Evolution of a new TeamConnection Family, Common do's and don'ts

Document Number TR 29.2333

Lee Perlov

TeamConnection/CMVC Development
IBM Software Solutions
Research Triangle Park, North Carolina

ABSTRACT

TeamConnection families evolve over time. As the family matures, things tend to change. New TeamConnection family administrators tend to do many similar things right and wrong. This document traces the growth of a typical TeamConnection family and reflects on these choices.

ITIRC KEYWORDS

- TeamConnection
- family administration

ABOUT THE AUTHOR

Lee R. Perlov

Mr. Perlov is a Staff Programmer in the TeamConnection/CMVC development group. He started working for IBM in 1985 in Gaithersburg, Md, in the Federal Systems Division on various projects for the United States intelligence community. He then moved to RTP to work on library development and support.

Mr. Perlov received a B.S.Acc degree in Accounting from the University of Florida in 1983. He also completed two years of graduate work in the Department of Computer Science at the University of Florida.

Contents

Introduction

The TeamConnection services team, one of the departments in our development group, recently created a new TeamConnection family to store their documents, class materials, tools, etc. It was created on Windows NT because there was an available machine. Also, no experienced family administrator was readily available to manage the family. As a result, this family provides an excellent example of what tends to happen as a new family evolves.

Goals

During my time in CMVC and TeamConnection development, I have helped many new family administrators make important choices about their families. I have developed a few guiding principles that I try to instill in new family administrators.

Good System Administration

Since a family is only as reliable as the system it runs on, you must have a good system administrator. A good systems administrator does all of the following:

System Maintenance

For reliable system behavior reboot regularly, at least weekly. Also, check the file systems regularly to make sure that files that grow during normal system operations (e.g. audit.log) are archived and pruned.

Good Backup

Mistakes happen - be prepared. You need regular backup of your family and your system, periodic archival of these backups, and testing of your recovery procedures. It also helps to document the configuration.

Good Security Standards

Enforce reasonable security standards. You need to protect your data and the integrity of TeamConnection processes. It is important to minimize the number of super users. In particular, it is important to maintain tight control of root password in Unix and physical system access in Windows NT and OS/2.

Appropriate Processes

Don't overwhelm new users - start with simple processes. TeamConnection allows for changes to processes at any point in the development cycle. Start with simple processes so that users can get used to the new tool, then add to the processes as you get closer to delivering your product (i.e. when the cost of a mistake increases).

Once your TeamConnection processes include defects/features and drivers, TeamConnection is helping you deliver your product in a well defined, repeatable manner, and you have a complete history of your changes. This is a mature process configuration.

Small Family Planning

Most TeamConnection families are small families, or at least they start that way. Lots of things will change over time. During the evolution of a new family, many things go right and some tend to go wrong. This document is a chance to identify common mistakes and opportunities. Just as important, it is a chance for new family administrators to feel a little less alone making their decisions. Hopefully, my experience with a new family for our services group will provide some suggestions that will minimize the difficulties associated with learning new things and evolving.

Large Family Planning

In almost all cases, when you are populating a new TeamConnection family from a large established codebase, you will take the time to do plenty of planning and preparation. Usually, there is an experienced system administrator that can lend advice, and in many cases an IBM consultant participates.

If anything, the biggest problem that occurs when setting up a new large family is that too many of the available configuration options are used and the team members are overwhelmed by the changes to the way they do work. Team members are asked to add new steps to their development processes, without a clear understanding of the benefit.

The one case where overwhelming change is not a risk is when you are migrating a family from CMVC to TeamConnection. CMVC family administrators can move their existing processes and customization directly to TeamConnection. The primary challenges for these family administrators and users are taking advantage of the new capabilities in TeamConnection and changes to the names of several objects. For example, CMVC Tracks become TeamConnection Workareas. Overall, this group has the lowest risk of "new family" problems of any new TeamConnection user group.

Even with planning, things can go wrong. Therefore, even family administrators for new large TeamConnection families should find something useful in this document.

Creating the family

Your initial decisions when creating a family are crucial to the acceptance of configuration management by the development team, and their opinion of TeamConnection as a help or a hindrance to the development of their own product.

