



Versioning and Cloning Processes In IBM DB2 UDB Warehouse Manager

April 25, 2003

Andrew Perkins
IBM Americas Advanced Technical Support
Data Management - Business Intelligence
aperkin@us.ibm.com



TABLE OF CONTENTS

1	Introduction	1
1.1	Have you ever wished?	1
2	Cloning 101	2
3	Simple Cloning – Implementing Rudimentary Versioning	3
3.1	Implementing Versioning	3
3.2	Cloning new version	4
3.2.1	Exporting the process template	5
3.2.2	Editing the tag file	6
3.2.3	Importing the cloned process	8
3.3	Re-cloning or maintaining clones	8
3.3.1	Making version-specific modifications	9
3.3.2	Modifying a process template	9
3.4	Summary	10
4	Mass Cloning	11
4.1	The Challenge	11
4.2	The Approach	12
4.3	Developing the Master Template Processes	14
4.3.1	Testing the Master Templates	16
4.4	Cloning	17
4.4.1	Manual cloning	17
4.4.2	Automating the cloning process	18
4.5	Summary	29
5	Moving processes from the development environment to the test environment to the production environment	30
5.1	The strategy	30
5.2	A small twist on this strategy	31
6	Summary	32

1 Introduction

1.1 Have you ever wished?

DB2 UDB Warehouse Manager (WHM) has a very nice GUI development environment that is called the Data Warehouse Center (DWC). It provides a nice process modeler to develop ETL workflows. While developing your processes, have you:

1. Wished that you could create a new 'version' of a process to apply the next round of modifications?
2. Ever wanted to create a new process similar to one that already exists and wished that you could copy the current process as a starting point?
3. Have you ever had a large number (10 to hundreds) of processes that all looked the same with variations in file names, step names, UDP parameters, etc?

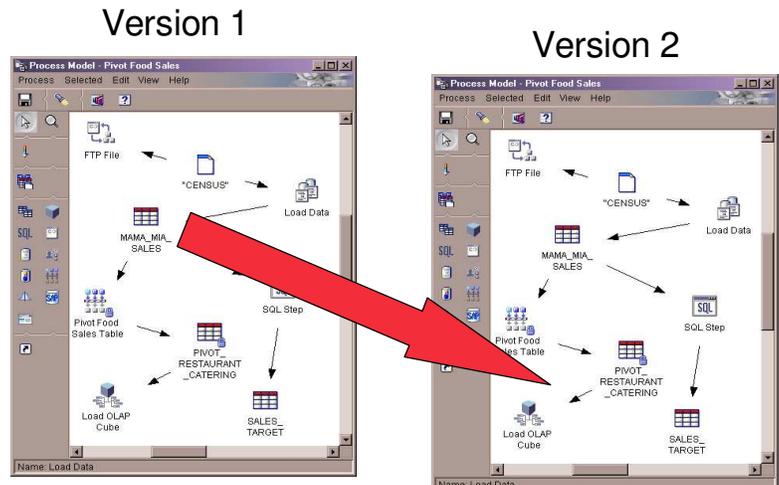


Figure 1-1 Create a new version

When developing processes in Warehouse Manager, we sometimes find that we would like to create a new process that is very similar to one that already exists. The reasons for this may be varied. We may need to make a modification to a current production process but still need to keep the old process intact. We may have several processes that have the same structure with the only difference being the data objects. Or, we may have an extreme need to create hundreds of similar processes from some type of template.

The aim of this paper is to document the process cloning techniques that were developed for use in deploying hundreds of processes to meet the requirements of a particular global customer. We will see how to use a simple cloning technique to provide rudimentary process versioning as well how to develop a template for more extensive cloning. We will also explore some of the design issues that have to be considered when developing processes that need to be cloned.

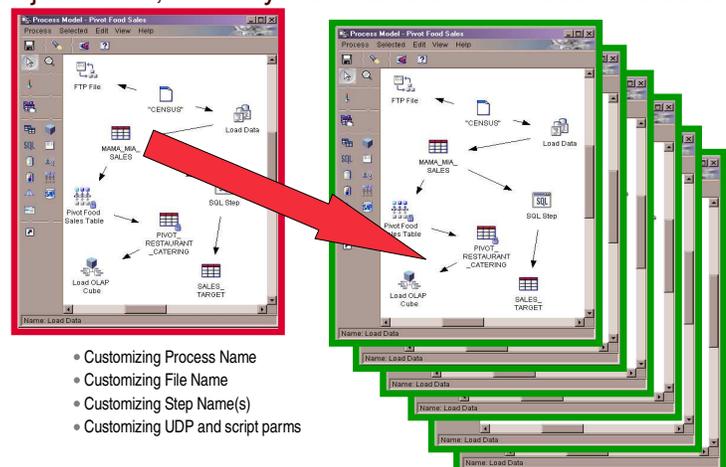


Figure 1-2 Mass Cloning

Of course, we must have a disclaimer that states that these techniques may not work in all instances and should be used only as a guide to developing your own techniques.



Also, keep in mind that these techniques reflect the capabilities of the DWC at the DB2 V7.2 level. DB2 V8 brings many new features to the DWC that would positively impact these techniques, for example, the capability to export a process without exporting dependent processes.

2 Cloning 101

While the DWC is very nice graphical environment for developing our process models, it does not lend itself to the batch orientation of cloning a process. We can only copy steps to a new process but can't copy a set of steps, or a process, and include the flow control information. All object names, such as Process and Step names, have to be unique across the Control Database (CDB). So, this adds up to a lot of labor to copy all the steps of a process and add back in the control flows. Therefore, we would like to find a better way to help us clone processes.

It is important in developing our cloning techniques that we do not do anything that would cause WHM to become unsupported, like modifying the contents of the Control Database (CDB). The key to our cloning is using the external tag representation of our process that is obtained using the Export to tag file capability of DWC. We will limit any modifications we make to this tag file. Even then, we have to be careful not to modify objects inadvertently.

Basically, we will create a process that has replaceable tokens, export the process to a tag file, edit the tag file to replace the tokens with actual values and import the modified tag file back into the DWC. This is somewhat similar to what is done with the WHM ISP Toolkit which uses predefined templates with replaceable tokens with which another vendor can use to build DWC processes from within their own tool. Evolutionary Technologies, for example, uses this to create DWC process from within their tool, ETI*Extract. The primary difference is that we are actually using DWC to create our templates.

When we export a process to a tag file, we create a primary file with the extension of tag, filename.tag, and there may also be a number of other files with a numeric extension, i.e. filename.1, filename.2, etc. These extra files, called blobs, are created to hold information that cannot be embedded in the tag file. For example, if we use a SQL step and we use the SQL Wizard to build our SQL, a blob file is created to hold the metadata from the SQL Wizard but the entire composed SQL statement is also contained in the tag file. These blob files are very sensitive to changes in length of strings and should not be modified during the cloning process.

As we design processes that we want to be cloned, we have to make certain tradeoffs to ensure that our resulting process can easily be cloned, especially when we have to do mass cloning to create hundreds of processes. In this size of effort, our goal is to avoid having to manually edit any of the cloned processes.



3 Simple Cloning – Implementing Rudimentary Versioning

In a lot of instances, all we may want to do is to copy an existing process to create a new ‘version’ of the process. This could be to make changes to a current process while retaining the current process. Or perhaps, we know that we have a handful of processes that all look pretty much the same and we just want to copy a process to give us a starting point to develop the new one. In this section, we will take a look at a simple cloning technique that will allow us to implement this type of simple cloning in the context of implementing rudimentary process versioning.

3.1 Implementing Versioning

To create a new ‘version’ of a process, we want to be able to copy all of the steps in the process to a new process with a new name. We are not looking to change the contents of the process, but just make a new copy of the process. One of the first things that we have to consider is that every object in a CDB must have a unique name. Therefore we have to have a new name for each ‘version’ of a process as well as each step within that process.

To accomplish this, we need something in each process and step name to replace during the cloning process to make it unique. We will use a token which is simply a string enclosed within square brackets, i.e. [token_name]. We need to determine what token(s) we need to be able to clone these processes. In this case, we will simply use one token: [version]. Cloning works at a process level. In other words, we clone a process, not a step. We can also provide tokens to the Subject Area name so that our cloned processes will be organized separately from our source processes.

Now that we have made some decisions regarding how we want our ‘versioning’ to look, we can start to create a template out of the current processes. To do that, we have to insert our [version] token into the name of every process and step as they are created. These tokenized processes become our base or template process from which we clone and create version instances of these template processes.

Figure 3-1 shows a set of template processes. Notice that in the Subject Area name, the Process names and the step name we have appended to the end of each a string V[version]. The part of the string enclosed in square brackets is the token that we will use for substituting the actual version number. For example, after we clone a version 1 process, this string will become V1.

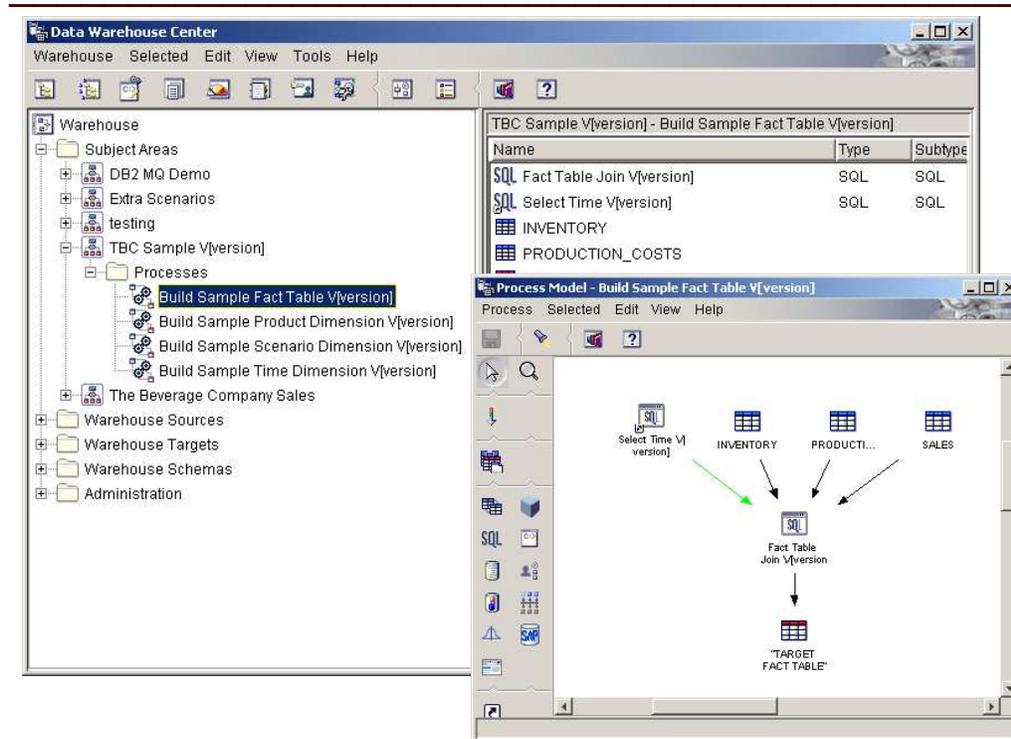


Figure 3-1 Process templates of the TBC Sample processes

We can also use these templates as a set of Master Templates of our processes to which we make only 'official' changes. Once the Master Template has been created or modified and approved, then we can clone our version instances. We can then make version specific changes to the version instance before testing and moving to the production environment.

