the documentation trigger is used for many different purposes. Documentation, of course, but also for tracking changes with date or history semantics built into it. You can use the cmdlets to modify the contents of the documentation section. For more information, see Handling Documentation Triggers.
- ControlID
  Sometimes, developers in-house or external partners create objects in the same ID range as you. This surfaces in the merge process as conflicts and hence as work to do. The application merge utilities provide ways to handle that, depending on your needs and requirements. For more information, see Handling Application Object Properties.
- CaptionML
  Captions are an integral part of the solution, but when you compare two versions of the same object with two different languages, you will see extensive differences. We plan to add support for handling captions as part of the application merge utilities in a later version. However, you can choose as a temporary workaround to export captions to an external file during the application merge. For more information, see How to: Add Translated Strings for Conflicting Text Encoding Formats in the MSDN Library.

Handling Application Object Properties

Application object properties are listed in a separate section at the top of the exported text files. For each exported object, the following properties are listed:

- Modified
- Date
- Time
- VersionList

Each of them is different by nature, and they are not part of the merge process as such. However, the resulting text files must have these set according to the practices that your solution follows. When you run a cmdlet such as Merge-NAVApplicationObject, you can specify if you want to change the default behavior, which is to clears date, time, and version list, and copies the value for Modified from the TARGET application object. But you can also run dedicated cmdlets, Get-NAVApplicationObjectProperty and Set-NAVApplicationObjectProperty, to manage these properties after the object merge.

For detailed description and possible values, type the following command in the Microsoft Dynamics NAV 2013 R2 Development Shell:

```
Get-Help Merge-NAVApplicationObject -detailed
```

Post-processing Application Object Properties

Often handling or setting application object properties is done at a later time than the merge itself. For example, you run the merge, resolve the conflicts, and once you're done, you want to stamp the current date and time on all objects. To do that, use the **Set-NAVApplicationObjectProperty** cmdlet to change the values of the Version List, Date, Time, or Modified properties in the specified text files. You can use the **Get-NAVApplicationObjectProperty** cmdlet to extract information about the application objects before you change them.

The following command sets all object properties on codeunit 1 (COD1.TXT). VersionList is set to *DemoV1*, Modified to Yes, and Date and Time are set to the current date and time.

```
Set-NAVApplicationObjectProperty -Target .\COD1.txt -VersionListProperty "DemoV1" -
ModifiedProperty Yes -DateTimeProperty (Get-Date -Format g)
```

Tip:

The **Get-NAVApplicationObjectProperty** and **Set-NAVApplicationObjectProperty** cmdlets can be used independently of the other cmdlets whenever you want to set or clear application object properties.

For detailed description and possible values, type the following command in the Microsoft Dynamics NAV 2013 R2 Development Shell:

```
Get-Help Set-NAVApplicationObjectProperty -detailed
```

Handling Documentation Triggers

Documentation triggers in text files are essentially free format, so there is no unique way to merge the content. If you want to be explicit about how the content of the documentation triggers is merged, you can set the *-DocumentationConflict* parameter when you run the **Merge-NAVApplicationObject** cmdlet. Set this parameter when you are merging objects with the same type of content in the documentation trigger, such as technical descriptions or a version list. By default, conflicting lines of documentation are merged into the result file with the content from the modified object listed first. This is particularly useful when the objects contain version history in the documentation triggers.

For detailed description and possible values, type the following command in the Microsoft Dynamics NAV 2013 R2 Development Shell:

```
Get-Help Merge-NAVApplicationObject -detailed
```

Handling IDs

To reduce noise, by default the **Merge-NAVApplicationObject** cmdlet suppresses conflicting modifications on non-functional incidents such as the ordering of variables and methods that are sometimes referred to as "control IDs". If you set the*–Strict* parameter, these occurrences are treated like any other conflicts and reported accordingly. Significant IDs, such as table and field IDs, are fully considered during compare and merge.

