Building and packaging the CMVC Server/Client for UNIX

Document Number TR 29.3408

Angel Rivera

CMVC Customer Support
IBM Software Solutions
Research Triangle Park, North Carolina, USA

DISCLAIMER:

This technical report is not an official publication from the CMVC group. The author is solely responsible for its contents.

ABSTRACT

The purpose of this technical report is to provide a real example of the use of CMVC for developing, maintaining, building and packaging the CMVC Server and Client (line commands and GUI) for UNIX, which encompasses several operating systems and database management systems. This example illustrates the complexity associated with the development of applications in heterogeneous environments.

ITIRC KEYWORDS

- CMVC
- UNIX
- Building
- Packaging

ABOUT THE AUTHOR

ANGEL RIVERA

Mr. Rivera is an Advisory Software Engineer and team lead for the CMVC Direct Customer Support team. He joined IBM in 1989 and since then has worked in the development and support of library systems.

Mr. Rivera has an M.S. in Electrical Engineering from The University of Texas at Austin, and B.S. in Electronic Systems Engineering from the Instituto Tecnológico y de Estudios Superiores de Monterrey, México.

CONTENTS

FIGURES

BUILDING AND PACKAGING CMVC FOR UNIX

The purpose of this technical report is to provide a real example of the use of CMVC for developing, maintaining, building and packaging the CMVC Server and Client (line commands and GUI) for UNIX, which encompasses several operating systems and database management systems. This example illustrates the complexity associated with the development of applications in heterogeneous environments.

This document describes the challenges encountered when developing in heterogeneous environments and how we cope with them. It also describes the development process that we used with CMVC for the development of the application.

Then, in order to give you the extent of the complexity associated with the development of an application in heterogeneous environments, the rest of this document is dedicated to the prerequisites, configuration, building and packaging the CMVC Client and Server for UNIX.

CMVC for UNIX is made available for the following operating systems. This is the list of the latest versions that are supported at the time this document was written:

- AIX Version 4
- HP-UX Version 10.20
- Solaris 2.4 and 2.5.1

The CMVC Server for UNIX is built using the following relational databases (although we do not provide all possible combinations of operating system and database).

- DB2 UDB Version 5.2
- Oracle Version 7.3.4
- Informix Version 7.3
- Sybase 11.9.2

  Although we do not do a native build on Sybase 11, the build on Sybase 4.9, works fine on Sybase 11.

A set of quick status checklists with a summary of the build and package activities for the Client/Server code and the GUI code are provided in the appendices.

DISCLAIMER

To avoid misunderstandings with the purpose of this technical report and to better understand its scope, the following disclaimers are in order:

- This technical report is not an official publication from the CMVC group. The author is solely responsible for its contents.
- The intention in writing this detailed process is certainly NOT to dictate to you the ONLY way to build the code; the intention is to document the process that we have used and that has proven to be successful.
- This process is NOT perfect and there is room for improvement, and you are welcome to improve it and to provide feedback to update this document to reflect that improvement.
- This technical report was prepared when working with CMVC 2.3.1.4 in the operating systems and database versions described in this document.

  Therefore, if you have a different version of the mentioned software, then you may expect some differences in the information or in the procedures described in this technical report.
- This technical report covers information that the author has gathered thru the years while working with the CMVC technical support team.
- It is the intention of this technical report to provide recommendations and guidelines that can be helpful to CMVC administrators and project leads. In some cases, the procedures will not be exhaustive, and will just show the overall sequence that has worked before, which might be different in your case.
- Real values that were used in our setup will be used in this technical report. Thus, you will need to customize the commands that you issue to reflect the values that are meaningful to your setup.
- It is assumed that the reader has knowledge of CMVC, the appropriate database management system (DBMS) and the appropriate operating system.

  This technical report is not a substitute to the information provided by CMVC and the appropriate DBMS and operating system. Please refer to the appropriate documentation provided with the corresponding software.

CHALLENGES RELATED WITH HETEROGENEOUS ENVIRONMENTS

There are several challenges that we faced during the development of CMVC for UNIX in a heterogeneous environment:

- The main challenge is to maintain all needed combinations of hardware and software.
- The next challenge was to seamlessly work across the different platforms. To overcome this challenge we used the same shared userid in all systems called "build".
- Then, we used the message-of-the-day (motd) technique to communicate between the developers and reserve working areas.

  This technique consists of a file named $HOME/motd which details the comments from developers. This file is shown automatically at login time and there is a shell script named "motd" which invokes the "vi" editor with this file to allow the developers to update this file.

MAINTAINING ALL NEEDED COMBINATIONS OF HARDWARE AND SOFTWARE

The main challenge is to maintain all needed combinations of hardware and software.

The concurrent development of the CMVC Server for UNIX is done across multiple platforms/versions, multiple hosts and multiple relational database management systems.

Furthermore, the development of the CMVC Client was done for standalone mode or integrated with Softbench, plus in AIX and HP-UX we had to provide two versions, one for Motif 1.1/X11R4 and another for Motif 1.2/X11R5. Recently, we provide only the standalone version on Motif 1.2/X11R5. That is, we are no longer providing the integrated version of the GUI (thus, please ignore the code that is under the "intg" subdirectories, or the mode "intg").

The tasks of obtaining the hardware and software, renewing the contracts, requesting and installing patches, providing maintenance (preventive and corrective) constitute a major portion of the development activities.

ROLE OF 'BUILD' USERID AND EXPORTED DIRECTORY

All building activities in all operating systems is performed from the userid 'build' which has to have the same main characteristics in all the platforms: same user id number, same group id number and same home directory.

```
uid=210(build)
gid=222(syscmvca)
groups=0(system),1(staff),2(bin),3(sys),222(syscmvca)
HOME=/home/build

NOTES:
* The groups should be appropriate to the platform
* The group system is required because it builds a setuid-root
  application (cmvcd, the CMVC daemon).
```

The actual home directory for this userid resides in an exported file system from a file server (in our case in host 'tcaix05') which must have enough capacity to hold the builds, install images and miscellaneous for all the needed combinations (in our case 4.0 GB).

This exported file system is mounted in all other platforms, which have the same userid, groupid and home directory:

```
mount tcaix05:/home/build   /home/build
```

The advantages of this scheme are:

- This approach allows for maximum reuse and sharing of common files such as .profile and directories such as $HOME/bin.
- It is possible to work simultaneously in the same 'build' account from different operating systems. For example, one developer could be doing a build for AIX 4 and DB2 UDB 5 meanwhile another developer could be using HP-UX 10 and Oracle 7.
- It is possible to perform CMVC release and level extracts and use the same host and specify only the target directory.

  Furthermore, it is possible to specify the "nomount" option when extracting a release/level in order to not use NFS. This local extract is faster than an extract that involves NFS and avoids the risks mentioned in “Risks of building code across NFS mounts” on page 5.
- The number of host list entries in CMVC is reduced because the system login id is always "build". We need to create only those host list entries for our CMVC user ids and hostnames.

  One important consideration when using the CMVC GUI in such environment is that there is ONLY one "Cmvc" resource file and one CMVC tasks list. Furthermore, the developers need to be aware of the CMVC user id when using the GUI, because it will reflect the name of the previous developer.

Risks of building code across NFS mounts

If your NFS setup is not too reliable, then you may have problems during the extraction of large releases or levels. One alternative is to consider doing a local extract by using the "nomount" option; also, you can consider using AFS or DFS.

New versions of NFS now support TCP instead of UDP to verify that each packet that is sent is actually received. When using NFS with UDP you should test the executables and search for compiled/linked objects of size 0 (zero):

```
find .
```

```
#!/usr/bin/ksh
#-------------------------------------------------------------------------+
# NAME: get_opsys
#
# Return a string identifying the operating system.
# Output:  OPSYS = (AIX, HP-UX, SunOS, Solaris)
#
# Notes about uname:
#  * The command "uname -a" returns the following information:
#
#      operating_system_name   node_name   release_name   version_name
#
# * Examples of uname:
#
# AIX 4:       AIX oem-ppc3 1 4 000022559000
# HP-UX 10:    HP-UX oem-hp01 B.10.01 A 9000/735 2010377972 two-user license
# Solaris 2.4: SunOS oem-sn01 5.4 generic sun4m sparc
#
#-------------------------------------------------------------------------+
get_opsys()
{
raw= uname -a
OPSYS= echo $raw | cut -f1 -d' '

# Test for Solaris (5... or higher) or SunOS (4...)
# The problem is that the Solaris op-sys is identified as SunOS, and
# there is no string that indicates that is "Solaris". The distinction
# is done only by the version number.

if ﬀ "$OPSYS" = "SunOS" ❝
then
   var= echo $raw | cut -f3 -d' ' | cut -c1-1
   if ﬀ $var -ne 4 ❝
     then
       OPSYS="Solaris"
   fi
fi
export OPSYS
echo "get_opsys: " $OPSYS
}
get_opsys

# end of file
```

Figure 1. get_opsys(): Shell script to find out name of operating system

Shared .profile

The shared .profile has to be comprehensive enough to accommodate for the differences between platforms and databases. For example, the location of one utility may be /bin in one system or /usr/bin in another.

We use the name of the host inside a case statement in order to provide environment variables that deal with these differences.

USING A BULLETIN TO COMMUNICATE NEWS (MESSAGE-OF-THE-DAY)

Because the same build directory is used across the systems, it is important to have a "bulletin" for messages to avoid collisions with the usage of subdirectories.

It is recommended to use the file $HOME/motd (for "Message Of The Day") to place a note that would be of importance to the other users of the build directory. This file is displayed immediately after the logon and before the first prompt.