3.2 Cloning new version

Now that we have process templates created, we can clone these processes to create version instances. To do this, we need to export a process, edit the tag file to change the [version] token to the appropriate value and then import the new tag file. This procedure will result in a new process with new names for the process and all of the steps. The contents of the step definitions are still the same.

We will also have to be concerned with DWC created target tables because if we have two processes that have the same DWC created table as a target table, then one will fail when we do a promote to test mode since DWC will attempt to create a table that already exists. To avoid this, we can simply add the [version] token to the DWC created table name.

As stated earlier, we want to manage our clones at the process level. That means that we need to be able to independently export a single process. We will digress to discuss the behavior of the export utility.

The V7.2 export utility will look for certain kinds of dependencies between processes and export those processes in addition to the requested process. For example, Step S1 in process P1 populates Table T1 and T1 is used as a source for Step S2 in Process P2:

T0→S1→T1→S2→T2. In this case, if we select only P2 to be exported, the V7.2 export utility will also export P1. Also, if we use shortcuts, all process linked via incoming shortcuts are exported as a group. Therefore, we may have to design around this behavior.

However, in V8 the export utility has parameters to modify this behavior and will allow us to independently export a process even if we have these dependencies. There is one caveat when we export a process that has a shortcut. We can choose to export only the one process but include the shortcut definitions. When we import this process the step associated with the shortcut must already exist. In this case, we must ensure that we clone the processes in the correct order or, especially the first instance of a version, clone all processes at the same time.

There is also a special build of the new export utility for V7.2 FP7, command line only, which can be requested via IBM support channels. We highly recommend using the new version of the export utility in any cloning scenario.

3.2.1 Exporting the process template

The first thing that we have to do when cloning any process template is to export the process template to the tag file, also known as the Interchange File. Create a special subdirectory to keep these process template tag files. For our TBC Sample, we will create a subdirectory called TBC_Sample_Master and under this we create a subdirectory for each process.



Figure 3-2 Directory structure for process template tag files

Once we export a process template to the tag file, we don't have to export it again unless the process template changes. We can then clone it multiples times without exporting it over and over. It also provides a good backup of our process templates.

Since we make use of shortcuts between all of our processes, we have to be careful of the order in which we clone our processes. Normally we want to clone one process at a time. However, this is the first time we are cloning for V1. We can either figure out the correct sequence to clone the individual processes or we can clone all of the related processes at once.

Ok, we are now ready to export our TBC Sample process templates in order to create our first version, V1.

We decide for this first time to clone all processes at once, so we select all four of our processes during the export process. We can create a special subdirectory, TBC_Sample_Master\all_processes, to hold the tag file and its associated files.

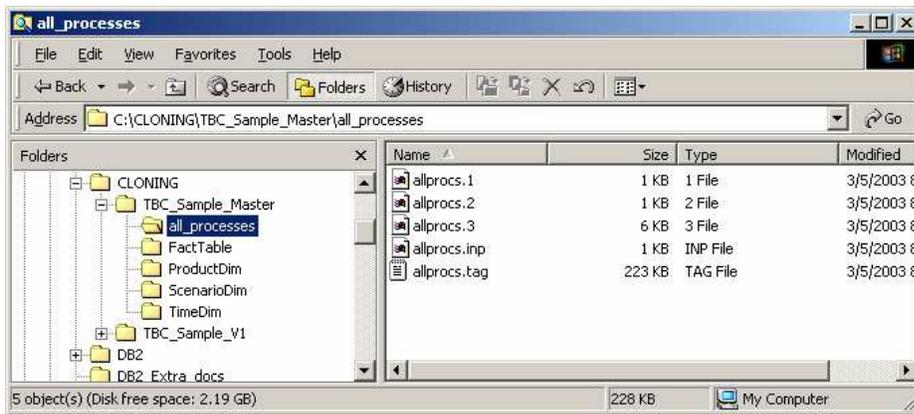


Figure 3-3 Subdirectory for export of all four processes

We see that we have the allprocs.tag file and three blob files and the allprocs.inp file in the all_processes folder. The all_processes.inp file tells the export utility which processes to export during this execution.

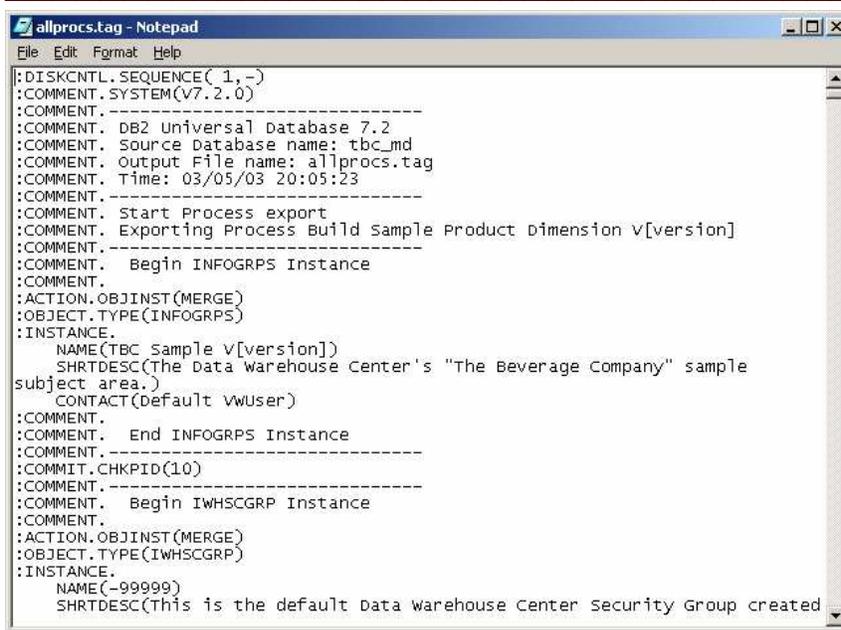
3.2.2 Editing the tag file

The actual cloning takes place at this point. We first want to create a directory structure for our TBC Sample V1 processes that looks just like the subdirectory structure for our TBC_Sample_Master directory. We will copy the process template tag file to this new structure before editing. Make sure to copy all of the files. In our case we copy all five files from TBC_Sample_Master\all_processes to the TBC_Sample_V1\all_processes subdirectory.



Figure 3-4 Directory structure for TBC Sample V1 processes

We edit the allprocs.tag file under the V1 subdirectory structure. We can use notepad or any other text editor to do a find and replace changing the token, [version] to the value 1.



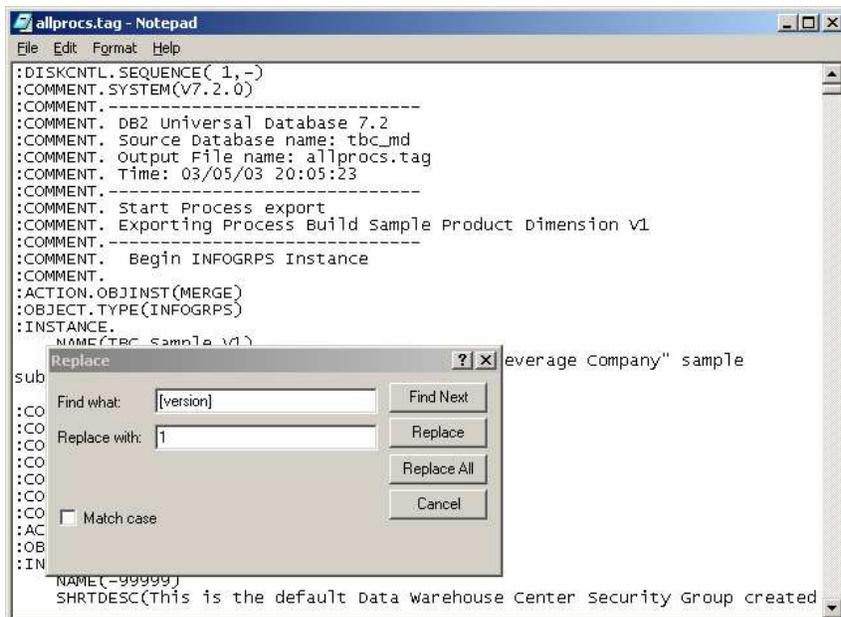
```

allprocs.tag - Notepad
File Edit Format Help
:DISKCNTRL.SEQUENCE( 1,-)
:COMMENT.SYSTEM(V7.2.0)
:COMMENT.-----
:COMMENT. DB2 Universal Database 7.2
:COMMENT. Source Database name: tbc_md
:COMMENT. Output File name: allprocs.tag
:COMMENT. Time: 03/05/03 20:05:23
:COMMENT.-----
:COMMENT. Start Process export
:COMMENT. Exporting Process Build Sample Product Dimension V[version]
:COMMENT.-----
:COMMENT. Begin INFOGRPS Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(INFOGRPS)
:INSTANCE.
  NAME(TBC sample V[version])
  SHRTDESC(The Data warehouse Center's "The Beverage Company" sample
subject area.)
  CONTACT(Default vwuser)
:COMMENT.
:COMMENT. End INFOGRPS Instance
:COMMENT.-----
:COMMENT.CHKPID(10)
:COMMENT.-----
:COMMENT. Begin IWHSCGRP Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(IWHSCGRP)
:INSTANCE.
  NAME(-99999)
  SHRTDESC(This is the default Data Warehouse Center Security Group created