Output from the Merge and Update Cmdlets

The output from the cmdlets comes in different ways, flavors, and form. The following describes first what is output to the console, exemplified by the **Merge-NAVApplicationObject** cmdlet, followed be the data written to disk as files and folders, and then for scripting powerful purposes collections of Info objects and their usages in a scripting environment like PowerShell.

Output Window

The **Merge-NAVApplicationObject** cmdlet processes the application objects one at a time. For each application object, a summary is output, including one of the following results:

- Merged
- Conflict
- Inserted
- Deleted
- Unchanged
- Failed

The output from the cmdlets has a summary and a details section. It also has a description of the above mentioned result types. The following example illustrates the output from running the Merge-NAVApplicationObject cmdlet:

```
Summary:
  Merge operation processed 14 application object(s) with a total of 5 individual change(s).
  80,0% of individual modifications were automatically merged.

Details:
  Processed 14 application object(s):

    Merged        3 objects - with changes in MODIFIED that were successfully merged with any changes from TARGET
                              into RESULT.
    Conflict      2 objects - with changes in both MODIFIED and TARGET that could only be merged partially.
                              Partially merged objects and corresponding .CONFLICT files are added to RESULT.
                              This also includes objects that are deleted in MODIFIED/TARGET and changed in
                              TARGET/MODIFIED.
    Inserted      0 objects - in MODIFIED that do not exist in TARGET and are inserted into RESULT.
    Deleted       0 objects - that exist in ORIGINAL, but do not exist in MODIFIED and are unchanged in TARGET.
    Unchanged     9 objects - in TARGET which are not changed in MODIFIED and are copied from TARGET to RESULT.
                              This also include objects deleted in both MODIFIED and TARGET and objects unchanged
                              in MODIFIED and deleted in TARGET.
    Failed        0 objects - that could not be imported, such as an object that is not valid or that contains
                              unsupported features.

  Processed 5 individual changes:

    Conflict      1 changes
    Merged    80,0% of all changes

To see detailed explanation of these, type: "get-help Merge-NAVApplicationObject -detailed".
```

Summary

Each application object can have multiple logical modifications. An example of this is codeunit 1. Potentially, that can be tagged as an application object with conflicts, but 99 out of 100 modifications might be applied automatically. The summary shows information about both modification and object level.

Details

The Details section shows the number of application objects for each of the six possible results with a description of the result.

The following table describes how and when the result types are set by the Merge-NAVApplicationObject cmdlet depending on the input from MODIFIED and TARGET.

TARGET / MODIFIED	Customized	Added	Deleted	Untouched
Customized	Merged or Conflict	-	Conflict	Merged
Added	-	Inserted or Conflict	-	Inserted
Deleted	Conflict	-	Unchanged	Deleted
Untouched	Unchanged	Unchanged	Unchanged	Unchanged

The following table describes how and when the result types are set by the Update-NAVApplicationObject cmdlet, depending on the input from DELTA and TARGET input:

TARGET / DELTA	Customized	Added	Deleted	Untouched
Customized	Updated or Conflict	-	Conflict	Updated
Added	-	Inserted or Conflict	-	Inserted
Deleted	Deleted	-	Unchanged	Deleted
Untouched	Unchanged	Unchanged	Unchanged	Unchanged

Files and Folders

The cmdlets generally take an output parameter and in most cases it is named –RESULT. It can either be an existing folder, or a filename. If a folder of the given name does not exist, a single file is created that contains all application objects. If an existing folder is provided, separate text files for each application object are created in that folder using standard C/SIDE naming, such as COD1.TXT for codeunit 1.

In addition, three extra folders are created. There is one for each input folder, ConflictOriginal, ConflictModified, and ConflictTarget. Separate files for each object is created or copied there, for seamless scripting purposes. For an example, see How to: Complete the Merge Process in a Three-Way Merge Tool.