An example is shown below:

```
System/DB:        Orac7.3 Inf7 DB2v5 Syb11
 oem-ppc3 AIX4       X     X     X
 oem-sn13 solaris    X                 X
 oem-hp03 V10        X     X

Release: cs231
        cmvc231/aix4/cs/informix7   - L 002, Migration - Angel
        cmvc231/hp10/cs/informix7   - L 002, bug X - JRY (oem-hp03)
Release: gui231
        cmvc231/solaris/gui/x11r5/stal  - PMRs - Angel
```

The shell script used for displaying the message of the day is shown below:

```
#!/usr/bin/ksh
#
# Edit and update the message-of-the-day (motd)

# Take last line of file and save in temporary file
outfile="/tmp/motd.build"
let x= cat $HOME/motd | wc -l | tr -d ' '
let y=$x-1
head -$y $HOME/motd > $outfile
echo Entering vi to edit current motd
echo Note: do not enter date, it will be appended automatically
sleep 2

# Edit file and append time/date stamp
chmod u+w $outfile
vi $outfile
echo Exiting motd, date is being appended
echo "***End $HOME/motd as of  date . 'motd' to update***" >> $outfile
mv $HOME/motd $HOME/motd.old

# Save old file and install new one
cp $outfile $HOME/motd
if ffl -f $outfile “
then
   rm $outfile
else
   echo "Error, new motd not copied.  Saving $outfile"
fi

# Show results
clear
echo "Displaying new motd:"
sleep 2
cat $HOME/motd
exit 0

# end of file
```

Figure 2. motd: Shell script to edit and view the message-of-the-day

OVERALL PROCESS WITH CMVC

CMVC is used to manage the software configuration and the file changes for the CMVC Server and Client for UNIX. This section describes the overall process for handling the files and the different CMVC objects associated with the CMVC Server and Client for UNIX.

FAMILIES, COMPONENTS AND RELEASES

The CMVC 2.3.1 product is developed and maintained in the "toro" family. The components and releases are:

- The main component for the server and the line commands in the toro family is cmvc and the release is cs231 .
- The component for the tools to install the code in the toro family is called cmvcInstallp and the release is inst231 .
- The component for the GUI in the toro family is called cmvcgui and the release is gui231 .
- The parent component for the obsolete components and releases is called "OBSOLETE". In this way, these obsolete objects do not clutter the tree view of the active components.

The components have the default process which include the Verify subprocess for Defects and Features and the Design, Size and Review subprocess for Features (and not for Defects).

The releases have the Fix, Track and Level subprocesses during all the development phases.

The Approval subprocess for the releases is activated during the times when the code is about to be prepared for verification and for packaging; in that way, the changes to the release are carefully monitored and controlled.

Instead of three separate releases, it might be possible to use only one release for all the code (line commands, GUI, installation tools), which could minimize the overhead associated with tracks and levels. However, we found that having three releases gives us also much needed flexibility for handling situations where the development activities concentrated on the server code and there were frequent code changes, and thus, we could continue at slower pace of change with the GUI.

In each refresh of the CMVC code to customers, we provide a full replacement. We identify these refreshes by using the CMVC Level as the 4th identifier from left to right; for example, our latest refresh for the code was Level 4 or refresh "2.3.1.4".

PROCESS MODEL

This section describes the development process followed by the CMVC development team to handle defects and features.

There are three important characteristics in this process:

- There is only one active level at a time.

  This avoids the problem of having prerequisite tracks that are in another active level (spaghetti tracks).

- The criteria to commit a level is that the code can be built successfully, regardless if the function is working as desired. This guarantees that any level can be extracted and built correctly.

  If the level builds fine, but if there are functional problems, then new defects are created and handled in the next level.

  We try to avoid a large number of file changes and tracks in a level.

  We would rather have many small levels than a single extremely large level. There are a couple of reason for this decision:

  – The Level -check and Level -commit are by far the most expensive operations in CMVC, in terms of CPU usage and in exploitation of the database.

  – If you wait a long time before committing a level, then the users who created defects or features will not see any verification records and the testers of your software may not see a driver built in a long time.

While reading this document, it might be helpful to have a copy of the Appendix A "CMVC State Diagram", page 72 from the CMVC Concepts manual. In this way you will be able to follow the state transitions and the relationships between the elements.

PROJECT LEAD CREATES COMPONENTS AND RELEASES

1. The project lead creates the necessary components under the family.
2. These components use the Design-Size-Review (DSR) subprocess for features, but the DSR subprocess will not be used for defects. The verify subprocess will be used for both features and defects.
3. The project lead creates under the appropriate component the relevant releases.

   Each release should use the subprocesses fix, track, test and level. Depending on the stage of the release, it might be necessary to add the approval subprocess.

Originally we used the test subprocess when the team was larger and we had one tester fully dedicated to CMVC, but now that the team is smaller, we found that the verification records provide us with the opportunity to ensure that the fix indeed addressed the problem. Thus, we do not use the test subprocess anymore.

HANDLING DEFECTS AND FEATURES

1. The project lead or a team lead will assign to the developers the appropriate defects and features to be implemented.
2. The developers accept the defects and the defects move to the 'working' state.
3. The features are designed, sized, reviewed, and accepted, until eventually they will be in the 'working' state.
4. A track is created to associate each defect or feature with the appropriate release or releases.

DEVELOPERS WORK WITH FILES USING TRACKS (CODE AND UNIT TESTING)

This is the phase for Coding and Unit Testing.

1. The developers will create/check-out, compile, test, debug and check-in the files affected by the release.

   Because the release uses the track subprocess, during check-in, the developers need to specify the appropriate track.

   The existing files that are not changed as a result of the feature (but need to be included during the compilation), are extracted.

   We recommend that the user performs a release/level extract of the committed version.
2. All the components that have file changes will have a a fix record for a track.

   You can see the fix record by selecting the "Fix Records..." option from the "Windows" pulldown in the "CMVC - Tracks" window.
3. You can see the file changes in a track by selecting the "Change History" option from the "Show" pulldown in the "CMVC - Tracks" window.

PREPARING THE CODE FOR FUNCTION VERIFICATION TEST (FVT)

Once the developers decide that it is time to integrate the code and go to the Function Verification Test (FVT) phase, then the following actions will take place:

The project leader will confer with the developers to make sure that all the files that will be in FVT must have completed the unit testing.

1. The project leader will create a level for the release.

   The initial state of the level is 'working'.

2. The fix records for each relevant track need to be completed.

   Ideally, the component owner for the fix record should review the code that was changed, and then mark the fix record as completed. This action will certify that the code compiles at the individual level and that unit testing has been performed.

   However, in practice, this step is done only on difficult file changes where a second opinion is advisable.

3. When all the fix records are marked as completed, then the track is moved from the 'fix' state to the 'integrate' state.

4. For each track, a level member has to be created for the level. The level member window can be shown by selecting the "Level Members" option from the "Windows" pulldown in the "CMVC - Levels" window.

   This action will move the level to the 'integrate' state.

   This action is done by the builder, who is the person in charge of adding the tracks to the level and then extracting the level in order to compile the code.

5. Then a build is done of the level in order to produce the testing driver for the release.

   For more information, see the process document for the build activity.

6. If a problem is found during the compilation/linking of the code from the level and if a change has to be done in a file, then it is necessary to move the track to the "fix" state; in order to do that it is necessary to do the steps described in “Backing out a faulty FVT driver” on page 13.

7. Before extracting the level, it is necessary to check it in order to ensure that all file changes are present, and that there are no missing prerequisites.

   If there are missing prerequisites, then the missing tracks should be added or the appropriate tracks present in the level should be removed (in case the prerequisite track is not ready to be integrated).

8. If starting a build from scratch, then the builder will perform a full extract of the latest committed level. Then, the builder will extract the Delta of the current integrated Level.

If not starting from scratch, then a Delta of the current Level might be sufficient. In case of doubt, delete the current build tree and do a full level extract plus a delta extract.

9. If no problems were found during the build activity, then the level is committed and completed.
10. We may change the value for the level field "type" to indicate how the code will be distributed:
    - Gold, if the code is going to be part of a standalone CD-ROM or participate in the IBM Software Solutions for AIX Showcase CD-ROM.
    - Refresh, if the code is just going to be placed in the internal and external FTP sites for users to download.
    - Beta, if the code is going to be delivered to few customers as beta code.

Backing out a faulty FVT driver

If during the preparation of an FVT driver, a problem is found, then it could be necessary to back-out the offending track, fix the files, and repeat the process. This 'backing out' action can be accomplished by following the steps below:

1. Remove the level member associated with the track that contains the file or files to be modified.

   This can be done by selecting the "Remove" option from the "Actions" pulldown in the "CMVC - Level Members" window.

2. Move the track from the "integrate" state to the "fix" state, so that the files in question can be modified.

   This can be done by selecting the "Fix" option from the "Actions" pulldown in the "CMVC - Tracks" window.

3. Activate the fix records for the track.

   This can be done by selecting the "Activate" option from the "Actions" pulldown in the "CMVC - Fix Records" window.

4. Check-out, modify the contents of the files, compile, test, and check-in the files again.
5. Complete the fix records; this action will move the track to the integrate state.
6. Go back to step 5 on page 12.

DEALING WITH TRACKS/FEATURES DURING FVT

1. Once the level moves to the "complete" state, the tracks move to the "test" state.

   In case the release does not have an environment list, then the track moves to the "complete" state.

2. Once the track is in the "test" state, then the FVT tester will test the code and will have the opportunity to accept, abstain, or reject the track.

   Originally we had a full-time tester and thus we needed the test subprocess, but now our team is smaller and we do not use the test subprocess, instead we use the verification records.

3. If no problems are encountered that are related to the track/feature at hand, then the tester will "accept" the test record for the track.

   To access the test record, select the "Test Records..." option from the "Windows" pulldown in the "CMVC - Tracks" window.

4. If the tester finds a problem, then the tester will "reject" the test record.

   Then the tester should open a defect reporting the problem. See “Dealing with defects created during FVT” on page 15.

5. Once the test records for a track have been accepted/rejected/abstained, the track moves to the "complete" state.

   This in turn moves the defect or feature to the "verify" state.