When you start setting up a new TeamConnection server, I recommend creating a sandbox family. For example, we provide `univwin` to populate a TeamConnection family on a Windows NT server. It creates objects, runs builds, and sets the security to none so that anyone can play in the family. Even after you have your production family up and running, you can come back to the sandbox whenever you need to try something new. Also, `univunix` is available for Unix and `univos2` for OS/2.

In the case of our services family, we already had lots of sandboxes and production families available. After all, we are the developers of TeamConnection and we use it for our own product development every day! So when we needed a family to store the materials for our TeamConnection Administrator's Class and our TeamConnection Developer's Class, we decided to create a new family and administer it ourselves.

The next step was to decide what operating system to use as the server for our family. Since everyone in the services department travels with a Windows NT laptop that has the TeamConnection Windows NT server installed, we picked an underutilized 150Mhz Pentium PC running Windows NT Workstation 4.0.

One of our team members did the initial installation and setup. Here were the relevant decisions and a few comments about them.

Create users

Initially, we created every one of our users with super user authority. Since we are all experts on TeamConnection, why not have all authority? While this arrangement could easily work for us, it isn't what we recommend to customers when we provide consulting services. We find this a very common mistake among new users of TeamConnection. While it may provide faster startup, we recommend setting accesses to components properly in order to avoid future problems.

Create components

Among the components created were three for our class materials. Since components are used for defining processes, managing releases, managing defects and features and assigning user access, we created one component for each class and another parent component to group the classes together.

Some of the advantages of creating these components in our family were:

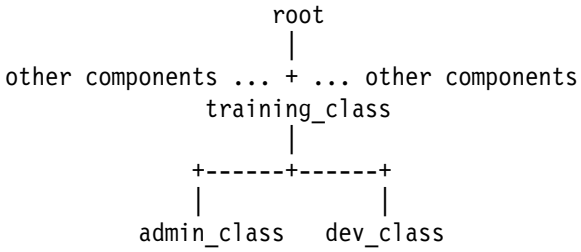
- We can assign user access for both classes from the common `training_class` component.
- We can put parts that are unique to the "dev" or "admin" class in their own component, and common parts in training_class.
- We can assign defects and features to the component for a specific class or to the common component.

- We can create unique releases for each class, with their own component processes (e.g. defect/feature, design-size-review and verify).
- It is easy to change the components that owns parts, link releases and parts in releases, and delete obsolete components. So, any decision we made could easily be changed.

Some of the drawbacks to the decisions for our family were:

- There may be too few parts to justify keeping track of three components.
- Many or most of the parts may be common to the two classes, in which case we would spend extra effort because we would need to create and use a workarea for each release every time we modified a common part.

Initially, we chose the `prototype` process for our components. This meant that we would not use any defects or features for creating and updating our parts.



Create releases

As I alluded to already, we decided to create one release for each class, each one in their own component. Again, we selected the `prototype` process for our releases. Further, we created only one workarea for each release and we all shared it. Since we were each checking in unique parts, we didn't think any of the parts would be common across both releases. This sounded like an easy and safe way to start.

The advantages of this arrangement in our family were:

- In TeamConnection, we only need to create and edit parts. We could set our component, release and workarea in the GUI settings page and all of our changes could use these defaults.
- We only needed to use one of the TeamConnection GUI dialogs, Parts.
- We could defer all other decisions about process until later.

The disadvantages of this arrangement for our family were:

- Whenever anyone wanted to create a new version of their part changes using `workarea -freeze`, it versioned the changes for everyone using that workarea.
- We delayed committing our changes to the release, so we didn't really have a baseline for our releases.
- We decided later that several parts would be common, so we had to reference both releases when checking in changes.

All in all, it was a good choice.

Early bumps and bruises

When creating a new family, some things tend to be overlooked or put off until later. Here are the problems we ran into with our services family, and what we did to correct them. This is a chance for us to put our early problems in perspective and emphasize how the simple actions we took can make these problems disappear.

Team members losing each other's changes

We started out sharing a workarea for each release. Also, we all had super user access. As a result, it was easy to unlock a file when another team member had it checked out. Finally, we weren't using defects and features, so we didn't keep any records of why we were making our changes.