The following snippet illustrates the content of the RESULT folder after the merge has completed:

```
04-07-2014  15:49             7.573 COD1.TXT
04-07-2014  15:49               412 COD1.CONFLICT
04-07-2014  15:49            31.250 COD2.TXT
04-07-2014  15:49             1.097 COD7.TXT
04-07-2014  15:49           120.374 COD8.TXT
04-07-2014  15:49               270 COD8.CONFLICT
04-07-2014  15:49             3.033 PAG6.TXT
04-07-2014  15:49             1.595 PAG8.TXT
04-07-2014  15:49             4.487 TAB3.TXT
04-07-2014  15:49            48.786 TAB14.TXT
04-07-2014  15:49    <DIR>          ConflictModified
04-07-2014  15:49    <DIR>          ConflictOriginal
04-07-2014  15:49    <DIR>          ConflictTarget
```

The text files follow the same structure as any other text files with Microsoft Dynamics NAV application objects. . For more information, see How to: Export Objects in the MSDN Library. Two other files describes differences between two application objects (DELTA files) and conflicts requiring user invention from a merge or update process (CONFLICT files). Both are simple text files that you can open in text editors such as Notepad. For more information, see DELTA and CONFLICT-files.

Info Objects

The output from the most of the cmdlets are also rich objects, such as MergeInfo objects from Merge-NAVApplicationObject cmdlet. You can use this output in Windows PowerShell to do further proccessing on each of the application objects merged. For example, you can filter and get only those objects that have conflicts, and open them in Notepad or any three-way merge-tool. For more information, see How to: Filter and Pipe Merge-output.

To see the properties of the MergeInfo object, you can save the result of the merge operation in a Windows PowerShell variable as illustrated in the following command:

```
$myVariable = Merge-NAVApplicationObject -Original .\ORIGINAL\*.txt -TARGET
.\TARGET\*.txt -Modified .\MODIFIED\*.txt -Result .\RESULT
```

And then either get the properties using Get-Member:

```
$myVariable | Get-Member
```

Or the data and the properties using list or table formatting, such as the following:

```
$myVariable | Format-List
```

An example of the latter with properties to the left:

```
ObjectType  : Codeunit
Id          : 1
MergeResult : Conflict
Original    : X:\ORIGINAL\COD1.TXT
Target      : X:\TARGET\COD1.TXT
Modified    : X:\MODIFIED\COD1.TXT
Result      : X:\RESULT\COD1.TXT
Conflict    : X:\RESULT\COD1.CONFLICT
Error       :
```

ObjectType and Id identifies the application object, and MergeResult is the result state of the operation.

The Conflict property identifies the CONFLICT file that you must investigate. For more information, see How to: Use the Join/Split Cmdlets.

The four .TXT file paths are pointing at separate files for the application objects as input to the cmdlet, plus the resulting, partially merged result. These can be used to invoke or initiate further processing, such as launching third-party three-way merge tools. For more information, see How to: Filter and Pipe Merge Output.

DELTA and CONFLICT-files

Differences from the Compare-NAVApplicationObject cmdlet are output as DELTA files, such as PAG312.DELTA, and conflicts detected running the Merge-NAVApplicationObject are output as CONFLICT files, such as TAB10000.CONFLICT. Both types are plain text files and can be opened any text editor.

CONFLICT files

CONFLICT files follow the layout pattern from TXT files. It identifies the object and element and hints to what the issue or solution is.