6. The originator of the defect or feature is notified that the defect or feature is now in the verify state, and then the verification record for the defect or feature needs to be accepted, rejected, or abstained.

   When the verification records are modified, the defect or feature moves into the "complete" state.

7. If the verification record is rejected, the originator MUST create a new defect or feature in order to correct the problems.

   Note: After a verification record is rejected or abstained, CMVC will NOT automatically create a new defect or feature.

DEALING WITH DEFECTS CREATED DURING FVT

1. The testers will create defects for those problems encountered during FVT.

   The component owners will be notified by CMVC of the new defects and they will handle the defects, such as accept, reject, assign.

2. For each accepted defect, the owner of the defect needs to create a track.
3. Repeat the process of adding the track to a level.

Severities for defects

The definition of the severities that are used for the defects is shown below:

1. Function can not be used. Immediate fix is necessary.
2. Function can be used but with serve restrictions.
3. Function can be used with minor restrictions, not critical.
4. Function can be bypassed, but should be corrected.

PREPARING THE BUILDS

GENERAL ASSUMPTIONS

- For each operating system it is necessary to have a build host with the appropriate hardware and software.
- The build for each platform/database combination has to be done in each platform (no cross-compiler available).
- A single userid, build , will be used for all building activities.
- The relational database has to already be installed and a database (for example, in DB2 this is called a database instance) with the name of "build" has to be created.
- The task of building and packaging the complete product for all the environments is complex and prone to error due to the magnitude of the combinations. It is suggested to use the checklist described in Appendix D, “CMVC Client/Server UNIX, Build and Package Checklist” on page 67 to keep track of the progress for each build.

ONE-TIME ONLY ACTIVITIES

Create CMVC family 'build' to be used with DB2

When using DB2, it is necessary to create a CMVC family named 'build' on each platform in order to perform the pre-compilation steps to resolve the SQL queries with the actual database.

This CMVC family must have the following characteristics:

```
login:     build
gid:       <same as the gid for the db2instance>
```

The database name 'build' is used in the Make.<database>.<platform> make scripts.

After the database has been created:

1. Execute "db2 connect to build"
2. The results should be:

```
 Database Connection Information

Database product       = DB2/6000
SQL authorization ID   = BUILD
Local database alias   = BUILD
```

3. Execute "db2 connect reset"

Create necessary links

In order to have a common interface for the build process, in some cases it is necessary to provide symbolic soft links between the actual path for the databases/tools and the paths used inside the MAKE.cmvc and related files.

To do these links, it is necessary to execute the following command as root:

```
ln  -s  <actual-path>  <desired-path>
```

NOTE: The actual-path (source) values shown in this document are the ones that apply to our environment. The desired-path (target path) should be the ones shown below.

AIX

- Host oem-ppc3 and database Oracle 7.3.4:

  ```
  ln  -s  /disk2/u01/app/oracle/product/734 /usr/oracle73
  ```

- Host oem-ppc3 and database Informix:

  ```
  ln  -s  /disk2/informix  /usr/informix
  ```

HP-UX

- Host oem-hp03 and database Oracle 7.3.4:

  ```
  ln  -s  /cmvc_db/u2/oracle/product/7.3.4  /usr/oracle73
  ```

- Host oem-hp03 and database Informix:

  ```
  ln  -s  /users/informix  /usr/informix
  ```

Solaris

- Host oem-sn01 and database sybase49:

  ```
  mkdir /usr/db
  ln  -s  /export/home2/sybase/sybase49  /usr/db/sybase49
  ```

- Host oem-sn13 and database sybase11:

```
mkdir /usr/db
ln  -s  /export/home/sybase  /usr/db/sybase
```

- Host oem-sn12 and database Oracle 7.3.4:

```
ln  -s  /export/home2/u01/app/oracle/product/7.3.4   /usr/oracle73
```

INSTALL THE CMVC CLIENT

The CMVC client needs to be installed in each machine in order to interact with the CMVC family "toro".

To install the latest published CMVC code, using either smit-installp or tar files, get the images as follows:

1. ftp ftp.software.ibm.com
2. user anonymous
3. cd ps/products/cmvc
4. cd aix4/231 (or relevant combination)
5. dir
6. prompt
7. ascii
8. mget README* (and other text files)
9. binary
10. mget *.obj (or relevant *.tar or *.tar.Z files)
11. exit.
12. Follow the instructions from the README.inst* file.

EXTRACT THE RELEASE "TOOLS231"

Many of the tools and necessary files (such as .profile) for using the build account and to perform most of the tasks mentioned in this document are in the release "tools231".

Extract the release and specify:

```
host:             tcaix05.raleigh.ibm.com
directory:        /home/build
user id:          210
group id:         222
file permissions: 777
```

One of the files that will be extracted is .profile, which needs to be customized for the particular host that you are going to use. Specifically, you may need to add a complete subsection in the huge "case"/"switch" statement which is appropriate for the particular hostname, in order to setup the proper variables.

CREATION OF THE BUILD/SHIP/PACKAGE TREE

To create the top portion of the Build/Ship/Package tree (to be called just "Build tree") as shown in “Top directory structure - created via shell script” on page 25, do the following:

1. Login in as 'build'.
2. Create the directory that will complete the specification of TOP, such as 'cmvc231'. In this way the $TOP will be $HOME/cmvc231.
3. Change to the directory $TOP.

   The shell script will create the build tree from the current directory.
4. Execute the shell script as follows:

   ```
   mkbldtree
   ```

ESTIMATED TIME FOR BUILD AND PACKAGING

The average approximate time involved in a single complete build is shown below. However, the compilation times may range from 2 hours in fast machines such as oem-hp04 to 5 hours in slow machines.

Furthermore, this average time should be multiplied for each combination of platform and database to be built.

This time assumes that this is not the first build, and assumes that all the hardware and software is operational, and that the one-time activities have been performed already. The breakdown is shown below (in hours)

```
Extract the installation/source files:    0.5
Perform the build:                        4.0
Package the files:                        1.0
Perform the installation test:            1.0
```

Estimated time for all combinations

The approximate time involved in a complete build for all the combinations of platform/database is:

```
1 full-time week (40 hours)
```

BUILDING THE CODE

OVERVIEW OF THE WHOLE PROCESS

It takes approximately one full working week to do a complete build from scratch for a single CMVC level, such as 2.3.1.4.

The status checklist for the builds, Appendix B, “CMVC for UNIX, build and package status checklist” on page 63 is the "score card" or "cheat sheet" that was used in level 2.3.1.4 and you can appreciate the enormous amount of work that is involved here.

Furthermore, it is difficult to fully automate the process because of many limitations: network problems, disk space, manual specification of the options to build, etc.

The CMVC Level is NOT committed until all the builds are done. Once all the compilations in all the platforms are clean, then the level can be committed. This means that if a bug is encountered after the level is committed, a defect will be opened for the next level. The reason for this is to guarantee that a full extract of a level is buildable in all platforms, even though the function itself may not be correct.

The whole sequence for the complete preparation for a single level is summarized below and is further explained in detail in other sections in this document:

1. Tell other CMVC developers to stop using the affected directories and to do a checkin or a copy to another directory of any files that are important to them.
2. Login as build into the specific system to use, such as "oem-ppc3" for AIX 4 for DB2 V5.
3. Change the directory to the desired combination, such as:

   ```
   cd $HOME/cmvc231/aix4/cs/db2v5
   ```

4. Delete "rm -fr" the appropriate top directory, such as:

   ```
   rm -fr $HOME/cmvc231/aix4/cs/db2v5
   ```

5. Do a full extract (attribute: -full) of the latest committed level, such as 4.

   ```
   Level -extract 4 -release cs231 -node tcaix05      \
      -root "/home/build/cmvc231/solaris/cs/sybase11" \
      -uid 210 -gid 222 -dmask 777 -verbose -full
   ```

6. Do a delta extract of the current non-committed level, such as 5.

   ```
   Level -extract 5 -release cs231 -node tcaix05      \
      -root "/home/build/cmvc231/solaris/cs/sybase11" \
      -uid 210 -gid 222 -dmask 777 -verbose
   ```

7. Modify manually "qmake" to use the appropriate options, such as:

```
PLATFORMOPT=AIX4
PLATFORMSTR=STR_AIX432
DATABASEOPT=db2v5
DATABASESTR=STR_DB2V52FP11
BUILDOPT=all        # that is, both server and line commands
COMPILEOPT=noNetLS  # that is, do not include the NetLS code
```

   Note: The PLATFORMSTR and DATABASESTR values are only used to add the serviceability strings into the executable code, in order for the "what" command to get them. These values do not affect the compilation of the application.

8. Start "qmake" and monitor its progress.

   There are some compilations that usually fail and require manual intervention, such as Solaris. The workarounds are known and easy to apply manually, but there is no way to automate them.

9. After the compilation is done, then do "qmake ship" to copy the shippable code into the "ship" directory.

10. Repeat the extract and build process for the GUI, for the standalone version of the client.

11. As root, change directory to /home/build/cmvc231/installp and kickoff the "make -f Makefile.<platform>.<option>" to create the appropriate tar file.

12. Login as root again to create the tar file for the client.

13. After all the tar files for servers and clients are done, create backup tapes (using the non compressed files).

14. For AIX 4 we have to prepare also installp smit images, which are done by using "bld.all.images4".

15. After all the install smit images are done, then create backup tapes (using the non compressed files).

16. Now it is time to copy the files into:

    - A CD-ROM.
    - The IBM external FTP site.

    In order to ftp the code, it is necessary to compress all the tar and installp smit files.

17. FTP to the staging directory in Boulder and follow the appropriate instructions to place the code there. It takes around 24 hours to be copied in to the real external FTP site.