So, when it came time for us to edit each other's parts, we had some confusion and some lost changes. This was only a minor problem, but it was an indication that our processes needed to start "growing up".

Solution

- Work in your own workareas

  TeamConnection allows you to share changes between workareas using `workarea -refresh` and `part -link`. Since it takes only a few seconds to create your own workarea, it is not necessary to share workareas.

- Do not use super user accounts for normal development

  A super user can circumvent the TeamConnection development processes you configured to insure effective teamwork. In one case, after we added drivers to the process, a super user inadvertently integrated a workarea directly into the release, bypassing the driver and corrupting our release baseline.

  In the case of our family, most of the members of the team are also TeamConnection experts. As such, they want to keep a super user account. The best way to accommodate them is to remove super user access from their regular account, then create a new super user account that they will use less frequently.

  I always recommend that the login name reflect super user authority. So, I have a super user account named "su_perlovl", whereas my regular account is "perlovl". I use su_perlovl rarely to insure I am following the development process, as intended.

After about a week our family would not start

About a week after we created our services family, the family daemons crashed and would not restart. One of the team members used an ObjectStore (the database used by TeamConnection) tool to verify the integrity of the database, and the tool reported problems. He was afraid we had lost all of our work. However, after I rebooted the Windows NT server everything worked fine.

Solution

- Reboot regularly

  An operating system like Windows NT needs to be rebooted regularly. Otherwise, there is a degradation of performance and unexpected problems occur.

- Perform regular backups

  If the database is corrupted by keeping TeamConnection running while the server is experiencing severe problems, it may be necessary to restore from a backup. We try to copy the family database to another file every night as an incremental backup, and copy the entire directory structure to another directory weekly as a full backup.

- Perform vital records backups

  What we call vital records backup is merely copying the files you have backed up to another location. Choose somewhere far enough from your current server so that they cannot both be damaged by the same event, such as a fire in your building. One of the strengths of TeamConnection is that you can restore a backup onto any server platform (i.e. AIX, HP-UX, Windows NT and OS/2), regardless of the current server operating system. We copy our weekly backup to tape and have that tape sent to another physical location weekly.

- Do not use a Windows NT Server as a desktop system when it is performing as a server

  Windows NT, while a multitasking operating system, can easily be slowed down by an application running on the desktop. Also, these applications can hasten the need to reboot. As a result, desktop applications tend to severely impact TeamConnection family server performance. We only use our Windows NT server system to run TeamConnection daemons and perform family administration activities.

Some other problems and inconsistencies

Here are some other, smaller errors and difficulties we ran into during the early stages of our family's evolution:

- Using `tcadmin` to create our family, we specified `e:\dept_cfj` for the family directory. The family created was:

  ```
  e:\dept_cfj\dept_cfj\dept_cfj.tcd
  ```

  This created an extra dept_cfj subdirectory.

- Everyone worked in one release and no one performed a workarea freeze. As a result, only the most current version of each part was saved. This is an easy mistake to make, since there is nothing to force you to create new versions.

- Windows NT Workstation does not have the TCP/IP tools of NT Server. This would have helped by providing remote access.

Some good ideas

Not everything we did was bad. We really liked these decisions:

- We set the release and component processes to `prototype`.

  In other words, all we needed to do was create a workarea and we were up and running. This is easy to learn and to use, reducing stress and effort for new users.

- We limited our file names to 8.3

  We voluntarily limited our filenames to a filename of 8 characters and a file type of 3 characters. For example, we defined our "readme" as README.txt. This will extract the same on all PC and Unix filesystems, including DOS FAT.

- We created an "obsolete" component for deleted components

  A component named "obsolete" is a good parent for other components that are no longer used.

Mature system administration

Gradually your team will become accustomed to TeamConnection and the data in the family will become critical to your group's success. At this point, you should have done all of the things I just listed to ensure you have mature system and family administration practices. Again, your family is only as reliable as the system it runs on, and the processes you have in place. Since we made the changes I have listed, we have not had a failure or lost any data.

Enhancing the development process

As we moved closer to delivering a product using TeamConnection we wanted to reduce the risk of developers causing code to be lost, builds to fail, etc.. At this point, we needed to balance our desire not to be encumbered by process and the need for our team to insure that our changes were brought together without mistakes. Here are some elements I added to our TeamConnection processes in our services family to help insure that no data was lost and our product was built correctly.