The following example illustrates how a code conflict is described in a CONFLICT file. The conflict is listed as CodeModification that identifies the procedure with the conflict so that you can choose how to resolve the conflict. Note that the CONFLICT files are not valid application object files and you cannot import them into the development environment and compile them until you have resolved the conflicts.

```
OBJECT Modification ApplicationManagement(Codeunit 1)
{
  OBJECT-PROPERTIES
  {
    Date=;
    Time=;
    Version List=;
  }
  PROPERTIES
  {
    Target=ApplicationManagement(Codeunit 1);
  }
```

```
  CHANGES
  {
    { CodeModification  ;Target=ApplicationBuild(PROCEDURE 3);
                         ConflictHint=Code Conflict is inlined in source. }
  }
  CODE
  {

    BEGIN
    END.
  }
}
```

Example of conflict inlining from (COD1.TXT):

```
    PROCEDURE ApplicationBuild@3() : Text[80];
    BEGIN
      {>>>>>>>} ORIGINAL
      EXIT('35473-ORIGINAL');
      {=======} MODIFIED
      EXIT('35978');
      {=======} TARGET
      EXIT('35473');
      {<<<<<<<}
    END;
```

A PropertyModification hints using ChangeType, here identified as a deletion:

```
OBJECT Modification AccSchedManagement(Codeunit 8)
{
  OBJECT-PROPERTIES
  {
    Date=;
    Time=;
    Version List=;
  }
  PROPERTIES
  {
    Target=AccSchedManagement(Codeunit 8);
    ChangeType=Deletion;
  }
  CODE
  {

    BEGIN
    END.
  }
}
```

DELTA files

DELTA files also follows the layout pattern from TXT files. A DELTA file identifies the object and element and describes modification or delta found:. The following example illustrates a CodeModification with the code before (OriginalCode) and after (ModifiedCode).

```
OBJECT Modification ApplicationManagement(Codeunit 1)
{
  OBJECT-PROPERTIES
```

```
  {
    Date=;
    Time=;
    Version List=;
  }
  PROPERTIES
  {
    Target=ApplicationManagement(Codeunit 1);
  }
  CHANGES
  {
    { CodeModification  ;Target=ApplicationBuild(PROCEDURE 3);
                         OriginalCode=BEGIN
                                        EXIT('35473-ORIGINAL');
                                      END;

                         ModifiedCode=BEGIN
                                        EXIT('35978');
                                      END;
                                       }
  }
  CODE
  {

    BEGIN
    END.
  }
}
```

How to: Merge Application Changes

Microsoft Dynamics NAV includes Windows PowerShell cmdlets that can help you apply changes to your application by comparing and merging application objects from different versions. For example, you can use the **Merge-NAVApplicationObject** cmdlet to update your solution when Microsoft releases an update.

When you use the Microsoft Dynamics NAV 2013 R2 Development Shell, the Windows PowerShell cmdlets compare two sets of application objects, calculate the difference, and apply as many of changes as possible to a third version. The sections in this topic illustrate how you can use the **Merge-NAVApplicationObject** cmdlet to merge application changes into your solution. The sections are based on a scenario you want to apply changes from an update to your version of Microsoft Dynamics NAV. For more information, see Merging Application Object Source Files.

The scenario is based on the following three versions of the Microsoft Dynamics NAV application:

Version	Description
ORIGINAL	The Microsoft release of Microsoft Dynamics NAV 2013 R2.
MODIFIED	The updated version of the original, such as Microsoft Dynamics NAV 2013 R2 Cumulative Update 9.
TARGET	Your solution based on Microsoft Dynamics NAV 2013 R2, such as **MySolution**

The steps in the following procedures compare the ORIGINAL version to the cumulative update in MODIFIED and apply the relevant changes to your TARGET solution. As a result, you have an application that contains your solution with the updates from the TARGET application. Alternatively, the TARGET solution can be the cumulative update from Microsoft and the MODIFIED solution can be your solution. The actual versions that you use to set each cmdlet parameter depend on your concrete scenario.

First, you prepare the application object files for the ORIGINAL version. You can do that by exporting the application object from the development environment, or by using the development environment command **ExportObjects**. The following procedure illustrates how to export the objects by calling finsql.exe file from a command prompt.