LOCATION OF THE SOURCE CODE

The source code for the CMVC Server/Client for UNIX, Version 2.3.1, is in the "toro" family:

```
family:   toro
release:  cs231    -> server and line commands
release:  gui231   -> GUI
release:  inst231  -> packaging tools in /home/build/cmvc231/installp
release:  tools231 -> miscellaneous tools in /home/build
```

BUILD/SHIP/PACKAGE DIRECTORY STRUCTURE

Top directory structure - created via shell script

The top portion of the Build/Ship/Package directory structure is shown below and is manually created via the shell script 'mkbldtree' from the release tools231 (see section “Creation of the Build/Ship/Package tree” on page 20).

```
$HOME/
      bin/           * general tools from release tools231
      cmvc231/
           <operating-system>/
               cs/
                       <database>
               gui/
                       stal/
```

Figure 3. Top directory structure - created manually via script

Some examples are:

```
$HOME/cmvc231/aix4/cs/db2v5
$HOME/cmvc231/aix4/gui/stal/x11r5
```

Source directory structure - created by extracting release 'cs231'

The following leaves of the directory structure tree are created when performing a Release/Level extract, and contain the actual source code that will be built.

```
$TOP/
     bin/
     samples/
     src/
        adm/
        archive/
        cat/
        daemon/
        inc/
        lib/
            etc/
            net/
                packet/
                request/
                        file/
                        ni/
                           client/
                           server/
                session/
        restore/
        tools/
              common/
              common1/
              informix/
              oracle7/
              sybase/
              xwind/
        ui/                                  *** Command Lines
           simple/
                  cmd/
                  etc/
                  flags/
```

Figure 4. Source directory structure - created by release 'cs231' (1 of 2)

```
server/
       db/
          common/
          informix/
                   def/
                   gen/
                   init/
                   src/
          oracle7/
                   def/
                   gen/
                   init/
                   src/
          sybase/
                   def/
                   gen/
                   init/
                   src/
          xwind/                  *** for DB2
                   def/
                   gen/
                   init/
                   src/
       etc/
       tx/
       vc/
```

Figure 5. Source directory structure - created by release 'cs231' (2 of 2)

Notes:

1. The $TOP used in this document is:

   ```
   $HOME_FOR_BUILD/<cmvcVRM>/<platform>/cs/<database>

   Where:
     <cmvcVRM>  is 'cmvc231' for 2.3.1
     <platform> is "aix4", ├hpux10", 'solaris'
     <database> is db2v5, informix, oracle7, sybase49

   Example for AIX 4 with DB2 on 'oem-ppc3':
   /home/build/cmvc231/aix4/cs/db2v5
   ```

Install directory structure - created by extracting release 'inst231'

The following leaves of the directory structure are created when performing a Release extract of 'inst231'. (see section “Extraction of inst231 using Release or Level Extract” on page 35).

```
$HOME/
      cmvc231/
           installp/
                    com/
                        cmvccltstal/
                        cmvccltstalmEn_US/
                        cmvcsrvdb2/
                        cmvcsrvinfx/
                        cmvcsrvmEn_US/
                        cmvcsrvora7/
                        cmvcsrvorac/
                        cmvcsrvsyb/
                    hpux/
                    makecdrom/
                    nls/
                    solaris/
```

Figure 6. Bottom directory structure - created by release 'inst231'

Bottom directory structure - created by doing make with 'ship'

The following leaves of the directory structure are created when performing a make while specifying the 'ship' action (see section “Gathering the shippable files (ship step)” on page 42).

```
$TOP/ship/
             client/
                    cmvc/
                         usr/
                             lib/
                                 nls/
                                     msg/
                                         En_US
                             lpp/
                                 cmvc/
                                      bin/
                                      samples/
                    install/
                            r2/
             server/
                    cmvcsrv/
                         usr/
                             lib/
                                 nls/
                                     msg/
                                         En_US
                             lpp/
                                 cmvc/
                                      bin/
                                      bind/
                                      install/
                                      samples/
```

Figure 7. Bottom directory structure - created by doing make with 'ship'

ACTIVITIES TO BE DONE PER EACH VERSION, RELEASE OR MODIFICATION

There are some activities that need to be done only once per each version, each release or each modification, and there is no need to repeat them for every build.

Files that need to be modified due to changes in Copyright or VRM

This section lists the files that need to be modified when there are changes to the Copyright notice (such as a new year) or to the Version-Release-Modification (VRM) number. It is also applicable for the information on the service level, which is included in the source files as an static char that has the strings "@(#)"; in that way a what command can be executed on the executable files to find out this service information, such as:

```
cd /usr/lpp/cmvc/bin
what cmvcd | grep cmvc

The result may be:

  cmvc     CMVC Version 2.3.1.4, (C) Copyright IBM Corp.,1993,1999
  cmvc     5765-207, 5765-202, 5765-397, 5622-063
  cmvc     96/08/28 SID=1.26.1.52
  cmvc     Platform: AIX 4
  cmvc     Database: Informix
```

In order to incorporate these strings we did the following:

- The include file "whatinfo.h" was created that contains all the static strings with @(#) and the keyword "cmvc". Based on Define symbols passed to the compiler, the proper operating system or database string was activated.

  A condensed version of this file is shown in Figure 8 on page 31.

```
char copyrightInfo[]
   = "@(#) cmvc     CMVC Version 2.3.1.4, (C) Copyright IBM Corp.,1993,1999";
char productInfo[]
   = "@(#) cmvc     5765-207, 5765-202, 5765-397, 5622-063";

/* Determine the platform */

#ifdef AIX4
char platformInfo[]
   = "@(#) cmvc     Platform: AIX 4";
#endif

/* Determine the database */

#ifdef informix
char databaseInfo[]
   = "@(#) cmvc     Database: Informix";
#endif
```

Figure 8. Header file with service keywords for the what command

- The include file "whatinfo.c" was created that contains the code that uses these static strings.

  Some compilers will strip these static variables if they are not used in the program, thus, we had to create a dummy section of code that does not do anything practical, but because the variables were actually used, then the compiler will not strip them.

  A condensed version of this file is shown in Figure 9 on page 32.

```
int    dummy;
char * dummyC;
char * dummyP;
char * dummyS;
char * dummyL;
char * dummyD;

dummyC = copyrightInfo;
dummyP = productInfo;
dummyS = serviceInfo;
dummyL = platformInfo;
dummyD = databaseInfo;

if (dummy == -7077)
{
    printf("%s%s%s%s%s\n", dummyC, dummyP, dummyS, dummyL,
               dummyD);
}
```

Figure 9. Source file with service keywords for the what command

- In the main file for each executable, we did the following:
  - Included the following declaration of a static string that contains CMVC expandable keywords in order to place the date when the main file for the executable was extracted, and the version id associated with that file:

    ```
    /* Support Information */
    char serviceInfo[] = "@(#) cmvc     %D% SID=%I%";
    ```

  - Included the header file whatinfo.h

    ```
    /* Include the header file with the information for the 'what' command */
    #include "whatinfo.h"
    ```

  - Included in the main routine, immediately after the declaration of the variables and before any executable statement, the code file whatinfo.c

    ```
    /* Include the source file with the information for the 'what' command */
    #include "whatinfo.c"
    ```

Release "cs231"

The following files must be changed in release "cs231" (the actual code) in order to update the Copyright information and the Title:

- *.c and *.h

```
qmake               (when adding new platforms, DBMSs)
MAKE.cmvc           (when adding new platforms, DBMSs)
Make.dbms.platform  (specific, such as: Make.sybase.solaris)
src/inc/whatcmd.h   (Operating System for the client)
src/inc/whatinfo.h  (Operating System and DBMS, for the server)
src/cat/cmvc.msg    (Message catalog)
```

- Readme files:

  ```
  src/adm/README.what.is.new
  ```

Release "gui231"

The following files must be changed in release "gui231" (the actual code) in order to update the Copyright information and the Title:

- source code

  ```
  cmvc.c
  ```

- message catalog

  ```
  cmvchelp.msg
  ```

- Read me files

  ```
  hp/README.gui
  ibm/README.gui
  solaris/README.gui
  ```

- Build utilities

  ```
  qmake  (when adding new platforms)
  solaris/stal/Makefile.x11r5.solaris26  (specific)
  ```

Release "inst231"

The following files must be changed in release "inst231" (the files for packaging) in order to update the Copyright information and the Title:

- Shell scripts

  ```
  bld.all.images4
  aix4/tarinstall.aix4
  hp10/cmvcinstall
  hp10/tarinstall.hpux10
  solaris/cmvcinstall
  solaris/tarinstall.solaris
  makecdrom/cmvcinst
  makecdrom/cmvcinst.hpux10
  ```

- Readme files, such as:

```
platform/tar/README.contents.tar.txt
```

Files that need to be modified when adding new files

At this stage, if there are new files to be packaged they will most likely be:

- Documentation files in src/adm/README.*. Thus, only the src/adm/Makefile needs to be updated.
- Samples in samples/*. Thus, the samples/Makefile and either samples/client.samples or samples/server.samples need to be updated.

PREPARATION FOR THE BUILD

This section is only applicable for the person who is responsible for the production build on the build machine.

For the target builds where the "build" directory is exported from tcaix05, you can perform all the extracts and the file checkout/checkin activities from build@tcaix05 and all the builds from build@target_server.

1. Login as "build" to the target server.

   The home directory is "/home/build". This directory resides actually in tcaix05 and is exported and mounted in the server.

Note: We cannot do incremental builds for CS (Client/Server), because there is not a good way to keep track of the dependencies with the current structure of the make files. This is the reason for a complete cleanup of the build-tree for each build.

Process Tracks, Levels and Level Members

Ensure that there is a level with all the appropriate tracks. For more details, see “Process model” on page 10.

Extraction of the tools for the build and packaging process

Source code extraction requires some knowledge of CMVC operations. Users are suggested to refer to CMVC manual "Understanding IBM AIX CMVC/6000 Concepts" for details.