Adding to the process

As soon as you have enough data in a TeamConnection release, you will want to freeze that information by committing it into the release. Also, you will want to incrementally add to that release in a controlled manner and keep a record of why you added your changes. Drivers are used as collectors of workareas and allow you to commit a group of related changes at one time and in an orderly manner. Defects and features identify why you are making the changes in your workareas.

- Add drivers to ensure clearly identified deliverables
- Add defects/features to create history for changes
- Update access lists to control who updates parts

Other changes

Here are some minor changes we made as we enhanced our process

- Linking shared parts across releases

  As I already mentioned, some of our parts are common across releases. I identified those parts and added them to the other releases using `part -link`. Now, whenever this part changes we are reminded to specify a workarea for all releases that are common for that part.

- Use path information for parts

  Our parts were created without path information. This is fine for a small release, where all of the parts are related, but it can be confusing when the parts extracted into a directory do not have a relationship, or there is more than one file in the release with the same name, such as README. In those cases, you need to add path information to make the parts manageable when they are extracted.

- Autopruning saves space, but loses change history

  We had turned on autopruning for one of our releases to see if we liked it or not. Since we were doing all of our work in one workarea, as soon as I committed that workarea our entire change history was pruned!

  We decided to turn autopruning off until we have a much more established baseline and the changes in each workarea become smaller.

Taking full advantage of TeamConnection

You aren't taking full advantage of TeamConnection until you do a few things to ensure your product builds reliably and that you can track the results of your build.

Use TeamConnection builders

A fully automated build insures that only what is in the TeamConnection family is built into your product. Using TeamConnection to build your process removes manual intervention from one of the most common problem areas in development, building exactly what you have stored in your library.

Since our family produces class materials, the built product is a zip file containing those materials. So I added a `zipbuild` builder. This builder is:

- Written in DOS batch, so it will run in Windows and OS/2
- Imbeds the TeamConnection context Family, Release, Workarea in the Zip file using TeamConnection environment variables
- Uses the `(@)#` symbol so that `what` and `tcwhat` can identify the TeamConnection context
- Displays lots of useful information that will show up in build messages

Look at the examples in “Zipbuild builder for OS/2 and Windows,” “TeamConnection build messages for zipbuild” on page 12, “TeamConnection keywords in a text file” on page 14 and “Using tcwhat to show TeamConnection keywords” on page 14 for how we use a builder and keywords.

Note: In the following examples, lines were continued using backslashes (\) in order to accommodate formatting limitations.

Zipbuild builder for OS/2 and Windows

Of course, I needed to start a build processor and build agent in order to do the builds. The easiest place to do that for my family was the TeamConnection family server machine. In most cases, it will be your development machine.

```
@echo off
echo TeamConnection zip file builder
echo -TC_FAMILY:     %TC_FAMILY%
echo -TC_RELEASE:    %TC_RELEASE%
echo -TC_LOCATION:   %TC_LOCATION%
echo -TC_INPUT:      %TC_INPUT%
echo -TC_INPUTTYPE:  %TC_INPUTTYPE%
echo -TC_OUTPUT:     %TC_OUTPUT%
echo -TC_OUTPUTTYPE: %TC_OUTPUTTYPE%
echo -TC_WORKAREA:   %TC_WORKAREA%
echo Build Start
echo @(#) Package context: Family %TC_FAMILY%, Release %TC_RELEASE%, \
   Workarea %TC_WORKAREA% > tmpfile
zip -z %TC_OUTPUT% %TC_INPUT% < tmpfile
erase tmpfile
echo --------------------
dir %TC_OUTPUT%
echo Contents of zip file
```

```
unzip -l %TC_OUTPUT%
echo Processing complete
```