```

Figure 3-5 Tag file before changing token, [version]

In Figure 3-5, we can see a couple of tokens embedded in a string beginning as V[version]. One has the process name in a comment and one is in the Subject Area (INFOGRPS) definition. In Figure 3-6, we can see that both of these tokens have been changed to 1 resulting in the string V1.



```

allprocs.tag - Notepad
File Edit Format Help
:DISKCNTRL.SEQUENCE( 1,-)
:COMMENT.SYSTEM(V7.2.0)
:COMMENT.-----
:COMMENT. DB2 Universal Database 7.2
:COMMENT. Source Database name: tbc_md
:COMMENT. Output File name: allprocs.tag
:COMMENT. Time: 03/05/03 20:05:23
:COMMENT.-----
:COMMENT. Start Process export
:COMMENT. Exporting Process Build Sample Product Dimension V1
:COMMENT.-----
:COMMENT. Begin INFOGRPS Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(INFOGRPS)
:INSTANCE.
  NAME(TBC sample V1)
  SHRTDESC(The Data warehouse Center's "The Beverage Company" sample
subject area.)
  CONTACT(Default vwuser)
:COMMENT.
:COMMENT. End INFOGRPS Instance
:COMMENT.-----
:COMMENT.CHKPID(10)
:COMMENT.-----
:COMMENT. Begin IWHSCGRP Instance
:COMMENT.
:ACTION.OBJINST(MERGE)
:OBJECT.TYPE(IWHSCGRP)
:INSTANCE.
  NAME(-99999)
  SHRTDESC(This is the default Data Warehouse Center Security Group created

```

Replace dialog box:

Find what: [version] Find Next

Replace with: 1 Replace

Replace All Cancel

Match case

Figure 3-6 Replacing the token, [version], with a value, 1

3.2.3 Importing the cloned process

With the editing of the tag file, we have essentially cloned our four processes. All that is left is to import the tag file into the DWC. After the import, as we can see in Figure 3-7, we have a new Subject Area, TBC Sample V1. We also have our four processes with all process and step names reflecting the change of the [version] token to 1 resulting in all names ending with the string V1.

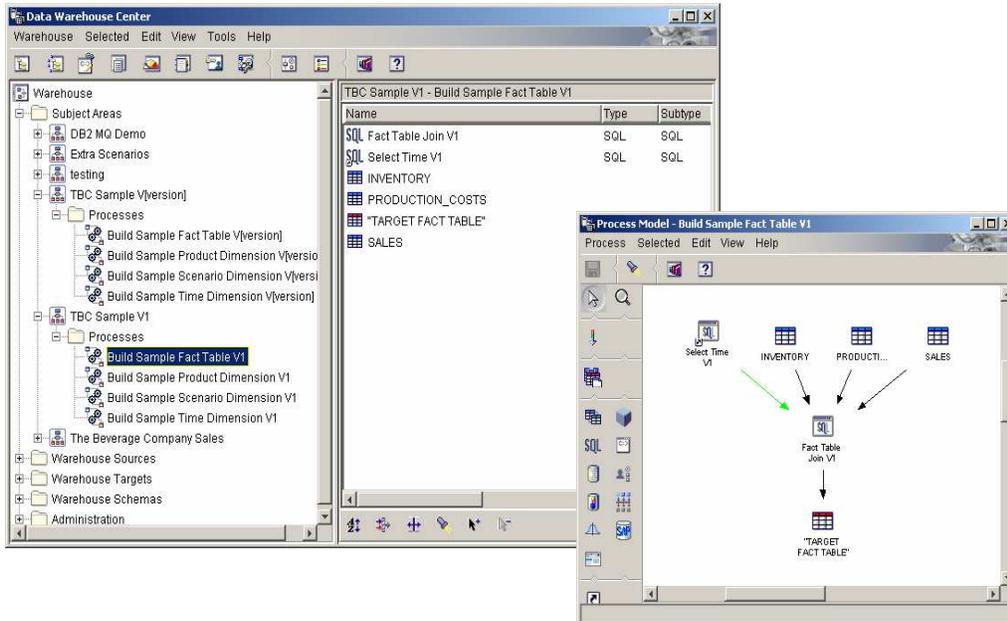


Figure 3-7 The resulting cloned processes

Now that we have newly cloned processes, we are ready to proceed with whatever we need to do. We may just want to put the V1 processes into the production environment. Perhaps we need to make some version specific change to the processes before testing and moving into the production environment. Or, perhaps we can now use this cloned process to create an entirely new, but similar, process.

3.3 Re-cloning or maintaining clones

Now that we have a cloned process, what happens when we need to make a change? We will take a look at a couple of possible maintenance scenarios and discuss how we might address them with our versioning technique.

The steps involved in cloning a single process is the same as for cloning all processes, except that we will deal with only one process. The files will go into the subdirectory that we created for each individual process. We should also ensure that we use the new export utility.

3.3.1 Making version-specific modifications

Assuming that our process templates are the desired, official design of a process, there may be times when, for some reason, the standard process may not work for a particular version instance. Maybe the standard process is dependent on some hardware feature that is not currently available but is expected to be a temporary condition or perhaps there is a product bug introduced that causes temporary design changes. We may decide that instead of modifying the standard process template, we make a modification to the version instance process directly since we expect that by the next version, the situation will be remedied. We would then modify the cloned process, test it and move it into the production environment which will leave our proper design reflected in the standard process template. Future fixes would have to be applied directly to the V1 process.

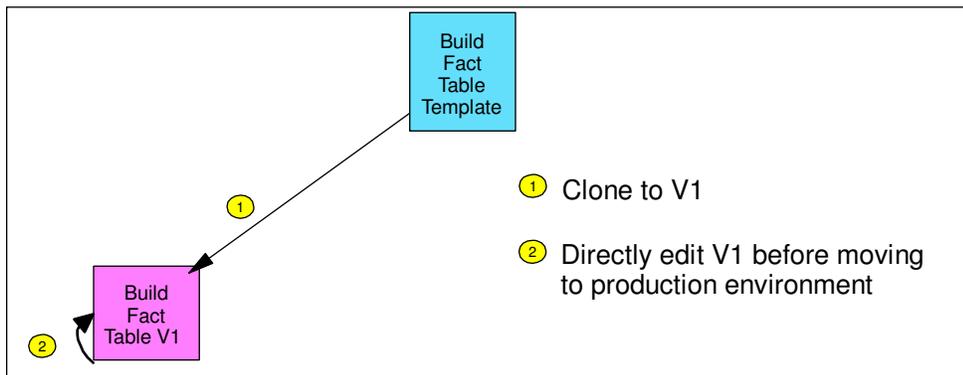


Figure 3-8 Making version-specific modifications

For example, we might have a need to add an extra step to our Build Sample Fact Table process due to work around a database design that could not be implemented in time but we expect that the database design change will have occurred for our next version. After cloning the Build Sample Fact Table V1, then we could edit it directly and make the needed change. This will leave our correctly designed template unchanged.

We may also need to make emergency fixes to a process. This type of change should be made first to the version instance process. Then, if appropriate, the same fix may be applied to the standard process template for future versions.

3.3.2 Modifying a process template

We may have situations where the one of standard process templates is modified but we want to reflect those changes without changing the version number. For example, we need to make a minor modification to process that builds the time dimension, Build Sample Time Dimension. This process is currently at the V1 level but we don't really want to create a V2 as that connotes a major change. We really want some designation that identifies this as a minor change.

We can simple use something like 1.1 for the version or we could designate a minor 'release' of a process the process by adding another token, [release]. In this situation, we would then have names similar to "Build Sample Fact Table v[version]r[release]". We could make a change to

the standard process template and clone that process with the same version but with a new release. We would then have multiple releases of a process within the same version.

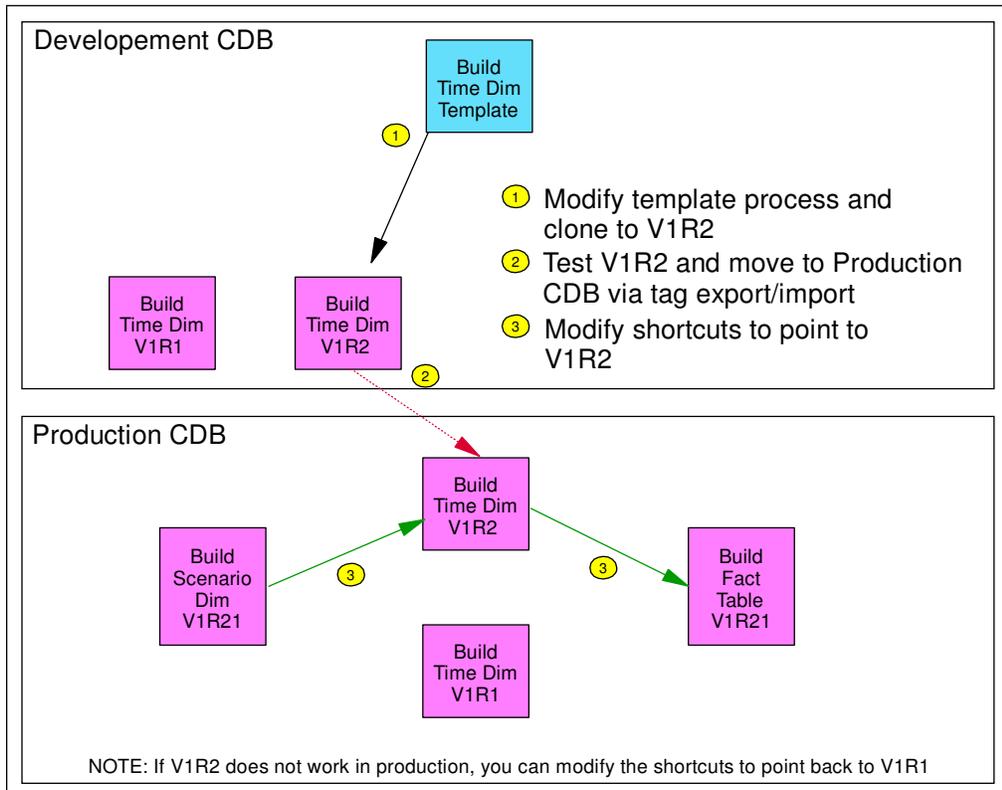


Figure 3-9 Modify a process template and re-clone

In Figure 3-9, we have created a V2R2 of the Build Time Dimension process which we test and move into the production environment. We would then have all processes at the V1R1 level with the exception of the Build Time Dimension process which would be at the V1R2 level. We may have to manually update shortcuts to/from the Build Time Dimension process to reflect the new version and release of this Build Time Dimension process. This also gives a bit of a fallback in case there is some problem with the new process as we can adjust the shortcuts back to point back to the V1R1 process. Once we are convinced the new process is working, then we simply delete the V1R1 process from the production control database, if desired.