Extraction of tools231 using Release or Level Extract

The release 'tools231' contains utilities to be included in the $HOME/bin directory that will help to perform tasks such as copying the code from the ship tree into the final ('real') directories for the server, client and gui code. This procedure will allow us to have a setup for testing the code, using the directories that the end-users will use.

You can use a working version of any CMVC client to extract the source files as follows:

1. Start the CMVC client with the family **toro**.
2. Bring up the Release window and select the release **tools231**.

   An alternative way is to bring up the Level window and select the desired level.
3. Select the action "extract" and specify:
   - For releases, choose to extract the committed versions of the files

     Choose the option "Current versions" if this is the first build of the release (in case there are no committed versions yet).
   - For levels, choose to extract either Delta or Full.
   - Specify the Destination (we use a local extract, without involving NFS):

     ```
     host:      nomount
     directory: /home/build/bin
     ```
   - Specify the File and Directory Access:

     ```
     User number (UID):      210
     Group number (GID):     222
     File permissions:
     Directory permissions:  777
     ```
   - Choose to expand the keywords (this is the default).
   - Choose NOT to perform the CRLF conversion (this option is only available from the DOS/Windows and OS/2 clients).
4. Select OK to start the extraction of the files for the release.

Extraction of inst231 using Release or Level Extract

The release 'inst231' contains utilities to be included in the $HOME/cmvc<VRM>/installp directory that will help to perform tasks such as preparing the installable images for CD-ROM and tape for all the combinations.

You can use a working version of any CMVC client to extract the source files as follows:

1. Start the CMVC client with the family **toro**.

2. Bring up the Release window and select the release **inst231**.

   An alternative way is to bring up the Level window and select the desired level.

3. Select the action "extract" and specify:

   - For releases, choose to extract the committed versions of the files

     Choose the option "Current versions" if this is the first build of the release (in case there are no committed versions yet).

   - For levels, choose to extract either Delta or Full.
   - Specify the Destination (we use a local extract, without involving NFS):

     ```
     host:      tcaix05
     directory: /home/build/cmvc/installp
     ```

   - Specify the File and Directory Access:

     ```
     User number (UID):      210
     Group number (GID):     222
     File permissions:
     Directory permissions:  777
     ```

   - Choose to expand the keywords (this is the default).
   - Choose NOT to perform the CRLF conversion (this option is only available from the DOS/Windows and OS/2 clients).

4. Select OK to start the extraction of the files for the release.

BUILDING THE CMVC SERVER AND LINE COMMANDS

This chapter describes how to build the CMVC Server and Line Commands, where to obtain the source code, what is the directory structure used for the operations, and what is the sequence of these operations.

COMPLETE CLEANUP OF THE BUILD TREE

This type of cleanup is intended for those occasions when it is necessary to start from an empty build directory:

1. Change to the $TOP directory
2. Execute the following command to cleanup all the subdirectories:

   ```
   rm -fr *
   ```

Cleanup of the previous build

In case there is no need to extract the source code again, but is necessary to perform another build, then you can follow the steps below to trigger a new build, ensuring that all the files will be compiled.

1. Change the directory to the TOP directory, such as $HOME/cmvc231
2. Change the directory to the desired platform-type-database, such as "aix4/cs/db2v5".
3. Run qmake clean
4. If you are going to do the ship step later on, you should issue the following command:

   ```
   rm -rf ship
   ```

EXTRACTION OF THE SOURCE FILES

Source code extraction requires some knowledge of CMVC operations. Users are suggested to refer to CMVC manual "Understanding IBM AIX CMVC/6000 Concepts" for details.

Extraction using Release or Level Extract

Note:

It is necessary to perform an individual source extraction for each entry in the build tree that has "cs" in the Path, that is, a separate source extraction will have to occur in order to build the client/server for each platform. Refer to Figure 3 on page 25 to see each of the platforms.

You can use a working version of any CMVC client to extract the source files as follows:

1. Start the CMVC client with the family **toro**.
2. Bring up the Release window and select the release **cs231.**.

   An alternative way is to bring up the Level window and select the desired level.
3. Select the action "extract" and specify:
   - For releases, choose to extract the committed versions of the files

     Choose the option "Current versions" if this is the first build of the release (in case there are no committed versions yet).

     If you are doing Unit Testing or Function Testing you may specify to extract the current versions.
   - For levels, choose to extract either Delta or Full.
   - Specify the Destination (we use a local extract, without involving NFS):

     ```
     host:      tcaix05
     directory: /home/build/cmvcVRM/x/y/z

     where
         VRM is the Version and Release number, such as: cmvc231

           x is aix4, hpux10, solaris
           y is cs or gui
           z is for the database (only if building client/server):
                   oracle7
                   informix
                   sybase49
                   db2v5
     ```
   - Specify the File and Directory Access:

     ```
     User number (UID):      210
     Group number (GID):     222
     File permissions:
     Directory permissions:  777
     ```
   - Choose to expand the keywords (this is the default).

4. Select OK to start the extraction of the files for the release.

BUILDING THE CODE

The script "qmake" in the $TOP directory is a front end to the MAKE.cmvc shell script which is really the build shell script.

To build the code follow these steps:

1. Change the file permissions for qmake to be 755:

   ```
   chmod 755 qmake
   ```

2. Modify manually "qmake" to use the appropriate options, such as:

   ```
   PLATFORMOPT=AIX4
   PLATFORMSTR=STR_AIX432
   DATABASEOPT=db2v5
   DATABASESTR=STR_DB2V52FP11
   BUILDOPT=all        # that is, both server and line commands
   COMPILEOPT=noNetLS  # that is, do not include the NetLS code
   ```

   Note: The PLATFORMSTR and DATABASESTR values are only used to add the serviceability strings into the executable code, in order for the "what" command to get them. These values do not affect the compilation of the application.

3. Execute qmake
4. The output of the build will be stored in a file named nohup.build".
5. You will see an appropriate message to indicate that the build has finished.
6. You can kill the build process by doing Control-C.

MISCELLANEOUS NOTES

Notes on the MAKE.cmvc shell script

The MAKE.cmvc in the $TOP directory is used to initiate the build and ship activities. The format and the options for this command are shown below:

```
Usage:   MAKE.cmvc -p platform <-d dataBase> <-b buildOption>
                  <-c compileOption> <action>

Where:
      -p platform      Enter: aix4, hpux10, solaris
      -d dataBase      Enter: oracle7, informix, sybase4, sybase, db2v5
```

```
-b buildOption   Enter: all, server, client

-c compileOption Enter: normal, debug, FVT, noNetLS

action           Enter additional action such as 'clean' or 'ship'
```

Use MAKE.cmvc to set the correct environment variables for the Client/Server build.

If you would like to leave the job running overnight, or have it running in background, it is recommended that you use the 'nohup' command as follows:

```
nohup MAKE.cmvc -p aix4 -d db2v5 -b all -c normal build > nohup.build &
```

This will cause a file called 'nohup.build' to be generated. If you would like to see the progress of the build, you can look at the file that is being generated by issuing the command:

```
tail -f nohup.build
```

Once the build ends, you can Ctrl-c the 'tail' process in order to get back to the prompt.

FILES AFFECTED BY DIFFERENT MAKE.CMVC -C OPTIONS

In order to clarify what is the impact of compiling with different options the CMVC product, Figure 10 and Figure 11 on page 41 lists the files (from release cs231 and level 004) which are affected (their names are relative with respect to the TOP directory):

```
BETA option:
src/daemon/cmvcd.c
MAKE.cmvc

DEBUG option:
src/daemon/cmvcd.c
MAKE.cmvc

FVT option:
src/lib/server/etc/notify.c
src/daemon/cmvcd.c
MAKE.cmvc

NOFORK option:
MAKE.cmvc

IBMIUO option:
src/ui/simple/etc/netls.c
MAKE.cmvc
```

Figure 10. Files affected by different MAKE.cmvc -c options, part 1

```
NO_NETLS option:
src/ui/simple/cmd/Access.c
src/ui/simple/cmd/Approval.c
src/ui/simple/cmd/Approver.c
src/ui/simple/cmd/File.c
src/ui/simple/cmd/Component.c
src/ui/simple/cmd/Defect.c
src/ui/simple/cmd/Environment.c
src/ui/simple/cmd/Host.c
src/ui/simple/cmd/Size.c
src/ui/simple/cmd/Report.c
src/ui/simple/cmd/Notify.c
src/ui/simple/cmd/Test.c
src/ui/simple/cmd/Track.c
src/ui/simple/cmd/User.c
src/ui/simple/cmd/Coreq.c
src/ui/simple/cmd/Verify.c
src/ui/simple/cmd/Fix.c
src/ui/simple/cmd/Level.c
src/ui/simple/cmd/LevelMember.c
src/ui/simple/cmd/Feature.c
src/ui/simple/cmd/Release.c
src/ui/simple/cmd/Migrate.c
src/ui/simple/etc/netls.c
MAKE.cmvc
```

Figure 11. Files affected by different MAKE.cmvc -c options, part 2

Note: The DEBUG option uses also NO_NETLS and NOFORK

Making a DEBUG driver

To make the Client/Server with DEBUG compile options, it is necessary to use the -c flag for MAKE.cmvc as follows:

```
MAKE.cmvc  -p airx4  -d db2v5  -b all  -c debug
                                         *****
```

In MAKE.cmvc, the NO_NETLS define is added in the debug driver. Thus, if you want to debug the NetLS portion of the code, then you need to modify temporarily the MAKE.cmvc file by eliminating the -DNO_NETLS from the appropriate section. Furthermore, it is important not to exclude the NetLS libraries, by eliminating the following lines that disable the NetLS libraries:

```
NETLS_LIBS=
NCS_LIBS=
```

Making an FVT driver

To make the Client/Server for FVT, it is necessary to use the -c flag for MAKE.cmvc as follows:

```
MAKE.cmvc  -p aix4  -d db2v5  -b all  -c FVT
                                        ***
```

The following files are affected by the FVT compile option:

```
MAKE.cmvc
src/lib/server/etc/notify.c
```

Note: Because only the notify.c source file is affected, the client code that is built with this option is NOT affected.