TeamConnection build messages for zipbuild

```
There is not a parser associated with this
part object.
Messages from the builder:
6021-403 A successful build resulted from using the builder zipbuild. The
         builder return code is 0

******************Builder Listing Follows************************


The following line is the command which was passed to the system:
#####################################################################
'zipbuild.cmd '
#####################################################################

The following is the stdout and stderr output from the command:
---------------------------------------------------------------------
TeamConnection zip file builder
-TC_FAMILY:     dept_cfj
-TC_RELEASE:    class1_admin
-TC_LOCATION:   E:\DEPT_CFJ\DEPT_CFJ\BIN\FHBBUILD
-TC_INPUT:      day1_1.prz day2_2a.prz day3_1a.prz day1_2.prz \
                trtcoem.ps day3_3.sam readme.pubs doscmd.exe \
                trtcoem.txt README.txt doscmd.c day3_4.prz \
                trtcvscm.txt class1_3.sam class.cmd class2_2a.sam \
                trtcvscm.ps class2_3.sam day1_3.prz day1_1c.prz \
                day2_3.prz
-TC_INPUTTYPE:  TCPart TCPart TCPart TCPart TCPart TCPart \
                TCPart TCPart TCPart TCPart TCPart TCPart TCPart \
                TCPart TCPart TCPart TCPart TCPart TCPart TCPart \
                TCPart
-TC_OUTPUT:     admclass.zip
-TC_OUTPUTTYPE: TCPart
-TC_WORKAREA:   1
Build Start
  adding: day1_1.prz (deflated 66%)
  adding: day2_2a.prz (deflated 58%)
  adding: day3_1a.prz (deflated 66%)
  adding: day1_2.prz (deflated 76%)
  adding: trtcoem.ps (deflated 87%)
  adding: day3_3.sam (deflated 83%)
  adding: readme.pubs (deflated 62%)
  adding: doscmd.exe (deflated 67%)
  adding: trtcoem.txt (deflated 78%)
  adding: README.txt (deflated 61%)
  adding: doscmd.c (deflated 71%)
  adding: day3_4.prz (deflated 68%)
  adding: trtcvscm.txt (deflated 74%)
  adding: class1_3.sam (deflated 85%)
  adding: class.cmd (deflated 22%)
  adding: class2_2a.sam (deflated 73%)
  adding: trtcvscm.ps (deflated 83%)
  adding: class2_3.sam (deflated 84%)
  adding: day1_3.prz (deflated 50%)
```

```
  adding: day1_1c.prz (deflated 63%)
  adding: day2_3.prz (deflated 46%)

enter new zip file comment (end with .):

--------------------

 Volume in drive E has no label.
 Volume Serial Number is 7032-D7B7

 Directory of E:\DEPT_CFJ\DEPT_CFJ\BIN\FHBBUILD

10/26/97  10:19p             1,073,557 admclass.zip
               1 File(s)      1,073,557 bytes
                             593,816,576 bytes free
Contents of zip file
Archive:  admclass.zip
@(#) Package context: Family dept_cfj, Release class1_admin, Workarea 1
 Length    Date    Time    Name
 ------    ----    ----    ----
 655548  10-25-97  05:56   day1_1.prz
 245466  10-25-97  05:56   day2_2a.prz
  83808  10-25-97  05:56   day3_1a.prz
 115317  10-25-97  05:56   day1_2.prz
 296373  10-25-97  05:56   trtcoem.ps
  17779  10-25-97  05:56   day3_3.sam
   4821  10-25-97  05:56   readme.pubs
  89600  10-25-97  05:56   doscmd.exe
  82607  10-25-97  05:56   trtcoem.txt
   4248  10-26-97  22:19   README.txt
   3431  10-25-97  05:56   doscmd.c
  59878  10-25-97  05:56   day3_4.prz
 137926  10-25-97  05:50   trtcvscm.txt
  89690  10-25-97  05:56   class1_3.sam
     51  10-25-97  05:56   class.cmd
  53258  10-25-97  05:56   class2_2a.sam
 361377  10-25-97  05:56   trtcvscm.ps
  60161  10-25-97  05:56   class2_3.sam
 200229  10-25-97  05:56   day1_3.prz
 619122  10-25-97  05:56   day1_1c.prz
 215194  10-25-97  05:56   day2_3.prz
 ------                    -------
3395884                    21 files
Processing complete
----------------------------------------------------------------------

**********End Of Builder Listing**********
End of build report for: 'admclass.zip'
```

Use what strings in text and output files

Since my class materials were all binary files, I could only use TeamConnection keywords in my README file.