To prepare the application object files

1. In a command prompt, type the following:

```
finsql.exe command=exportobjects, file=all.txt,
servername=<server name>, database=<database name>,
ntauthentication=yes
```

   For example, to export all objects in the original Microsoft-provided version of Microsoft Dynamics NAV to a single text file, type the following:

```
finsql.exe command=exportobjects, file=original_all.txt,
servername=MyServer, database="Demo Database NAV (7-1)",
ntauthentication=yes
```

2. Optionally, split the exported application object file into separate files in an ORIGINAL folder using the **Split-NAVApplicationObjectFile** cmdlet in the Windows PowerShell IDE or in the Microsoft Dynamics NAV Development Shell:

```
Split-NAVApplicationObjectFile -Source original_all.txt -
Destination .\ORIGINAL
```

2. Repeat these steps until you have the three sets of text files.

   The cumulative update is shipped as a TXT file (as well as FOB file) and available from PartnerSource:

   1. Original
   2. Modified
   3. Target

When you have the text files that you need, and you have the folder structure that you need, you can run the Merge-NAVApplicationObject cmdlet to merge the changes between ORIGINAL and MODIFIED.

To run the Merge-NAVApplicationObject cmdlet to merge application objects

1. Open the Microsoft Dynamics NAV Development Shell in Administrator mode.
2. Navigate to the location of your folders by typing a command such as the following:

   ```
   cd c:\upgrade
   ```

   In this example, the UPGRADE folder on the C drive contains the three folders that you created in the previous procedure, and it contains an empty RESULTS folder. You can now run the cmdlet.
3. To run the cmdlet and automatically merge as many objects as possible, type the following command:

   ```
   Merge-NAVApplicationObject -Original .\ORIGINAL -TARGET .\TARGET -
   Modified .\MODIFIED -Result .\RESULT
   ```

   Depending on the number of objects that you are merging and the number of differences found, this can take a few seconds, a few minutes, or longer. When the process completes, the result is shown. For more information, see Output from the Merge and Update Cmdlets. If no conflicts are found, the application merge is complete and you can import the text files from the RESULT folder into the Microsoft Dynamics NAV development environment and compile them. If any conflicts are found, you must resolve these. For more information, see How to: Manage Conflicts.

How to: Filter and Pipe Merge Output

Windows PowerShell strengths shine when you use piping. While not necessary for piping, the following example uses the ability to capture the Merge-NAVApplicationObject cmdlet output in a variable, and then work off of that.

To run the Merge-NAVApplicationObject cmdlet to merge, filter and pipe application objects

1. Use the three folders or file, and an existing, empty RESULT folder:

   ```
   $result = Merge-NAVApplicationObject -Original .\ORIGINAL -TARGET
   .\TARGET -Modified .\MODIFIED -Result .\RESULT
   ```
2. Show the standard console output using the variable, $result

   ```
   $result.Summary
   ```
3. For all object with conflicts (MergeResult equals ‘Conflict’), select only the Original and Target file properties, and output it in the PowerShell grid view:

   ```
   $result | Where-Object MergeResult -eq 'Conflict' |
             Select Original, Target |
             Out-GridView
   ```
4. Same for all properties and show in list format:

   ```
   $result | Where-Object MergeResult -eq 'Conflict' |
             Format-List
   ```

For more examples, type: "Get-Help Merge-NAVApplicationObject -examples" in the Development Shell.

How to: Complete the Merge Process in a Three-Way Merge Tool

Even in application objects with conflicts, the Merge-NAVApplicationObject cmdlet typically succeeds in merging most of the modifications. However, conflicts are often found, and you may find it easier to handle these in a three-way

merge tool, so you can pass the partially merged file and ORIGINAL and TARGET to an external tool. The following example shows how you can pass each of the conflicting application objects to such a tool.