Making a BETA driver

By specifying a BETA driver, the ONLY difference from a normal driver is that the 'what' service information will say 'This is BETA code'.

The objective of a beta driver is to be provided when a level has not yet been committed, such as when providing a temporary fix to a customer.

Making a Gold driver

To provide executable code that is the Gold driver, it is necessary to use the -c flag for MAKE.cmvc as follows:

```
MAKE.cmvc  -p aix4  -d db2v5  -b all  -c normal
                                        ******
```

Note: Be sure to change the type of the CMVC Level to "gold".

GATHERING THE SHIPPABLE FILES (SHIP STEP)

Upon successful completion of the build step, run qmake ship to copy the files that need to be delivered to the customers from the appropriate build directory to the ship directories.

Then we could package the code in the ship directories into the installable images (in tar format or installp-smit) for the installation and verification tests.

BETA INSTALLATION

You do this step in lieu of the last step in the build and ship section.

You can copy the stuff from the ship directory into the normal directories (that is, /usr/lpp) by doing the following:

1. Logon as build.
2. Change to the bin directory
3. Change the user to root.
4. Issue the following command:

   ```
   ./copy.server <top_directory_where_ship_resides> <db>
   ./check.server <top_directory_where_ship_resides>
   ./copy.client <top_directory_where_ship_resides> <db>
   ./check.client <top_directory_where_ship_resides>

   Example for Solaris with Sybase 4,9:
   ./copy.server $HOME/cmvc231/solaris/cs/sybase49 sybase49
   ```

   The output of the 'check' scripts should be piped to more.
5. Exit from root.

KNOWN BUILDING PROBLEMS

This section contains a compilation of the errors found during the building process.

If the build fails with generic problems

Here are things to check if a build fails. This list is operating system dependent.

- Prior to running the MAKE.cmvc with a 'ship' action, from the /src/tools/xwind directory, issue: rm *.o *.c
- After touching the message catalog (src/cat/cmvc.msg), do a MAKE.cmvc with a 'clean' action and rebuild.

If the build fails for AIX

None.

If the build fails for HP-UX

- In HP-UX for client/server during "qmake ship":

  ```
  cpp: "execute.c", line 1: error 4036: Can't open include file
  'sqlhdr.h'.
  *** error code 1
  ```

  To solve the problem go to the corresponding directories and touch all the object files:

  ```
  cd src/lib/server/db/informix/gen
  touch *.o
  cd src/lib/server/db/informix/src
  touch *.o
  ```

  Then restart the build.

If the build fails for Solaris

- During the compilation of msgdef.c, you can ignore the following warning:

  ```
  "msgdef.c", line 153: warning: identifier redeclared: fwrite
    current : function() returning int
    previous: function(pointer to const void, uint, uint, pointer to struct
  {int _cnt, pointer to uchar _ptr, pointer to uchar _base, ucha... :
  "/usr/inclue/stdio.h", line 280
  ```

- During the first phase of the build, you can ignore the following warning (it is an old step that tried to compile the Japanese message catalog):

  ```
  LANG=ja; export LANG;
  couldn't set locale correctly
  ```

- During the pre-compilation by Sybase 11 "cpre", there are some warnings that can be ignored:

  ```
  M_WHEN_WARN, Unable to find the SQL statement 'WHENEVER WARNING'
  M_WHEN_NF, Unable to find the SQL statement 'WHENEVER NOT FOUND'
  ```

- During the compilation of certain *.c files, you may ignore the following warnings:

  ```
  "accept.c", line 95: warning: improper pointer/integer combination: arg #1
  "accept.c", line 166: warning: argument #2 is incompatible with prototype:
     prototype: pointer to struct sockaddr {ushort sa_family, array½14‘ of
     char sa_data} : "/usr/include/sys/socket.h", line 333
     argument : pointer to struct sockaddr_in {ushort sin_family, ushort sin
     port, struct in_addr {..} sin_addr, array½8‘ of char sin_zero}
  "accept.c", line 223: warning: argument #1 is incompatible with prototype:
     prototype: pointer to const char : "/usr/include/netdb.h", line 190
     argument : pointer to struct in_addr {union {..} S_un}
  ```

- During the compilation of netls.c, ignore the following warning:

  ```
  "netls.c", line 309: warning: empty translation unit
  ```

- During the compilation of binToLong.c, ignore the following warning:

  ```
  "/usr/include/stdio.h", line 81: warning: macro redefined: BUFSIZ
  ```

Known Problems when Testing the code

- In the Solaris system, if the cmvcd fails to run due to a semaphore problem, then do the following:

  1. su as root
  2. Issue the command: stopCMVC $LOGNAME

     This command will clear up the semaphores and other system resources.

BUILDING THE CMVC GUI

BUILD/SHIP/PACKAGE DIRECTORY STRUCTURE FOR THE GUI

The Build/Ship/Package directory structure for the GUI is shown below. This tree is automatically created when performing a Release/Level extract. However, the $TOP portion of the tree is manually created.

```
$TOP/
     config/
     help/
     hp/
         stal/
     ibm/
         stal/
     menus/
          Softdm/
                 CM/
                    Softcmvc/
     solaris/
         stal/
```

The following notes clarify important details:

- The $TOP used in this document is:

  ```
  $HOME_FOR_BUILD/<cmvcVRM>/<platform>/gui/<x11>/<mode>

  where:
    <cmvcVRM>  is 'cmvc231'
    <platform> is "aix4", hpux10", "solaris"
    <x11>      is 'x11r5' (Motif 1.2)
    <mode>     is 'stal' (for stand-alone)

  Example for Solaris, standalone:
    /home/build/cmvc231/solaris/gui/x11r5/stal
  ```

PREPARATION FOR THE BUILD

This section is only applicable for the person who is responsible for the production build on the build machine.

For the target builds where the "build" directory is exported from tcaix05, you can perform all

the extracts and the file checkout/checkin activities from build@tcaix05 and all the builds from build@target_server.

1. Login as "build" into the target server.

   The home directory is "/home/build". This directory resides actually in tcaix05 and is exported and mounted in the server.

Source code extraction requires some knowledge of CMVC operations. Users are suggested to refer to CMVC manual "Understanding IBM AIX CMVC/6000 Concepts" for details.

COMPLETE CLEANUP OF THE BUILD TREE

This type of cleanup is intended for those occasions when it is necessary to start from an empty build directory, such as after many changes have been done to the source files. If minor/few changes were done to the source files, this cleanup action might not be necessary.

1. Change to the $TOP directory
2. Execute the following command to cleanup all the subdirectories:

   ```
   rm -fr *
   ```

EXTRACTION OF THE SOURCE FILES

Extraction using Release or Level Extract

Note: It is necessary to perform an individual source extraction for each combination of <platform><gui><x11><stal>.

You can use a working version of any CMVC client to extract the source files as follows:

1. Start the CMVC client with the family **toro**.
2. Bring up the Release window and select the release **gui231**.

   An alternative way is to bring up the Level window and select the desired level.
3. Select the action "extract" and specify:

   - For releases, choose to extract the committed versions of the files.

     Choose the option "Current versions" if this is the first build of the release (in case there are no committed versions yet).

     If you are doing Unit Testing or Function Testing you may specify to extract the current versions.

- For levels, choose to extract either Delta or Full.
- Specify the Destination (we use a local extract, without involving NFS):
- Specify the File and Directory Access:

```
User number (UID):      210
Group number (GID):     222
File permissions:
Directory permissions:  777
```

- Choose to expand the keywords (this is the default).

4. Select OK to start the extraction of the files for the release.

BUILDING THE CODE

Cleanup of object modules and message catalogs

This type of cleanup is intended for a complete rebuild using the code that already exists in the build tree (that is, for incremental development). This is useful also when changing the message catalog, in order to ensure that all the code is compiled with the latest help.c file.

1. Change to the $TOP directory, such as:

   ```
   cd $HOME/cmvc231/aix4/gui/x11r5/stal
   ```

2. Execute the shell script to cleanup by removing the object modules and the generated message catalog:

   ```
   qmake clean
   ```

CONSIDERATIONS BEFORE BUILDING

It is very important to take into account the following considerations before building the code.

Making a DEBUG driver

So far, there is no provision in the makefiles to specify to compile with the debug option.

Making an FVT driver

In contrast with the Client/Server code, there is no provision and no need for an FVT version.

Making a BETA driver

To provide executable code that has a BETA identifier when performing the UNIX command "what cmvc | grep cmvc", edit the "qmake" file and ensure that the FLAGS that are used with the invokation of make has the string "BETA=-DBETA":

```
export FLAGS="-DBETA -D$PLATFORMSTR"
```

The BETA identifier looks like:

```
cmvc     *** This is BETA code ***  Date: 2000/06/28
```

Only "cmvc.c" and "migResources.c" need to be compiled with the BETA flag. This means that it might be possible to first perform a normal compile, then touch these 2 files and then invoke the make command again but this time by adding the BETA flag as indicated.

Note: Be sure to change the type of the level to "beta".

Making a Gold driver

To provide executable code that is the Gold driver, do not specify any additional strings.

Note: Be sure to change the type of the level to "gold".