Using this technique has the added benefit that we can see a history of changes made to a process if we keep in the development control database all of the old processes or export them to tag files and archive them. We have to remember, however, that importing/exporting tag files may not work across versions of the DWC.

3.4 Summary

In Section 3, we have learned some basic, manual cloning techniques in the context of providing some rudimentary process versioning capabilities. Again, this is not the only way that cloning can be implemented, but rather this is a starting point for developing a versioning/cloning strategy for you.



4 Mass Cloning

In Section 3, we saw an example of how a simple cloning technique can be used and how to implement rudimentary process versioning. In Section 4, we will be going to the opposite end of the cloning spectrum showing how cloning techniques were used by a large, global customer to clone hundreds of processes with tens of thousands of steps from a set of six templates. The process templates were designed such that there were no manual changes needed to the cloned processes. The cloning process was also automated, including the promotion of all of the cloned steps to production mode.

4.1 The Challenge

The objective of this customer's process was to take 160+ flat files from their Mainframe MVS system, get them down to the warehouse Regatta machine and loaded into their DB2 warehouse. There was actually no extraction and no transformation that needed to be done, so this was pretty much a data movement and load scenario. Even without having to do extraction and transformation, this was a very complicated set of processes due to a number of issues:

- a source file is really a set of files including one or more zipped data files, an audit file and an indicator file
- the distributed system had to remotely detect when any one of the 160+ files is ready to be loaded
- the source files are pushed from various regions around the globe on their own schedule and could arrive at any time day or night and must be loaded as soon as possible after they arrive
- the warehouse has a high availability requirement, 24 x 7, and no batch window
- there are complicated inter and intra-process coordination requirements
- there are requirements that the process must be able to detect when a table is offline for backup purposes and to pause and wait. It also has to detect whether an incremental or a full backup is in process and adjust the wait time accordingly
- interface with their host-based Infoman problem reporting system via an AIX executable
- interesting use of the External Trigger to start each process and also to link processes at execution time instead of using shortcuts

But, none of these will be discussed in this paper. We will concentrate on how we will develop and deploy the hundreds of processes and tens of thousands of steps.

This customer had defined some standard approaches to accomplishing this work. They analyzed each target table and, based on its characteristics in terms of size, uptime requirements, incremental vs. full replace, etc they determined that each incoming file could be loaded into the warehouse using one of five standard load strategies. For example, one strategy was to do a bulk load into a staging table and then do an insert into...select from type of process to append to the production table. This would allow the production table to remain online. Another load strategy would load replace directly to the production table which does not have such a strict 24x7 requirement.



They also noticed that the 'ftp' part of the process was the same for every file (set of files). Actually, they later added another ftp strategy to handle a load strategy that utilized a different set of files. So, any file could be downloaded and loaded into the warehouse using a combination of 1 of 2 ftp strategies and 1 of 5 load strategies.

Each file required two processes, one ftp and one load process. Given 164 files, that is 328 processes. The ftp process contained about 25 steps and a typical load was around 40 steps for a total of up to 65 steps per file. That gives us $164 \times 65 = 10,660$ steps.

And, once these 328 process and 10,660 steps are developed, how do we actually get these moved from the development environment to the test (QA) environment to the production environment? Then get 10,660 steps promoted from development mode to production mode?

And, this was just for phase 1... Phase two could quadruple the number of processes and steps.

It is quite a daunting task to develop this many processes manually.

4.2 The Approach

Of course, we developed cloning techniques. We incorporated versioning, ability to copy a process to create a new similar process as well as mass cloning. However, the basic process is the same as the simple cloning already discussed. We developed template processes, albeit more complicated templates with many more tokens, exported these templates to tag files, did global find and replace on the tokens and finally imported the new tag files into the DWC.

We absolutely had to automate the cloning process as well as the promotion of the cloned processes from development mode to production mode. And, of course, we ran into many issues that we had to solve along the way.

In our simple cloning example, we had process templates and version instances. In this scenario, we took this one step further. We had **Master Template Processes**, **Master Version Template Processes** and **File Instance Processes**. A Master Template Process was a fully tokenized process. We had [version] and [release] tags as in the above example, but we had many more tags in the actual step definitions. The Master Template Process is our official standard process and is where official, approved changes are made. The Master Version Processes are cloned from the Master Template Processes and are in themselves templates, with only the [version] and [release] tokens being changed. Version/Release specific changes can be made here. File Instance Processes are cloned from Master Version Template Processes for a specific instance of a source file. All tokens are changed at this time as appropriate for the source file.

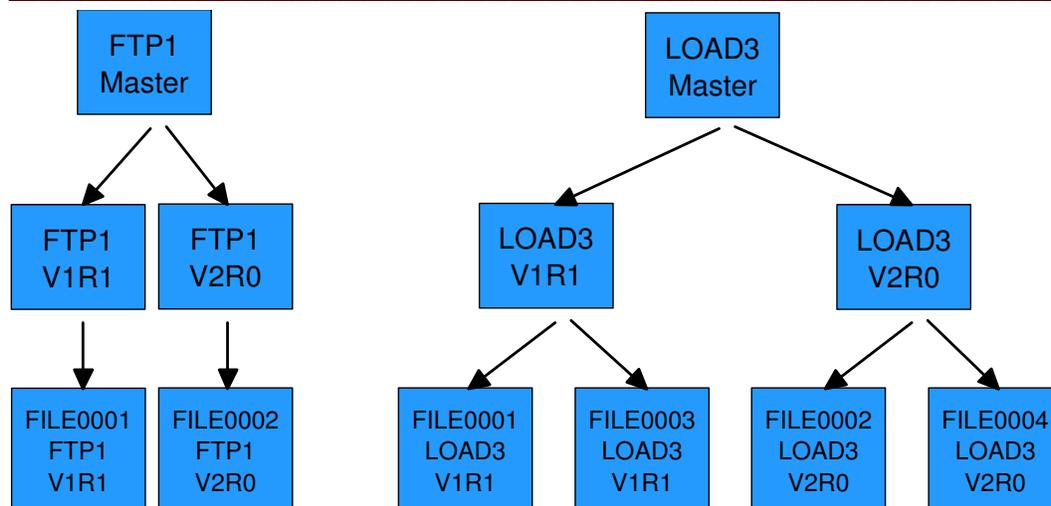


Figure 4-1 Relationship between Master Templates, Master Version Templates and File Instance Processes

As discussed earlier, this customer determined that there are two possible ftp strategies and five possible load strategies. A particular source file would need one ftp process and one load process to accomplish the objective of being loaded into the data warehouse. These actually translate very nicely into Master Template Processes. We have a total of seven Master Template Processes, FTP1, FTP2, LOAD1, LOAD2, LOAD3, LOAD4 and LOAD5. In Figure 4-1 we have the FTP1 Master Template Process and the LOAD3 Master Template Process at the top of the chart. From these we can clone Master Version Templates and in the middle layer we see a FTP1 V1R1 Master Version and a FTP1 V2R0 Master Version and also for LOAD3. Then to get a process we can actually execute, we clone the Master Version Processes to get File Instance Processes. In this example, every source file has a unique fileid which we include as a token in the process and step names. So, we can see that from the FTP1 V1R1 Master Version Process and the LOAD3 V1R1 Master Version Process, we cloned a File Instance Process for FILE0001.

From this chart, we can see that:

- FILE0001 consists of a FTP1 V1R1 process and a LOAD3 V1R1 process
- FILE0002 consists of a FTP1 V2R0 process and a LOAD3 V2R0 process

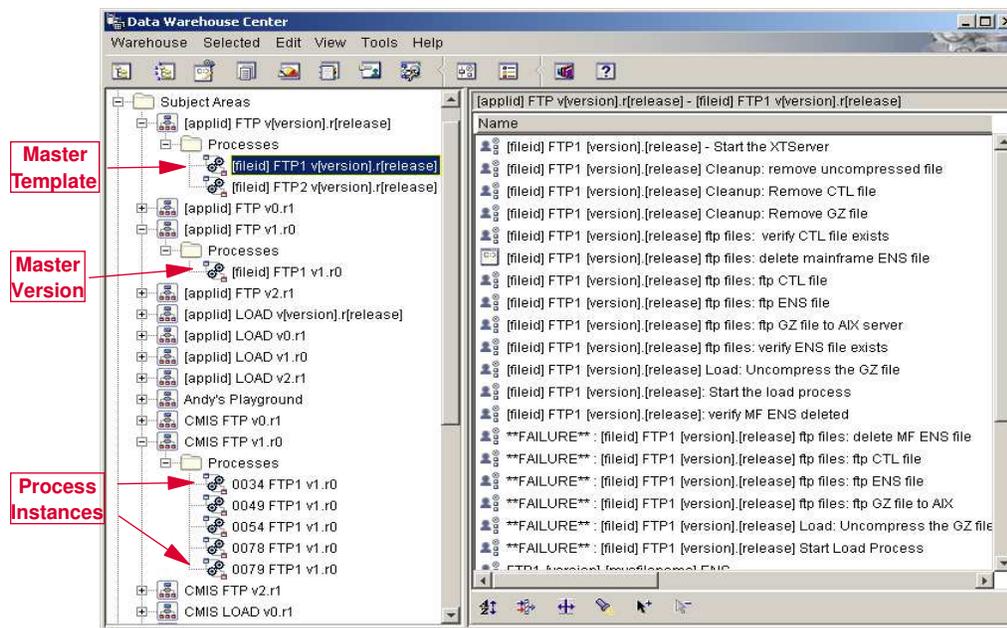


Figure 4-2 Master Templates, Master Version and Process Instances in DWC

Figure 4-2 shows us the Master Template for FTP1, the Master Version for FTP1 v1r0 and several Process Instances for specific fileids. The Process Instances are the processes that we can actually execute.