To run the cmdlet to merge, and pass conflicting files to external tool

1. Use the three folders or file, and an existing, empty RESULT folder:

```
$result = Merge-NAVApplicationObject -Original .\ORIGINAL -TARGET
.\TARGET -Modified .\MODIFIED -Result .\RESULT
```

2. For all object with conflicts (MergeResult equals 'Conflict'), then call NOTEPAD.EXE to see the conflict files one at a time:

```
$result | Where-Object MergeResult -eq 'Conflict' |
            foreach { NOTEPAD.EXE $_.Conflict }
```

3. For all object with conflicts, do invoke an external tool like freeware kdiff3.exe:

```
$result | Where-Object MergeResult -eq 'Conflict' |
            foreach { kdiff3.exe $_.Original $_.Target
$_.Modified -o $_.Result }
```

For more examples, type: "Get-Help Merge-NAVApplicationObject -examples" in the Development Shell.

How to: Manage Conflicts

Running the cmdlets is fast once you have the three sets of TXT files, ORIGINAL, MODIFIED, and TARGET. Handling the hopefully few conflicts is where the domain knowledge kicks in. Ideally it is as simple as importing back into C/SIDE, fix compile errors, and use the CONFLICT files as a to-do-list for manual updates required by you.
Even though the cmdlets aim at making everything import, some constructs caused by the merge process makes C/SIDE reject it. Having application objects in separate files often makes it easier to identify where the real problem is. Often it is as simple as a mismatching BEGIN-END-pair. See How to use Join/Split cmdlets.
You can start by opening each of the CONFLICT files in the RESULT folder, and read and understand what action is required. Often it is inlined in the objects and you can import the object into the development environment and make the relevant changes there. The following procedure shows how you can manage conflicts in the development environment.

To manage conflicts in the Microsoft Dynamics NAV Development Environment

1. Open the Microsoft Dynamics NAV Development Environment.
2. Import the text files from the RESULTS folder.
3. Compile the imported objects. Since conflict-inlining starts by {>>>>>>>}, a compile error is issued and your attention brought to the code needing your eyes.

   The following code example illustrates an object with a conflict.

```
PROCEDURE ApplicationBuild@3() : Text[80];
BEGIN
  {>>>>>>>} ORIGINAL
  EXIT('35473-ORIGINAL');
  {=======} MODIFIED
  EXIT('35978');
  {=======} TARGET
  EXIT('35473');
```

```
        {<<<<<<<}
    END;
```

4. When you have resolved all conflicts, all objects will compile correctly.

Manage Conflicts in Three-Way Merge Tools

Sometimes it is easier to see the application object visually next to each other in a three-way merge-tool, such as Araxis or BeyondCompare. For this, you can leverage the rich output from the cmdlets (see Info Objects) and pass it seamlessly to your favorite external tool, see How to: Complete the Merge Process in a Three-Way Merge Tool.

How to: Complete the Application Merge

Resolving conflicts is not completely the end of the exercise. Even though the cmdlets are based on the Microsoft Dynamics NAV metamodel, we recommend that you do a final validation of the automatically merged objects. The following list outlines the recommended process for completing an automated application merge:

- Code review – consider doing a review of the merges does. Even code that compiles may be wrong.
- Compile – the simplest of all checks
- Test – build a library of tests and run them after each merge session, preferable automated
- Save – if you use source code control, set appropriate metadata and check in the newest version. For more information, see How to use Get/Set-NAVApplicationObjectProperty.

How to: Use the Compare/Update Cmdlets

We recommend that you use the Merge-NAVApplicationObject cmdlet because it is the faster and easier way to merge objects. However, you can choose to use the Compare-NAVApplicationObject cmdlet and Update-NAVApplicationObject cmdlet to identify the differences before you apply them. Or, for example, when you want to apply the same modifications to multiple applications or similar.
The following procedure illustrates how you can use these two cmdlets in combination.