BUILDING THE GUI

1. Change to the $TOP directory, such as:

   ```
   cd $HOME/cmvc231/aix4/gui/x11r5/stal
   ```

2. Change the permissions to qmake to 755:

   ```
   chmod 755 qmake
   ```

3. Edit qmake and specify the platform, the mode and the X11 version. This script invokes the proper make file.

   ```
   make  -f <platform>/<mode>/Makefile<.x11r5>

   Where
        <platform> is:  ibm, solaris, hp
        <mode> is:      stal
        <.x11r5> is needed if building for x11r5

   For X11R5:
   make  -f solaris/stal/Makefile.x11r5

   Examples for making a beta:
   make  -f ibm/stal/Makefile BETA=-DBETA
   make  -f solaris/stal/Makefile.x11r5 BETA=-DBETA
   ```

An actual example of the active statements in the qmake script (that is, lines that are not commented) is shown below:

```
export PLATFORMOPT=solaris
export PLATFORMSTR=STR_Solaris26
export BUILDOPT=stal
export FLAGS="-D$PLATFORMSTR"
export X11R=.x11r5.solaris26
```

These variables will be used to invoke the make as follows:

```
nohup make -f $PLATFORMOPT/$BUILDOPT/Makefile$X11R FLAGS="$FLAGS"
```

4. It is recommended that if you want to leave the job running overnight or to have it running in the background, use the command 'nohup' as follows:

   ```
   For X11R5:
   nohup   make -f solaris/stal/Makefile.x11r5   > nohup.out &
   ```

   This will result in a file named 'nohup.out' to be generated. If you want to see the progress of the build, you can peek at this nohup.out file that is being generated by issuing the command:

   ```
   tail -f nohup.out
   ```

   Once the build ends (the link step for the 'cmvc' executable), you can ctrl-c the process for 'tail', in order to get back to the prompt.

KNOWN BUILD PROBLEMS

- When compiling the db2 migration tools for Oracle in an AIX machine, it might be possible to get the following error message:

  ```
  PCC-F-NOERRFILE, unable to open error message file, facility PR2
  ```

  To overcome this problem, verify that the ORACLE_HOME is properly defined in the file Make.xxx.yyy.

- If the cmvchelp.msg or the cmvcui.msg files (message catalogs) have syntax errors, which are most likely to be a lack of '\n\' at the end of the sentences, you will see the following error message when compiling the code (such as):

  ```
  $ make -f solaris/stal/Makefile.x11r5
  or
  $ make -f ibm/stal/Makefile

  LANG=C; export LANG; \
  gen_msg cmvchelp cmvchelp.msg > cmvchelp.msg1; \
  gencat cmvchelp.cat cmvchelp.msg1; \
  rm cmvchelp.msg1;
  *** Error code 139
  make: Fatal error: Command failed for target  cmvchelp.cat'
  ```

- If the *.msg message catalogs have a double quote inside the help strings, then the compilation of help.c will fail.

Known build problems for Solaris

- During the compilation of utilities.c, ignore the warnings:

```
"./hour.h", line 11: warning: initializer does not fit or is out of range
"./hourMask.h", line 11: warning: initializer does not fit or is out of range
```

COPYING THE OUTPUT FILES INTO THE SHIP DIRECTORY

It is not necessary to perform a "ship" step (as with the Client/Server code) to copy the output files into a ship directory. In a sense, the current build directory tree is also the ship tree.

PERFORMING A SANITY TEST (BASED ON A PSEUDO-INSTALLATION)

Perform the following steps to perform a sanity test of the CMVC GUI recently built:

1. Ensure that you are in the $TOP directory where the build took place.
2. Ensure that the DISPLAY environment is properly set, such as:

   ```
   export DISPLAY=oem-sn13:0.0
   ```

3. Invoke the CMVC GUI "cmvc" executable from the current directory:

   ```
   ./cmvc
   ```

PACKAGING THE CMVC SERVER, LINE COMMANDS AND GUI

This chapter describes how to package the CMVC server, line commands and GUI for UNIX.

The code for CMVC is distributed only as a full replacement of the code (also called, a fullpak). That is, there are no delta updates to the installed code for CMVC. Whether you are installing for the first time, or you are installing the CMVC code in a machine that already has CMVC, the installation process is the same, and the amount of files to be copied is also the same.

For AIX, HP-UX and Solaris, we provide the "tar" format in which the system administrator can specify where to install the code. The default is "/usr/lpp/cmvc". One disadvantage is that because this method bypasses the system utilities to install code in the appropriate operating system, it is not possible to use system utilities to query if CMVC is installed, nor to uninstall it. The uninstall process is simply the removal of the /usr/lpp/cmvc directory (or the appropriate directory, if the default was not used) by using the "rm" command as root; however, the root user needs to be very careful to avoid NOT deleting other directories or files that are not related to CMVC.

For AIX, we also provide the installp format which can be used in conjunction with "smit" to install the code. The advantage is that the AIX system utilities can be used to query if CMVC is installed (lslpp) and "smit" can be used to uninstall the product. The disadvantage is that the end user cannot specify another directory where to install the product, that is, via smit, you can only use /usr/lpp/cmvc as the home for the CMVC code.

OVERVIEW OF PACKAGING FOR THE SERVER

The CMVC server for UNIX that the system administrator installs (from CD-ROM or from tape) is the collection of files shown below. These files are placed together in a single installable image, one per platform and per database (DB2, Oracle 7, Informix, and Sybase 4.9). For example, one installable image for the DB2 server for AIX, one installable image for Oracle 7 for HP-UX.

CMVC UNIX files, which are built and shipped (ship/server) as part of the server (using MAKE.cmvc with either '-b server' or '-b all') process described in this document:

- directory: /usr/lpp/cmvc/bin

```
age
binToLong
chfield
cmvcarchive
cmvcd
cmvchost
cmvcrestore
dspmsg
monitor
notifyd
resetAge
sclean
vcPath
```

- directory: /usr/lpp/cmvc/bind # only for database DB2

```
*     # all files
```

- directory: /usr/lpp/cmvc/db2Convert # only for database DB2

```
*     # all files
```

- directory: /usr/lpp/cmvc/install

```
*     # all files
```

- directory: /usr/lpp/cmvc/samples

```
server.samples
<all the samples listed in server.samples>
```

- directory: appropriate directory for Message Catalog (see bin/copy.server)

```
cmvc.cat
```

OVERVIEW OF PACKAGING FOR THE CLIENT (LINE COMMANDS AND GUI)

The CMVC client for UNIX that the system administrator installs (from CD-ROM or from tape) and that the end-user uses is the collection of files shown in the next subsections. These files are placed together in a single installable image, one per platform and per mode (standalone). For example, one installable image for the Standalone client for AIX.

Standalone files

- Motif GUI files, which are built by the process described in this document:
  - directory: /usr/lpp/cmvc/bin

```
README.gui
cmvc
```

  - directory: /usr/lpp/cmvc/samples

    ```
    Cmvc
    system.cmvcrc
    migResources
    ```

  - directory: appropriate directory for Message Catalog (see bin/copy.gui.*)

    ```
    cmvcui.cat
    cmvchelp.cat
    ```

  - directory: /usr/lib/X11

    ```
    system.cmvcrc
    XKeysymDB.cmvc     # only for Solaris
    motifbind.cmvc     # only for Solaris
    ```

  - directory: /usr/lib/X11/app-defaults

    ```
    Cmvc
    ```

  - directory: /usr/include/X11/bitmaps

    ```
    cmvc.icon
    cmvc.error
    cmvc.info
    cmvc.question
    cmvc.tree
    ```

- CMVC UNIX line commands, which are built and shipped (ship/client) as part of the client (using MAKE.cmvc with either '-b client' or '-b all').

  - directory: /usr/lpp/cmvc/bin

    ```
    Access
    Approval
    Approver
    Component
    Coreq
    Defect
    Environment
    Feature
    File
    Fix
    Host
    Level
    LevelMember
    Migrate
    Notify
    Release
    Report
    Size
    Test
    Track
    User
    ```

```
Verify
```

– directory: /usr/lpp/cmvc/samples

```
client.samples
<all the samples listed in client.samples>
```

– directory: appropriate directory for Message Catalog (see bin/copy.client)

```
cmvc.cat
```

- CMVC UNIX utilities, which are built (in src/adm) and shipped (in ship/client) as part of the server (using MAKE.cmvc with either '-b server' or '-b all').

– directory: /usr/lpp/cmvc/bin

```
Fileimport
Filemap
Filemigrate
dspmsg
xecit
```

save.

PREPARING AIX SMIT INSTALLP IMAGES

To prepare the AIX smit installp images do:

1. Login as build.
2. Go where the installation tools are:

   ```
   cd cmvc231/installp
   ```

3. For AIX 4 run:

   ```
   bld4.installp
   ```

   An individual preparation for the smit installp images is performed as follows (for CMVC release 2.3, this is already in BldShip)

   - For AIX 4, do:

     ```
     BldShip4  db2  0001 0004
                          *******> level
     ```

4. You can disregard the warning message:

   ```
   End of inslist file: No such file or directory
   ```

PREPARING THE FILES TO GENERATE THE INSTALLATION IMAGES

The installation images for CMVC are prepared in installp format (only for AIX) and in tar format for AIX, HP-UX and Solaris.

"Make" files are used to generate the installation images and they may need to be updated to reflect the directories where the build took place.

For example, let's assume that the original make file for preparing the CMVC server for Sybase had directory "sybase4" and now the directory that contains the new built code is "sybase11", then the following file would need to be updated to reflect the new directory:

1. Login as build.
2. Change the directory where the installation files are located:

   ```
   cd cmvc231/installp
   ```

3. Edit the following files and in the line that begins with "BUILDER_DIR", change the old directory "sybase4" for the new directory "sybase11".
   - Makefile.SOLARIS.sybase
   - Makefile.SOLARIS.client

GENERATING INSTALLATION IMAGES IN TAR FORMAT

To generate the installation images in tar format which are applicable to all platforms do the following:

1. Login as root into the appropriate platform. For example, to prepare the tar images for HP-UX, you need to login as root in an HP-UX machine.

   You need to be the "root" user id in order to prepare the installation images, because the ownership of the files needs to be "bin:bin" and the permissions for the "cmvcd" executable requires the "s" bit.