TeamConnection keywords in a text file

Here are the first few lines in that README file:

```
README for Education class materials:
Keywords:
$KW=@(#); $ChkD=1997/10/26 22:17:55; $FN=README.txt; $Own=jhook; \
$Ver=10-25-97a:2;
$EKW;
...
```

Using tcwhat to show TeamConnection keywords

```
C:\education>tcwhat admclass.zip
admclass.zip:
         Package context: Family dept_cfj, Release class1_admin, \
                          Workarea admin_copyright
```

Consolidating releases

Initially, we created separate releases for the Adminstrator's and Developer's classes. It made extraction of the parts belonged with each class easier. Once we added output parts, we were able to extract the built output files instead of extracting all of the parts in each release. At this point, there is no reason for separate releases. A new, combined, release was created, one of the releases was linked into the new release, then the unique parts in the second release were linked in.

The biggest advantage of using only one release is that only one workarea needs to be created for each defect or feature, and there is no need to specify "common releases" when checking in parts. This eliminates the risk of accidentally breaking links, resulting in out of date parts in some releases.

Automation and future enhancements

The evolution of a family is never really complete. It is easy to think of little (or big) changes that might make your development activities run a little smoother. Also, it is occasionally useful to clean house by removing processes, components, and other configuration options that aren't contributing to your productivity any longer.

After making the necessary process changes to ensure that we could deliver our class materials reliably, we considered what would make our job easier and more reliable. Here are some enhancements we are considering for our services family in the future:

Automated shutdown and backup

The automation of startup and shutdown, including a reboot, is useful. However, for my family I rely on a wide range of IBM internal use only tools for automation. Here is what you will want to do:

1. shutdown family
2. shutdown database
3. copy .tcd file(s)
4. send to another system (vital records backup)
5. reboot

Here is what we currently use for shutdown and backup. We need to schedule an NT "at" job (an "at" job is a specially scheduled process that can be started at a later time) and make the TeamConnection processes into Windows NT services to complete the automation. OS/2 and Unix are already fully enabled for this sort of automation. In Windows NT, a product needs to be Microsoft Back Office compliant (i.e. processes run as "services"). You can work around this by using the Windows NT Resource Toolkit to enable the existing code as a service.

Notes:

1. Some of the tools in the example are not generally available.
2. In the following examples, lines were continued using backslashes (\) in order to accommodate formatting limitations.

Simple shutdown script

```
@echo off
REM Stop TeamConnection
e:
cd \dept_cfj\dept_cfj
erase kill.bak
ren kill.out kill.bak
REM Kill TeamConnection daemons
REM Redirection may cause errors
kill -f notifyd
kill -f teamcd
kill -f teamagnt
kill -f teamproc
rem manually stop ObjectStore to get messages
net stop "ObjectStore Server R4.0" > kill.out 2>&1
```

```
net stop "ObjectStore Cache Manager R4.0" >> kill.out 2>&1
REM Copy database
erase dept_cfj.bak
copy dept_cfj.tcd dept_cfj.bak
reboot
```

Automated Startup

Here is our current start up script. It runs after Windows NT has rebooted and a user logs in. Since ObjectStore automatically starts as a Windows NT Service, the script starts the TeamConnection processes.

Startup script for Administrator login

```
@echo off
REM Start TeamConnection Daemons
e:
cd \dept_cfj\dept_cfj
erase starttcd.bak
ren starttcd.out starttcd.bak
rem Syntax: teamcd ... & stayed at teamcd (not background)
start teamcd dept_cfj@9000 3 > starttcd.out 2>&1
erase startntf.bak
ren startntf.out startntf.bak
start notifyd dept_cfj mailexit > startntf.out 2>&1 &
erase startbta.bak
ren startbta.out startbta.bak
start teamagnt -f dept_cfj -p bldsock -e NT -s @mal4@9002 \
> startbta.out 2>&1 &
erase startbtp.bak
ren startbtp.out startbtp.bak
start teamproc -s @9002 > startbtp.out 2>&1 &
echo all processes started
```

This tool is fairly rudimentary. Here are some improvements we look forward to implementing in the near future:

1. Automated startup of TeamConnection daemons (as a Windows NT service)
2. Record startup results better
3. FTP the backup to another system for vital records backup

Extractor Access Role

A common role within a development group is a member who doesn't do active development, only opening of defects/features and adding periodic remarks, as well as viewing and extracting of parts. We call this person an `Extractor`. All that needs to be done is to copy the acitons of `general` and add a few part actions.