4.3 Developing the Master Template Processes

Developing the templates to be used in mass cloning can be complex. Besides ensuring that each cloned process and step has a unique name, we need to determine what changes from one file instance to another and determine how to tokenize these differences. Plus, having tokens embedded in the actual workflow imposes some other limitations and considerations. Let's review some of these considerations:

- After cloning, each process and step must have a unique name. In this case, just using a combination of [version] and/or [release] tokens is not sufficient. Fortunately, each source file already has an assigned numeric fileid. Therefore, to ensure unique names when we clone to the File Instance Process level, we also use a token for fileid, [fileid].
- It does not appear that we can use tokens for database and table object names. This may limit using certain type of steps, such as a SQL step as we have to connect table objects to these steps. We use alternative technologies that could be defined as User Defined Programs (UDPs) that would take database and table names as parameters. These parameters could then be tokenized. We used technologies such as SQL Stored Procedures, Shell Scripts and a C program that took a SQL statement as a parameter and processed it.
- We took pains to avoid creating blob files as part of the export process. That way we did not have to worry about moving those around during the cloning process and inadvertently losing or otherwise changing them. A blob file gets created when there is a SQL Step in which the SQL Wizard was used to create the SQL statement. In the cases where we were able to use a SQL Step, we still used the SQL Wizard to create the SQL Statement, but we did an 'edit' of the generated SQL statement and added a

blank character to the end of the statement. Once you edit the generated SQL statement, the SQL Wizard can no longer be used and a side effect of this is that the blob file does not get created.

- We had to ensure that each process was independent from an export utility viewpoint. We had to ensure that we had no data dependencies and we could not use shortcuts. This becomes easier with the new version of the export utility.
- We could not use shortcuts for two reasons. First, due to the capabilities of the export utility at that time, we wanted to avoid those process dependencies. But, even more important, was the fact that an ftp process may have to invoke a different load process depending on the source file. For example, FILE001 may need FTP1 and LOAD1 but FILE0002 may need FTP1 and LOAD3. So, we could not link these processes at development time with a shortcut but rather at the time we create the File Instance Processes during the cloning process, so the cloning process had to determine how to link the two processes. We ended up using the External Trigger mechanism as we could pass parameters to it and we could tokenize that parameter.

Keeping these considerations in mind we start development on our seven Master Template Processes. As you can see in Figure 4-3, this looks very similar to our previous example in that we put tokens in the process and step names: [fileid], [version] and [release].

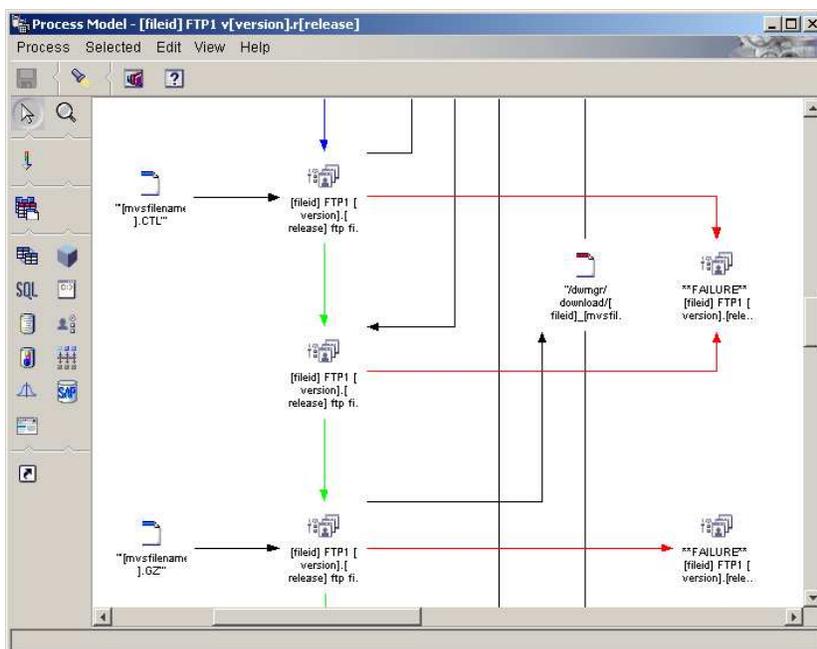


Figure 4-3 A snippet of the FTP1 Master Template Process

But, embedding tokens in the process and step names is not enough for Mass Cloning. We have to be able to embed tokens into the properties of the steps themselves. In Figure 4-4, we can see how we used tokens in the parameter list for a “Copy File using FTP” UDP. You can see tokens for the remote host name, [mvsipaddr], the remote userid [mvsuserid], the password [mvspassword]. You can also see that we use DWC tokens for the remote filename, &STBNS, and the local filename, &TTBN. These tokens will be resolved by the Warehouse Server at execution time by examining the file objects connected to the step with data flows.

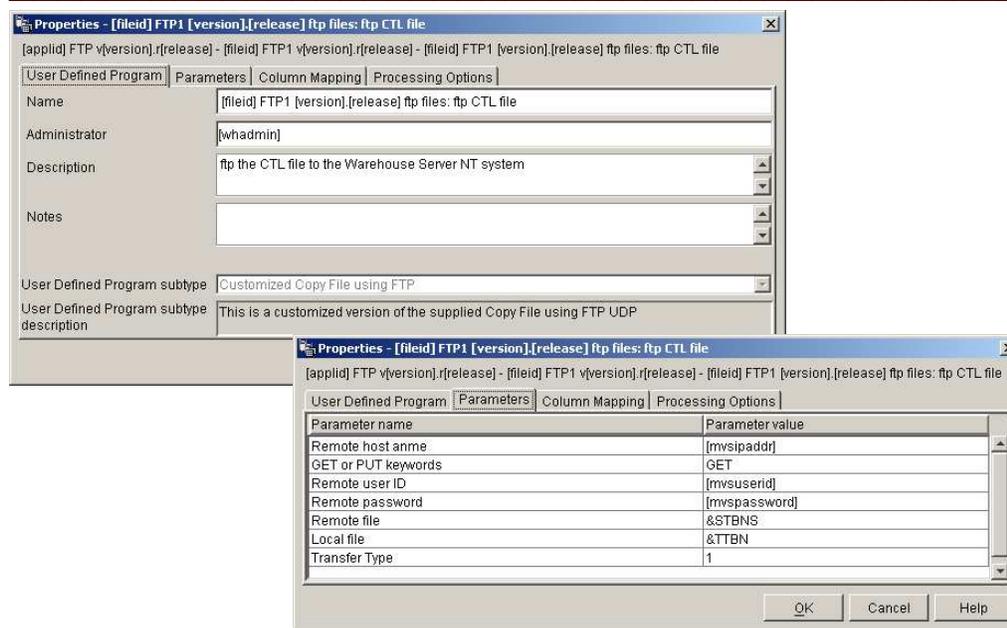


Figure 4-4 Tokens embedded in the properties of an ftp step

Please note that this is not the normal GUI interface to the “Copy File using FTP” UDP. We customized the registration of this UDP. The reason we did this is that the original step definition has the Password parameter as a parameter type of PASSWORD. When the export utility creates the tag file, any values associated with PASSWORD parameter types are stripped out of the tag file. Well, that just would not do for cloning since our password token would be removed from the tag file. If the passwords were stripped from the tag file, then, after cloning, we would have to go back in and edit each and every step that had a PASSWORD parameter by hand. To avoid that, we re-registered this program and defined the Password parameter as a CHAR parameter type. Therefore, when we export the process to a tag file, the password token is also included. Now, we can replace that token with the real password value during our cloning process. The drawback is that the password is in the clear.

4.3.1 Testing the Master Templates

During the development of our Master Templates, we will likely want to do testing as we develop the Master Templates. There could be several ways to approach the development process of these templates. We will look at two development approaches.

4.3.1.1 Develop a process then convert to a Master Template

One approach is to develop and test the process initially without using tokens and then convert the process into a Master Template. This may be a good approach when initially developing a brand new process. This is the easiest way to incrementally develop and test a process in a unit testing environment. In this approach, we do not need to know upfront what the tokens are. However, we will have to retrofit this process into a Master Template by adding the tokens. We may also find that there are some development changes because some of our steps may not lend themselves to be easily cloned.



After we have developed a Master Template and it needs to be modified, a process can be cloned for the purpose of modification. We would then modify this cloned process and test it. Again we will have to manually retrofit the changes back into the Master Template process.

4.3.1.2 Develop a process as a Master Template

This approach is to develop the process as a Master Template initially. We would then have to create a clone for testing. The advantage of this is that we are testing the fact that the process can be cloned. We need to understand what are tokens are upfront, or have the ability to create these as we develop the process. With this approach, we will be assured that the resulting process will be able to be cloned and we do not have the redevelopment effort to create a Master Process from the initial process. On the other hand, we will have to create a clone anytime we make a modification and want to test.

With this approach, modifications would be handled in a similar manner.

4.3.1.3 Combining the approaches

In reality, we will probably use a combination of these two approaches depending where we are in the development cycle. Early in the cycle, as we evaluate our process design, we will probably develop and test without tokens. However, as our process design solidifies, we will start developing our Master Templates. We can switch between the two approaches as needed.

4.4 Cloning

Now that we have one or more Master Template Processes developed, we have done the hard part. The process of the cloning is actually fairly straightforward. In this section, we will examine the cloning process and see how we may automate it. Again, this is in context of how this customer approached cloning, especially in the area of automating the process. While you may not be able to directly use this customer's tools, it does provide a starting point for you to base your own cloning process.

In this scenario, we actually have to clone twice. Once to create a Version Master from a Master Template and again when we need to create a Process Instance from the Version Master. Refer to Figure 4-1.

4.4.1 Manual cloning

Manual cloning is always a possible way to create clones. It is a low tech way to create clones. It is very feasible when you only need a few clones. However, it is more labor intensive and error-prone.

Manual cloning in this Mass Cloning example works just the same as in the simple example. In this section we will not go into the details of how to manually clone. We will explore the steps necessary to manually clone in this new example.



The first thing that we want to do is to create a Version Master of our Master Template Process. To do this, we simply export the Master Template Process to a tag file. We probably should not allow any blobs to be created. We will copy the tag file and edit it to replace the [version] and [release] tokens to the desired values. And, finally, import it back into the DWC. At this point, any version or release specific modifications can be made.