To run the Compare-NAVApplicationObject and Update-NAVApplicationObject cmdlets

1. Open the Microsoft Dynamics NAV Development Shell in Administrator mode.
2. Navigate to the location of your folders by typing a command such as the following:

   ```
   cd c:\upgrade
   ```

   In this example, the UPGRADE folder on the C drive contains the three folders that you created in the How to: Merge Application Changes procedure, and it contains two empty folders: DELTA and RESULTS.
3. To identify the delta between ORIGINAL and MODIFIED, type the following command:

   ```
   Compare-NAVApplicationObject -Original .\ORIGINAL\*.txt -Modified
   .\MODIFIED\*.txt -Delta .\DELTA -Force
   ```

   This generates a number of DELTA files that describe the difference between ORIGINAL and MODIFIED. You can apply those differences to TARGET by using the Update-NAVApplicationObject cmdlet. The following example illustrates how you can pipe the DELTA files to the Update-NAVApplicationObject cmdlet.

   ```
   Compare-NAVApplicationObject -Original .\ORIGINAL\*.txt -Modified
   .\MODIFIED\*.txt -Delta .\DELTA -Force -PassThru |
       Update-NAVApplicationObject -Target .\TARGET\*.txt -Result
   .\RESULT -Force
   ```

How to: Use the Join/Split Cmdlets

You can use the Join-NAVApplicationObjectFile cmdlet to combine multiple text files with application objects into a single text file, and you can use the Split-NAVApplicationObjectFile cmdlet to generate individual text files for each application object in a combined file.

To combine and split application object files

1. Open the Microsoft Dynamics NAV Development Shell in Administrator mode.
2. Navigate to the location of your folders by typing a command such as the following:

   ```
   cd c:\upgrade
   ```

   In this example, the UPGRADE folder on the C drive contains the three folders that you created in the How to: Merge Application Changes procedure. The ORIGINAL folder contains a large number of text files, and the MODIFIED folder is empty. At the root of the UPGRADE folder is a single text file with multiple objects, updates.txt.
3. To split the large text file in the MODIFIED folder into individual text files, type the following command:

   ```
   Split-NAVApplicationObjectFile –Source updates.txt –destination
   .\MODIFIED\*.txt -Force
   ```

   This generates number of text files, one for each object in the source file.
4. To combine the individual text files for codeunits in the ORIGINAL folder into a single text file , type the following command:

   ```
   Join-NAVApplicationObjectFile –Source .\ORIGINAL\COD*.txt –
   destination .\ORIGINAL\all_codeunits.txt -Force
   ```

   This generates a new text file with the contents of all files where the filename starts with *COD*.

How to Use Get/Set-NAVApplicationObjectProperty

You can use the Get-NAVApplicationObjectProperty cmdlet to print a list of specific object properties on a set of application objects, , and you can use the Set-NAVApplicationObjectProperty cmdlet to bulk-edit a number of application objects to set a specific version, for example.

To get and set properties on application object files

1. Open the Microsoft Dynamics NAV Development Shell in Administrator mode.
2. Navigate to the location of your folders by typing a command such as the following:

   ```
   cd c:\upgrade
   ```

   In this example, the UPGRADE folder on the C drive contains the result of the How to: Merge Application Changes procedure.
3. To get the version list for the codeunit objects in the TARGET folder, type the following command:

   ```
   Get-NAVApplicationObjectProperty -Source .\TARGET\COD*.txt | select
   VersionList
   ```

   The cmdlet prints a list that shows the values of the VersionList properties for the codeunits in this folder.
4. To set the version list for the codeunit objects in the RESULT folder, type the following command:

   ```
   Set-NAVApplicationObjectProperty -Target .\RESULT\COD*.txt -
   VersionListProperty "DemoV1" -ModifiedProperty Yes
   ```

The cmdlet updates the relevant text files with codeunits to set a new value for the VersionList properties and to set the Modified property to Yes.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship, and supply chain processes in a way that helps you drive business success.

United States and Canada toll free: (888) 477-7989 Worldwide: (1) (701) 281-6500 www.microsoft.com/dynamics