   If you are already logged in as "build", then issue:

   ```
   su - root
   ```

2. Change to the directory "installp". Assuming that the home directory of the "build" user id in that machine is "/home/build", then do the following:

   ```
   cd /home/build/cmvc231/installp
   ```

3. Execute the make file that will prepare the CMVC Client for Solaris:

   ```
   make -f Makefile.SOLARIS.client
   ```

   The generated tar files are:

- Standalone client:

  `$HOMEBUILD/cmvc231/installp/solaris/tar/solstal.tar`

- Message catalog for the GUI:

  `$HOMEBUILD/cmvc231/installp/solaris/tar/solstalm.tar`

4. Execute the make file that will prepare the CMVC Server for Sybase on Solaris:

   ```
   make -f Makefile.SOLARIS.sybase
   ```

   You need to answer what type of home directory is used (it depends on the platform):

   ```
   Select the HOME directory for build:
   1  /home/build        (AIX 3, AIX 4, SunOS, HP-UX 10)
   2  /export/home/build (Solaris)
   3  /disk2/users/build (HP-UX 9)
   ```

   The generated tar files are:

   - Server for Sybase:

     `$HOMEBUILD/cmvc231/installp/solaris/tar/solsyb.tar`

   - Message catalog for the server and line commands:

     `$HOMEBUILD/cmvc231/installp/solaris/tar/solsrvm.tar`

COLLECT THE INSTALLATION FILES IN A DISTRIBUTION MEDIA

Once the installation images are generated, then it is necessary to collect all the files related to the installation process in a distribution media, such as a CD-ROM.

The directory where the generated files were placed from the previous step, also contains the README file and the Korn shell script to interact with the end user (to ask which component to install and in which directory) and then to proceed to un-tar the files. For example:

```
$ cd $HOMEBUILD/cmvc231/installp/solaris/tar/

$ ls -1
README.contents.tar.txt
README.inst.tape.txt
README.inst.tar.txt
solora7.tar
solsrvm.tar
solstal.tar
solstalm.tar
solsyb.tar
tarinstall.solaris*
```

You can copy all the above files (actually, you may want to include only the README called "README.inst.tar.txt") into the distribution media. The file "README.inst.tar.txt" has the

installation instructions that the system administrator has to perform as user "root" to install the CMVC code.

APPENDIX A. INTRODUCTION TO THE CHECKLISTS

This section consists of the following:

- A quick status checklist that in one page of text contains all the information about the latest builds for UNIX only. This checklist can be maintained on-line.

  1. Use a forward slash to indicate when an activity has begun.

     This will allow you to see quickly which tasks need your attention.

  2. Use a plus sign to indicate when an activity has ended.

- A set of generic checklists that can be used to keep track of all the build activities that are necessary to produce CMVC 2.3.1 Client/Server and GUI for UNIX.

  1. Fill in the platform field in the title of the table. (Recommendation: use red ink)
  2. Fill in the host name in the second row in the header of the table. (Recommendation: use red ink)
  3. Use a forward slash to indicate when an activity has begun. (Recommendation: use pencil)

     This will allow you to see quickly which tasks need your attention.
  4. Use a backward slash on top of the forward slash to form an X to indicate when an activity has ended. (Recommendation: use pencil)

The explanation of all the steps that are needed to build, package and make tapes can be found in their appropriate files in the directory /home/build/process.

MISCELLANEOUS NOTES

- When extracting source code for AIX 4, HP-UX 10, or Solaris, use the following information:

  ```
  hostname: tcaix05
  userid:   210
  group id: 222
  ```

- The $HOMEBUILD directory depends on the host, such as in oem-ibm: /u/build
- The target directory depends on the combination of platform and database, such as in Oracle 7.3 for AIX: $HOMEBUILD/cmvc231/aix4/cs/oracle73
- When doing remote login from AIX into other platforms, take the following into account:

  1. Solaris

     ```
     Terminal type: xtermm
     This is code already in the .profile
     ```

2. HP-UX

   The following information is not in the profile and needs to be entered when asked:

   ```
   Terminal type: ansi
   ```

   When possible, you can telnet to an HP-UX box, set the DISPLAY variable back, then issue:

   ```
   hpterm &
   ```

APPENDIX B. CMVC FOR UNIX, BUILD AND PACKAGE STATUS CHECKLIST

Use as reference the following real quick checklist status for the building of level 2.3.1.6 for Solaris.

```
Subject: Status of CMVC CS/GUI builds for 2.3.1 Level 6
Date last updated: 22-Jan-2001

*** Client/Server builds *****       extract q s  tar  obj  t  t  ftp  c
                                            m h  s c  s c  e  a  i e  d
                                            a i  r l  r l  s  p  n x  r
                                            k p  v t  v t  t  e  t t  o
                                            e                        m

AIX 421 oem-ppc3 db2v52    all
AIX 421 oem-ppc3 oracle734    server
AIX 421 oem-ppc3 informix72   server

HP10.20 oem-hp10 oracle734 all
HP10.20 oem-hp10 informx72    server
HP11.00 oem-hp14 informx73 all    (runtime problems compile on oem-hp10)

Sol 24  oem-sn01 sybase49 server
Sol 251 oem-sn12 oracle73 server
Sol 26  oem-sn13 sybase11 all        f5 d6   + +  +        /           -

*** Motif GUI  builds for CMVC ***   extract qmake

AIX 4.2 oem-ppc3  x11r5/stal
HPUX 10 oem-hp10  x11r5/stal
Solaris oem-sn13  x11r5/stal         f5 d6   + +    +      /           -

*** Notes about the Operating Systems  **********************************

oem-ppc3  AIX 4.2.1:   cmpc3db2 (DB2 UDB 5.2 FP 8)
                       cmpc3ora (7.3.4), cmpc3inf (7.2)

oemk250a  HP-UX 10.20: Oracle 7.3
oem-hp10  HP-UX 10.20: cmhp10or (Oracle 7.3.4), cmhp10in (Informix 7.23)
oem-hp14  HP-UX 11.00: cmhp14in (Informix 7.3)

oem-sn01  Solaris 2.4: cmvcsn01 (Sybase 4.9)
oem-sn12  Solaris 2.5: cmsn12or (Oracle 7.3.4)
oem-sn13  Solaris 2.6: cms13syb (Sybase 11.9.2)

*** Notes: ***
 * To do level extracts, use the node: tcaix05 (normal directories)
```

```
 * Use always "noNetLS" builds.
 * Login as root in the native system to perform the preparation of the
   tar images (in order for the chgrp to work properly).

*** Convention: ***
  "-"  next activity to be performed
  "/"  activity started
  "+"  activity completed
```

APPENDIX C. CMVC GUI FOR UNIX, BUILD AND PACKAGE CHECKLIST

The build for the GUI will be for Standalone, X11R5.

Figure 12. CMVC GUI for UNIX, Build and Package Checklist, Level: _____

Activity	**AIX** (_____)	**HP-UX** (_____)	**Solaris** (_____)
logon as build			
change to target directory			
delete target build directory			
extract full latest committed level			
extract delta latest uncommitted level			
chmod 755 qmake			
update qmake for BETA or normal, x11r5			
qmake clean			
qmake			
su - root			
cd $HOMEBUILD/cmvc231/installp			
make -f Makefile.PLATFORM.client			
test code			

APPENDIX D. CMVC CLIENT/SERVER UNIX, BUILD AND PACKAGE CHECKLIST

Figure 13 (Page 1 of 2). CMVC Client/Server UNIX, Build and Package Checklist, Level: ____

Activity	**oracle** (_____) **all/server**	**informix** (_____) **all/server**	**db2v5** (_____) **all/server**	**sybase** (_____) **all/server**
logon as build				
change to target directory				
delete target build directory				
extract full latest committed level				
extract delta latest uncommitted level				
chmod 755 qmake				
update qmake and specify build options				
qmake clean				
qmake				
qmake ship				
-- prepare installp (obj) images for AIX --				
cd $HOMEBUILD/cmvc231/installp				
prepare installp images: bld.all.images4				

Figure 13 (Page 2 of 2). CMVC Client/Server UNIX, Build and Package Checklist, Level: ____

Activity	**oracle** (_____) **all/server**	**informix** (_____) **all/server**	**db2v5** (_____) **all/server**	**sybase** (_____) **all/server**
-- prepare tar images for others --				
su - root (ksh, set -o vi)				
cd $HOMEBUILD/cmvc231/installp				
make -f Makefile.PLATFORM.DATABASE				
make -f Makefile.PLATFORM.client				
perform quick sanity test				
perform FVT test				
prepare backup tapes with install images				
place install images in external ftp site				

APPENDIX E. BIBLIOGRAPHY

For more information on how to use CMVC, you can consult the following manuals and publications:

```
SC09-1596-01  IBM CMVC Client Installation and Configuration
SC09-1597-01  IBM CMVC User's Reference
SC09-1631-02  IBM CMVC Server Administration and Installation
SC09-1633-00  IBM CMVC Concepts
SC09-1635-01  IBM CMVC Commands Reference
```

HOW TO GET ELECTRONIC COPIES OF MANUALS AND TECHNICAL REPORTS

Many of the manuals and technical reports mentioned in this document can be downloaded as follows:

- From the Internet (open to everyone).

Internet

Web Home Page

Not available.

FTP

You can download the PDF version of the manuals from our external FTP site, by doing:

1. ftp ftp.software.ibm.com
2. login as 'anonymous' and for password give your email address.
3. cd ps/products/cmvc/doc/pdf
4. prompt
5. binary
6. mget *.pdf
7. quit

APPENDIX F. 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, DB2, CMVC, TeamConnection	IBM Corporation
UNIX	X/Open Co., Ltd.
NetLS	Gradient Technologies, Inc.
Motif, DFS	Open Software Foundation, Inc.
INFORMIX	Informix Inc.
ORACLE	Oracle Corp.
SYBASE	Sybase Inc.
NFS, SunOS, Solaris Sparc, SPARCstation	Sun Microsystems Inc.
HP-UX, SoftBench	Hewlett-Packard Company
Microsoft, Windows,	Microsoft Corporation
AFS	Transarc Corp

END OF DOCUMENT