The next step would be to clone actual Process Instances from the Version Master. Again, we need to export the Version Master to a tag file. We may still have the tag file created during its cloning. We will copy the Version Master tag file and again edit it to replace the remaining tokens to their desired values. This can be a time-consuming process if we have used a lot of tokens in the process. It is also error-prone and is subject to the typing skills of the person editing the tag file. Once the tag file has been edited, we simply import the tag file into the DWC. Of course, to actually execute the steps, we have to promote the steps to test mode or production mode.

To create additional unique Process Instances, we simply clone each one from the Version Master.

4.4.2 Automating the cloning process

As we see in the previous section, manually cloning processes can be very effective if our requirements are small. However, this customer needed to create hundreds of process from a set of seven templates. Cloning hundreds of processes manually would consume too much time and is virtually impossible to do error free. And this would have to be repeated every time we have a software release. So we had to find some way to automate cloning. This section will examine how we accomplished this automation.

Referring back to Figure 4-1 and to the previous section, we can see that there are some obvious places where we can automate the process. First, we have an opportunity to automate the process of cloning a Version Master from the Master Template. Second, we have an opportunity to automate the cloning of a Process Instance from a Version Master. However, we have another opportunity for automation that may not be obvious which is the promotion of steps to test mode and then to production mode. With tens of thousands of steps in hundreds of processes, manually using the DWC GUI to promote these steps would take a lot of effort and time.

Fortunately there are some batch command capabilities with WHM that allows us to create some Windows command or bat files that can help us automate the process. Figure 4-5 shows where in the cloning process we were able to provide some automation tools. We have a bat file that clones a Version Master from the Master Template, cre8ver.bat. There is another bat file that clones a Process Instance from the Version Master and there is a bat file that promotes or demotes the steps of a process.

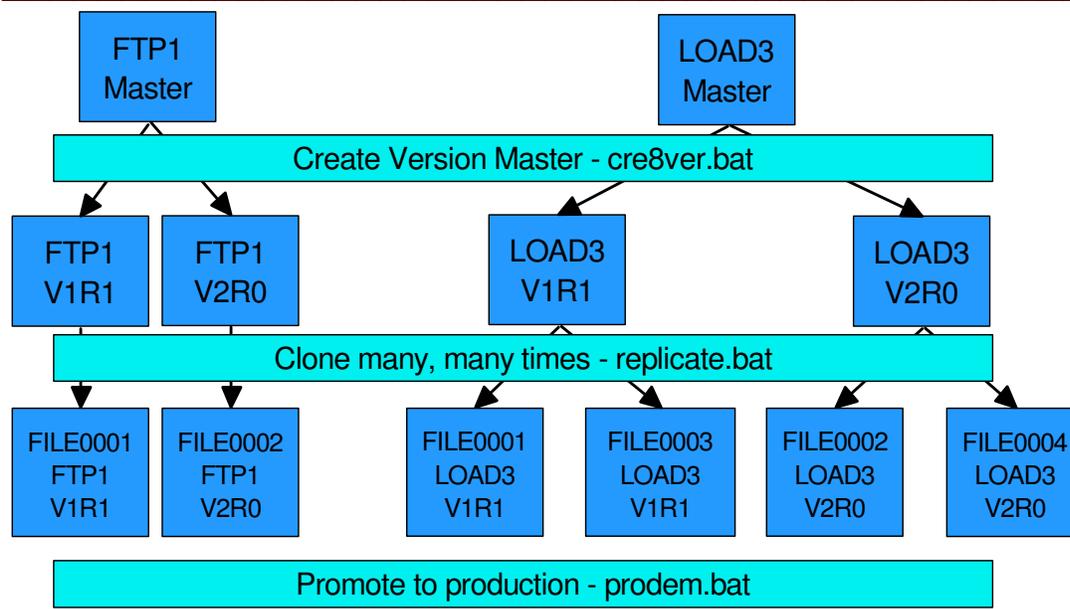


Figure 4-5 Points of automation in the cloning process

We will now examine each of these bat files to see how they work.

4.4.2.1 Technologies used

To accomplish automating cloning, we need to have command or batch versions of the manual tasks that we used in the Manual Cloning method.

First, we need to export processes to tag files. In the manual method, we just used the DWC GUI to accomplish the export. There is a command interface to the export utility, `iwh2exp2`, which we will use in our automated method. Conversely, we need to import the cloned tag file and we will use the command interface to the import utility, `iwh2imp2`.

Second, we need a way to promote and/or demote steps between development mode, test mode and production mode. For this, we will use WHM's external trigger mechanism. This allows us to promote and demote steps from a command line.

Finally, we need a way to accomplish global find and replace actions on a text file from a command line interface. This proved to be a bit tricky to find and at the right price. In the Windows environment, our search found several nice editing tools, but they were all graphical based. What we wanted was a capability like the `sed` stream editor in Unix. Fortunately, we found a Windows port of a number of GNU Unix utilities in a package call `unxutils` which can be found at a number of sites on the web.

With this set of tools, we were ready to begin our journey to automation.

4.4.2.2 Cloning a Version Master from a Master Template – `cre8ver.bat`

The purpose of this bat program is to create a Version Master from one Master Template.



When executing this program, we need to know which Master Template to clone and the replacement values for [version] and [release]. Recall that we have seven templates that fall into two categories, FTP and LOAD. We will take advantage of having two template types. So, the input parameters to the cre8ver.bat program are:

1. Process Type: FTP | LOAD | SMRY Note: SMRY is for future use
2. Process Type Suffix: A string to append to the Process Type: 1, 2, 3, etc
3. Version: A string to replace the [version] token
4. Release: A string to replace the [release] token
5. Home directory path: path to store the tag files

Invocation: **cre8ver LOAD 3 1 5 c:\tagfiledir would** clone the LOAD3 Master Template and create the LOAD3 v1r Version Master

We also want the flexibility to have the Master Template in one control database with the Version Master in a different control database. Therefore, we need information about the two control databases. The cre8ver.bat program needs to be customized.

```
set scdb=src_cdb
set scuser=src_user
set scpwd=src_pwd

set tcdb=tgt_cdb
set tcuser=tgt_user
set tcpwd=tgt_pwd
```

We want to verify our parameters and set up the variables that we will use during the execution of the program.

```
REM Check parms

if "%5"==" " goto parmsreqd

REM Setup variables

set proctype=%1
set procsufx=%2
set version=%3
set release=%4
set homedir=%5
set workdir=%homedir%\tagfiles\templates\version_%version%%release%
set filename=create_%proctype%%procsufx%
set logfile=%homedir%\logging\%filename%.log
set inpfiler=%workdir%\%filename%.inp
set tagfile=%workdir%\%filename%.tag
set tagfile1=%workdir%\%filename%.1
set newtagfile=%workdir%\%proctype%%procsufx%_process_v%version%%release%.tag
set db2sql=%workdir%\%filename%.sql
set db2rpt=%workdir%\%filename%.rpt
set db2rpttemp=%workdir%\%filename%.rpttemp
```

We next need to do some preliminary work to possibly set up directories, erase old files and log some information.

```
REM try to make the directory
```



```

md %workdir%

REM erase all old work files if they still exist

erase /Q %workdir%\%filename%.*

REM erase old tag and import log files

erase /Q %workdir%\%proctype%\%procsufx%_process_v%version%\%release%.*

REM create logfile and log setup info

echo crever starting > %logfile%
echo PARMS: >> %logfile%
echo Process Type: %proctype% >> %logfile%
echo Process Suffix: %procsufx% >> %logfile%
echo Version to create: %version% >> %logfile%
echo Release to create: %release% >> %logfile%
echo Work directory: %workdir% >> %logfile%
echo INP file: %inpfile% >> %logfile%
echo Tag file: %tagfile% >> %logfile%
echo New Tag file: %newtagfile% >> %logfile%

```

Now, we can export the Master Template to a tag file. The batch version of the export utility uses a control file that tells what processes to export. We decided to create this on the fly. Therefore, we need to know the full name of our Master Template Process. We could make an assumption, or we can query the control database. We chose to query the control database.

```

REM create the INP file with the correct process name based on the process type and process suffix

echo connect to %scdb% user %scuser% using %scpwd% > %db2sql%
echo select name from iwh.process where name like
'%%%proctype%\%procsufx%\%[version]\%[release]%%%' >> %db2sql%
db2cmd -c -i -w db2 +o -s -l %logfile% -r %db2rpt% -f %db2sql%
type %db2rpt% | sed -e 11!d -e "s/[ ]*$//" > %db2rpttemp%

echo ^<PROC^>> %inpfile%
type %db2rpttemp% >> %inpfile%
echo ^<IR^>>> %inpfile%
echo ^<SCHEMA^>>> %inpfile%
echo ^<UDP^>>> %inpfile%

REM execute iwh2exp to export the process to a tagfile

iwh2exp2 %inpfile% %scdb% %scuser% %scpwd% /S

```

Now that we have our tag file, we can edit it and replace the [version] and [release] tags creating a new file in the process. We use a simple invocation of the sed stream editor to accomplish the global find and replace. Sed is a very powerful utility typically found in Unix environments but has been ported to the Windows environment by the GNU organization. It has a very powerful 'programming' language which, in the Unix tradition, seems very complex. We use sed to do a global find and replace to replace the [version] and [release] tokens to values provided in the parameters. For more information on sed, you can search the web to find various FAQs and documents, check any Unix OS documentation or consult one of the many books written about sed.

```

REM substitute the [version] and [release] tokens

```



```
type %tagfile% | sed -e "s\[version\]/%version%/g" -e "s\[release\]/%release%/g" > %newtagfile%
```

We import the new tag file into the target control database.

```
REM import the new tagfile into the control database

iwh2imp2 %newtagfile% %workdir% %tcdb% %tcuser% %tcpwd%
```

Finally, we finish up our bat program and exit.

```
goto end
:parmsreqd
echo Invalid parameters...usage: crever proctype procsuffix version release workdir >> %logfile%
REM exit 8

:end

REM erase all old work files

erase /Q %workdir%\%filename%.*

echo *** cre8ver finished *** >> %logfile%
REM exit 0
```

4.4.2.3 Cloning a Process Instance from the Version Master – replicate.bat

This is the workhorse bat program which clones the Process Instances from the Version Masters. It is also the most complicated. This will be invoked one for each source file, which could be hundreds of times, and it will build all of the processes necessary to get the source file into the warehouse database.

This program assumes that there is a current export of the Version Master in the appropriate directory. We do not want to export the tag file every time this program executes because we would be exporting the same process over and over, which is unnecessary overhead.