New User exits

In a small family, user exits are nice to have, but not essential. There really isn't enough activity to require them. Then again, anything to save steps is helpful. Here are some user exits that could help this family run a little smoother.

New user notice

The welcome text that is mailed to a new user when their user account is created includes the following:

- Family name, server host name and port number

  ```
  dept_cfj@mal4.raleigh.ibm.com@9000
  ```

- Builder name and port

  Server mal4.raleigh.ibm.com
  Pool bldsock@9006
  Environment NT

- Access methods

  ```
  PASSWORD_OR_HOST
  ```

Other user exits we are considering

- Workarea automation

  Automatically add workareas to integrate into a driver.

- Driver Create automation

  Automatically create a new driver when a driver is committed, and allow only one driver in working state for a release.

- Driver Commit automation

  Automatically complete committed drivers at a specified time. It is good to update the driver type at that time.

- Shadowing parts to a reference directory

  Automatically copying updates to parts in a release, driver or workarea to a read-only directory structure that is available to every user so that they can use tools to analyze the contents of the parts.

Open issues

There are still some things we haven't decided how to handle with our service family and we are fully productive. So take heart, you aren't alone. Here is what we are still pondering:

- What should our driver naming convention be and how should we open our drivers?

  In some of our families, drivers are automatically opened as soon as a driver is completed within a release, and the name is derived from the date the driver was opened. Somehow, this doesn't seem appropriate for us. We don't know when we will commit each driver, so we can't pick an appropriate date. Further, the date the driver was opened doesn't seem to make any sense. The good news is that it is easy to rename a driver, so we aren't worrying about it.

- Port number too common

  The currently used port number is 9000. This is a very common number. We will probably change the number.

TeamConnection is still growing

TeamConnection is still growing and evolving. We always keep our eyes open for new features in TeamConnection that can help us do our jobs better.

For example, we just replaced the notifyd `mailexit` sample with a new sample that forwards mail to Lotus Notes.

Conclusion

A family with the appropriate processes and a smooth running system with sound system administration practices allows TeamConnection to improve the productivity of your development team.

I hope it is encouraging to know that even the TeamConnection development and services teams need to spend a little time deciding what TeamConnection processes and configuration options are most beneficial to their development needs.

The recommendations in this document helped the productivity of my department's development efforts, and we hope it will help yours too.

Appendix A. Bibliography

TeamConnection Publications

For more information on how to use TeamConnection, you can consult the following manuals:

SC34-4551 TeamConnection, Administrator's Guide
SC34-4552 Getting Started with the TeamConnection Clients
SC34-4499 TeamConnection, User's Guide
SC34-4501 TeamConnection, Commands Reference
SC34-4500 TeamConnection, Quick Commands Reference

Related Redbooks

The following IBM redbooks provide practical advice about TeamConnection from software specialists:

SG24-4648 Introduction to the IBM Application Development Team Suite
SG26-2008 TeamConneciton Family and Application Develoment
SG24-4610 TeamConnection Workframe Integration Survival Guide

Related Technical Reports

The following technical reports provide hints for using TeamConnection:

29.2267 TeamConnection frequently asked questions: How to do routine operating system tasks
29.2253 Comparison between CMVC 2.3 and TeamConnection 2

Appendix B. Copyrights, Trademarks and Service marks

The following terms used in this technical report, are trademarks or service marks of the indicated companies:

TRADEMARK, REGISTERED TRADEMARK OR SERVICE MARK	COMPANY
AIX, OS/2, IBM, CMVC, TeamConnection	IBM Corporation
ObjectStore	Object Design, Inc.
PKZip	PKWare, Inc.
UNIX, USL	UNIX System Laboratories, Inc.
Microsoft, Windows Back Office, Windows NT	Microsoft Corporation
X Window System	Massachusetts Institute of Technology

END OF DOCUMENT °