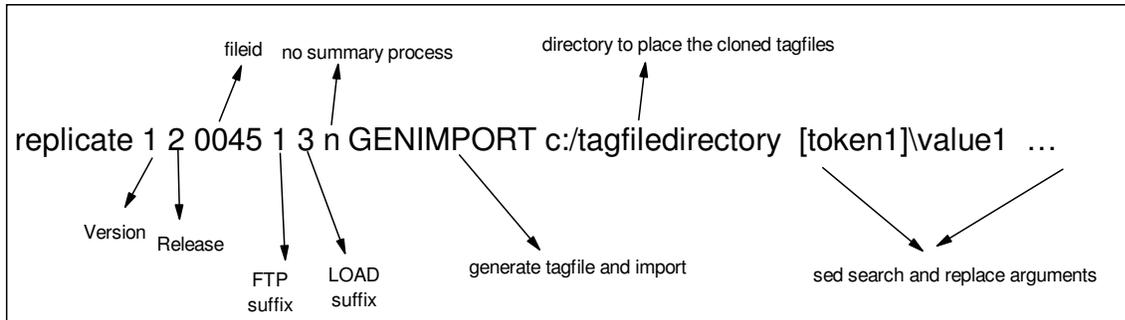
For each source file, we want the program to clone the appropriate FTP process, the appropriate LOAD process and, in the future, the appropriate SMRY process. We want to be able to include or exclude any of the process types. We also need to know what version and release to use and all of the replacement values for all of the tokens. There could be many tokens. We also want to be able to generate the tag file only or to generate the tag file and load into the control database. It is conceivable that we may not have connectivity to the target database, so this would allow us to create the tag files, move them to the target machine and then import them.

The parameter list consists of a set of positional parameters, followed by a variable number of parameters. The positional parameters are:

1. Version: version number used to select the Master Processes
2. Release: release number used to select the Version Master Processes
3. Fileid: fileid of the source file and will replace all occurrences of the [fileid] token
4. FTP process suffix: along with the version and release parameters identifies which FTP Version Master will be used

5. LOAD process suffix: along with the version and release parameters identifies which LOAD Version Master will be used
6. Summary process suffix: for future use but similar function as the FTP and LOAD suffix parameters
7. Action: GENONLY | GENIMPORT Create tag file only or create tag file and import
8. Home directory: where to find and save tag files

After parameter #8 is a variable number of parameters in the form of sed search arguments. There is one search parameter for each token.



Example: **replicate 1 2 0045 1 3 n GENIMPORT c:/tagfiledirectory [token1]\value1 ...** would create a FTP1 v1r2 process and a LOAD3 v1r2 process for source file id 0045. A Summary process would not be created. Tokens would be replaced as defined...i.e. value1 would replace all occurrences of [token1] and so on.

We need information about the source control database and, if the action is GENIMPORT, about the target control database. The source control database and the target database may be different or the same.

```
@ECHO OFF

set scdb=src_cdb
set suser=src_user
set spwd=src_pwd

set tcdb=tgt_cdb
set tuser=tgt_user
set tpwd=tgt_pwd
```

We want to verify our parameters and set up the variables that we will use during the execution of the program.

```
if "%9"==" " goto invalidparms

set version=%1
set release=%2
set fileid=%3
set ftp=%4
set load=%5
set smry=%6
set action=%7
set homedir=%8
```



```

if /i %action% EQU GENONLY goto setgenonly
if /i %action% EQU GENIMPORT goto setgenimport
goto invalidparms

:setgenonly
set import=N
goto continue1

:setgenimport
set import=Y
goto continue1

:continue1

set tagdir=%homedir%\tagfiles
set masterversiontagfiledir=%tagdir%\templates\version_%version%%release%
set deleteENSfile=%tagdir%\templates\master\delete_ENS.ftp
set outputtagfiledirectory=%tagdir%\%tcd%version%version%%release%
set outputscripfiledirectory=%homedir%\scripts\ftp_delete
set logdir=%homedir%\logging

set repllog=%logdir%\repl%fileid%.log
set sedcmdfile=%outputtagfiledirectory%\%fileid%_sedcmd.tmp

```

We create log entries and needed directories.

```

echo Starting Log %repplog% > %repllog%

echo Version: %version% >> %repllog%
echo Release: %release% >> %repllog%
echo File ID: %fileid% >> %repllog%
echo Ftp process: %ftp% >> %repllog%
echo Load process: %Load% >> %repllog%
echo Summary process: %smry% >> %repllog%
echo Action: %action% >> %repllog%
echo Home directory: %homedir% >> %repllog%

echo Master version tag file directory: %masterversiontagfiledir% >> %repllog%
echo Delete ENS file: %deleteENSfile% >> %repllog%
echo Tag file directory: %outputtagfiledirectory% >> %repllog%
echo Output script file directory: %outputscripfiledirectory% >> %repllog%

REM make the tag file directory

md %outputtagfiledirectory%

```

This section will read in the variable list of parameters and set up the control file for the sed utility. In this program we use a sed control file to provide the entire find and replace commands as it is too unwieldy to do that via the command line option.

```

REM
REM finish with the parms by
REM starting at parm 9, generate file that contains the commands to sed
REM

REM first setup the fileid search command

echo *** Generating sed substitute commands *** >> %repllog%
echo s^[fileid\]/%fileid%/gi >> %repllog%
echo s^[fileid\]/%fileid%/gi > %sedcmdfile%

```



```

echo s/[loadtype]/%load%/gi >> %sedcmdfile%

REM
REM special substitute for taking care of changing the first step in the ftp step to convert
REM the string FTP1 or FTP2 to FTP... this is for the use of the ftp driver program
REM

echo s/FTP%ftp%-%ver%/FTP-%ver%/g >> %sedcmdfile%

:nextsedcmd
set searchcmd=%9
if "%searchcmd%"==" " goto chkftp
echo s/%searchcmd%/gi >> %repllog%
echo s/%searchcmd%/gi >> %sedcmdfile%
shift /8
goto nextsedcmd

```

We need to check the FTP parameter to determine if we are generating the FTP process and, if so, the FTP suffix. We execute the sed utility to do the find and replace of the tokens and save it in another directory. We use the sed control file created above. We also create something called the delete_ENS ftp script file. This is something we need during the execution of our FTP process and is outside the scope of this paper. But, notice that we use the sed utility and the same sed control file to create it. However, we don't necessarily have all of the tokens that we have in the tag file but it is ok because, if sed does not find the token, it just ignores that command and continues with the next.

```

REM
REM check to see if we need to generate the ftp process
REM

:chkftp
if /i %ftp% NEQ n goto genftp
goto genload

:genftp

REM
REM create the ftp tag file for this fileid
REM

echo Generating the ftp process for %fileid% >> %repllog%
sed -f %sedcmdfile% %masterversiontagfiledir%\FTP%ftp%_process_v%version%%release%.tag >
%outputtagfiledirectory%\%fileid%_ftp%ftp%_process_v%version%%release%.tag

REM
REM create the delete_ENS ftp script file
REM

echo Generating the delete_ens ftp script for file %fileid% >> %repllog%
sed -f %sedcmdfile% %deleteENSfile% > %outputscriptfiledirectory%\%fileid%_delete_ENS.ftp
echo ftp tag file and delete_ens script created >> %repllog%

```

We check to see if we need to generate the load process and, if so, create it using the sed utility.

```

REM
REM create the load tag file for this fileid
REM

```



```

:genload
echo Generating the load process for file %fileid% >> %repllog%
if /i "%load%"=="n" goto gensmry
sed -f %sedcmdfile% %masterversiontagfiledir%\load%load%_process_v%version%%release%.tag >
%outputtagfiledirectory%\%fileid%_load%load%_process_v%version%%release%.tag

echo Load tag file created >> %repllog%

```

This is to create a summary process tag file in the future.

```

REM
REM create the summary tag file for this fileid
REM

:gensmry
if /i "%smry%"=="n" goto importtags
echo Generating the summary process for file %fileid% >> %repllog%
if /i "%smry%"=="n" goto import
echo Summary processing not valid at this time... >> %repllog%

```

If our action parameter is GENIMPORT, we need to import the generated tag file(s) into the target control database.

```

REM
REM check to see if we need to do imports
REM

:importtags
if /i %import% NEQ Y goto end
echo Beginning import of tagfiles for file %fileid% >> %repllog%

REM
REM import the ftp process
REM

if /i %ftp% EQU n goto impload
echo importing ftp tag file for %fileid% to the %tcd% Control Database >> %repllog%
iwh2imp2 %outputtagfiledirectory%\%fileid%_ftp%ftp%_process_v%version%%release%.tag %logdir% %tcd%
%tuser% %tpwd%
echo Finished importing ftp process

REM
REM import the load process
REM

:impload
if /i %Load% EQU n goto impsumry
echo importing load tag file for %fileid% to the %tcd% Control Database>> %repllog%
iwh2imp2 %outputtagfiledirectory%\%fileid%_load%load%_process_v%version%%release%.tag %logdir%
%tcd% %tuser% %tpwd%
goto end

REM
REM import the summary process
REM

:impsumry
echo importing summary tag file for %fileid% to the %tcd% Control Database>> %repllog%
echo Summary processing not valid at this time... >> %repllog%
goto end

```



Finally, we finish up our bat program and exit.

```
:invalidparms
echo one or more parms is missing ... exiting program with rc 8
REM exit /b 8
goto exit
:end
echo program finished ... exiting program with a rc 0 >> %repllog%
REM exit /b 0

:exit

erase /Q %outputtagfiledirectory%\*.tmp
```

4.4.2.4 Promote/Demote steps

This bat program will promote or demote steps between development mode, test mode and production mode within the DWC.

This program uses the WHM External Trigger process to accomplish the promotion or demotion. To do this we call the client portion of the External Trigger mechanism which will promote or demote one step one level. If we have tens of thousands of steps, we don't necessarily want to have to explicitly call this program tens of thousands of times. What we would like to be able to do is to provide a process name search string and promote/demote all steps in all of the processes that match the search string. We use this search string to query the control database to find all matching process names and all steps of these matching processes.

We need to know if we are going to promote or demote these steps and to what mode. We have four possibilities: Promote to test mode, Promote to production mode, Demote to test mode, Demote to development mode. We also need a search string for the process name. The parameters are:

1. Action: PT | PP | DT | DD where PT = promote to test mode, PP = promote to production mode, DT = demote to test mode, DD = demote to development mode
2. Search string: any DB2 style search string, including wildcards
3. Work directory: for temporary work files
4. RunID: used in names of temporary files

Invocation: **prodem PT FTP1 c:\workfiles** will promote all steps to test mode for all FTP1 processes, all versions and releases.

This program needs to be customized to provide information about the control database as well as information needed to connect to the External Trigger Server.

```
@ECHO OFF

set scdb=tgt_cdb
set scuser=tgt_usr
set scpwd=tgt_pwd

set whserver=192.168.0.101
```



```
set whport=11004
set whuser=tgt_user
set whpasswd=tgt_pwd
set whwait=1
```

We want to verify our parameters and set up the variables that we will use during the execution of the program.

REM Check parms

```
if "%4"==" " goto parmsreqd
```

REM Setup variables

```
set action=%1
set searcharg=%2
set workdir=%3
set runid=%4
set db2sql=%workdir%\%runid%_sql.tmp
set db2rpt=%workdir%\%runid%_rpt.tmp
set logfile=%workdir%\%runid%_prodem.log
set prodemfile=%workdir%\%runid%_prodem.bat
```

REM erase the old files

```
erase %workdir%\%runid%_*
```

REM create logfile and log setup info

```
echo prodem starting > %logfile%
echo PARMs: >> %logfile%
echo Action: %action% >> %logfile%
echo Search Argument: %searcharg% >> %logfile%
echo Work directory: %workdir% >> %logfile%
echo Runid: %runid% >> %logfile%
```

We first need to determine what action is being requested and convert it to what XTClient expects. We could have used the same parameter values as XTClient but we wanted to use parameter values that were a bit more descriptive. For example, we thought that using PT for promote to test was a bit more meaningful than using the numeral 2.

```
set whcommand=none
if /i %action% EQU PT set whcommand=2
if /i %action% EQU PP set whcommand=3
if /i %action% EQU DT set whcommand=4
if /i %action% EQU DD set whcommand=5
if %whcommand% EQU nont goto parmsreqd
```

We need to query the control database to get our list of steps. We create the connect and sql statement based on parameters and variables and echo to a sql script file. The sql statement actually creates the External Trigger client commands. We then execute the sql script, returning the results in a report file. Note that the syntax to invoke the XTClient changes in DB2 V8 from **java XTClient** to **java db2_vw_xt.XTClient**.

```
Echo connect to %scdb% user %scuser% using %scpwd% > %db2sql%
```



```
echo select 'java XTClient %whserver% %whport% %whuser% %whpasswd%' concat      bv.name
concat "' %whcommand% %whwait%' from iwh.businessview bv, iwh.process p, iwh.relationship r where
p.name like '%searcharg%' >> %db2sql% and p.iwhid=r.source_iwhid and bv.iwhid=r.target_iwhid >>
%db2sql%
```

```
db2cmd -c -i -w db2 +o -s -l %logfile% -r %db2rpt% -f %db2sql%
```

We now have a sql report that contains the External Trigger command for each step name returned. This report has extra lines, headers and such that we need to eliminate. Which we accomplish with the sed utility which creates the bat file. We then call the bat file to be executed.

```
type %db2rpt% | sed -n -e /java/p > %prodemfile%
```

```
cat %prodemfile% >> %logfile%
```

```
call %prodemfile% >> %logfile%
```

Finally, we finish up our bat program and exit.

```
goto end
:parmsreqd
echo Invalid parameters...usage: prodem action searcharg workdir runid >> %logfile%
REM exit 8

:end

echo *** prodem finished *** >> %logfile%
REM exit 0
```

We would repeat this process as needed to get the steps to the desired mode. For example, to get the steps for a newly cloned set of processes to production mode, we would execute prodem with an action of PT followed by an execution with an action of PP.

4.5 Summary

We have explored how we used an automated mass cloning technique at this customer to clone well over 300 processes with nearly 10,000 steps from seven process templates which are currently in their production environment. At the time of this paper, they are well into their phase 2 which could quadruple the number of processes and steps

A note from the author:

I developed these cloning techniques while working with this customer project as an ETL consultant and the Warehouse Manager developer. While we were designing the ETL workflows, running tests and finally developing the templates, I evolved the cloning techniques from a mere hope, to a manual process and finally to an automated process over a period of 4-6 weeks. Even though this is a large customer, the development team was small. It consisted of a Project Manager, a part-time Architect, the customer's technical lead, me and on demand resources to develop SQL Stored Procedures and AIX shell scripts.



As I understand, it took about 6 months from development start to first production. They ran in parallel with the current system for a couple of months before turning off the old system completely. I was involved for about 9 weeks during the heaviest period of development.

5 Moving processes from the development environment to the test environment to the production environment

Most customers do not develop, test and run production in one environment. It is typical to have a development environment, a test (or QA) environment and a production environment. Each environment has its own Warehouse Manager Server, Control Database and Warehouse Database. The development environment is used for developing and unit testing individual processes. The test environment is used for system and/or user acceptance testing. Then, of course, the production environment is where the final set of processes will actually execute.

In this section, we will explore how this customer moves processes between environments and how they use cloning to help. We will then present a slight twist on this idea.

5.1 The strategy

As part of the overall picture, we needed a strategy of moving processes to the different environments. At this customer, the test environment is basically a mirror image of the production environment even with the same names of objects like database names, schema names, agent names, etc. The primary difference is with userids and their passwords.

We used the development environment for general development activities and the development of the Master Template processes. We also used the development environment to store the cloned Version Masters. For unit testing, we also cloned Process Instances into the development environment. We did not clone all 300 processes and 10,000 steps into the development environment, rather we cloned only what was needed for unit testing. See Figure 4-2 for a screen shot from our development DWC GUI.

When we were ready to create a software release for system/user acceptance testing, we would start with a brand new control database. Into this database we would clone all 300+ processes and promote all steps to production mode. We would have to manually modify the agent and database userids and passwords. Once this software release passed testing, we would use db2move to copy over the entire control database into the production environment. Of course, we would have to again manually change the appropriate agent and database userids and passwords.

For subsequent releases to the test environment, we made a decision to re-clone into an empty test control database and move the entire database into the production environment with db2move. We did this for several reasons. First, any change to a Master Template Process would affect many Process Instances. If we import a modified process that has deleted steps and imported over an already existing process, we could end up with orphaned steps in the test control database, and subsequently, in the production control database. This would not affect the actual execution of the processes but, over time, could clutter the control database. To

eliminate orphans, we could first delete the existing processes before importing. Unfortunately, at the time of writing, there is no batch method to delete steps and processes from a control database.

There is one side point about using db2move to copy the test control database to the production environment. WHM records in its log table, some runtime information and statistics. When we replace the production control database with the test control database, we lose that information. If that is a concern, we could save the contents of the production log information. However, a db2move has less impact on our schedule than re-cloning everything into the production control database. At the time of this paper, there are some techniques in development by Dr. Paul Wilms on how to create a long-term history of runtime information. Conceivably, this historical information could be exported before the db2move of the test control database to the production environment. Then it could be imported into the new production control database to preserve this runtime history.

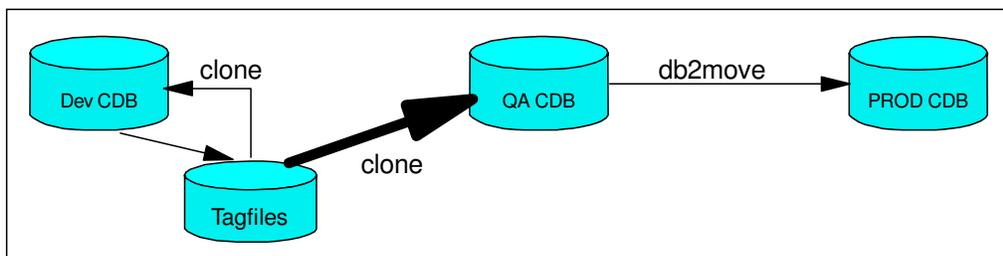


Figure 5-1 Managing process cloning and movement

5.2 A small twist on this strategy

There is a danger in storing the Master Templates and Version Master processes in the development control database. They could inadvertently get modified or deleted. This happened to us. We were lucky in that we had recently cloned and still had the tag files. We learned to nightly export the Master Templates and Version Master processes into a backup directory which we automatically scheduled using the DWC scheduling facility.

Perhaps a more secure alternative is to create a separate control database just to contain the Master Templates and Version Masters. This control database would never be an active control database for a Warehouse Server. We could also limit update of this database to certain personnel.

We would still use the development control database for the actual creation and modification of Master Templates. Once a Master Template is ready, then we would simply export it from the development control database and import it into this Master control database. When we need to clone a new Version Master, the source and target control databases would both be the Master control database. Then when a developer needs a Process Instance for unit testing, then it can be cloned from the Master control database to the development control database. When we are ready for a software release to testing, we would then clone from the Master control database to the test control database. After that, the process is the same.

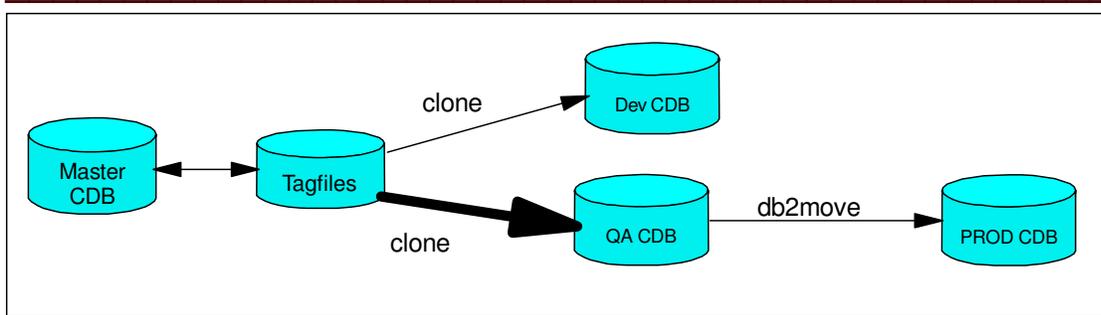


Figure 5-2 Using a Master control database

6 Summary

In this paper we explored how to clone Warehouse Manager processes by using two examples, one a simple cloning technique to version processes and a much more complex cloning technique developed for a large customer to massively clone WHM processes. Most customers won't have the need to clone on such a large scale, but the techniques discussed can help to develop their own cloning strategy.