How to build and package the CMVC client for OS/2

Document Number TR 29.2245

Angel Rivera

TeamConnection/CMVC Development
IBM Software Solutions
Research Triangle Park, North Carolina
Copyright (C) 1997 IBM

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 Client for OS/2.

ITIRC KEYWORDS

CMVC

Client for OS/2

Building

Packaging

ABOUT THE AUTHOR

ANGEL RIVERA

Mr. Rivera is a Staff Programmer and team lead for the CMVC development. 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

ABSTRACT . . . iii
ITIRC KEYWORDS . . . iii

ABOUT THE AUTHOR . . . v
Angel Rivera . . . v

Figures . . . ix

Building and packaging the CMVC Client for OS/2 . . . 3
Disclaimer . . . 3
Bibliography . . . 3

Overall process with CMVC . . . 5
Families, Components and Releases . . . 5
Process model . . . 6
Project lead creates components and releases . . . 7
Handling defects and features . . . 7
Developers work with files using tracks (Code and Unit Testing) . . . 7
Preparing the code for Function Verification Test (FVT) . . . 8
Backing out a faulty FVT driver . . . 9
Dealing with Tracks/Features during FVT . . . 10
Dealing with defects created during FVT . . . 10
Severities for defects . . . 10

Overall prerequisites . . . 11
Hardware requirements . . . 11
Software requirements . . . 11
Configuration of the hard disks . . . 12
One-time only activities . . . 13
Export the H: drive . . . 13
Install requested OS2DBCSB package from DTCP at YMTVM1 . . . 13
Install requested TCPH32 package from OS2TOOLS . . . 13
Creation of the Build/Ship/Package tree . . . 14
Copy the files from Software Installer . . . 14
Estimated time for building and packaging the CMVC client for OS/2 . . . 14

Building the CMVC client . . . 15
Build/Ship/Package directory structure . . . 15

Location of the source code . . . 15
Activities to be done per each version, release or . . . 16
Files that need to be modified due to changes in Copyright or VRM . . . 16
How to create a bitmap to be used by the Software . . . 17
Files that need to be modified when adding new files . . . 18
Preparation for the Build . . . 18
Process Tracks, Levels and Level Members . . . 18
Extraction of the files related to the installation . . . 19
Cleanup of the Build tree . . . 19
Cleanup of the object modules to force an entire build . . . 20
Extraction of the source files . . . 20
Building the code . . . 20
Building the Message Catalog . . . 20
Notes on things that look bad but are not . . . 20
Known building problems . . . 21
Starting the build . . . 21
Performing a quick sanity test . . . 21

Packaging the CMVC line commands and GUI . . . 23
Miscellaneous notes on Software Installer . . . 23
Overall packaging process . . . 23
Moving files into the Ship directory . . . 24
Compress the files to be shipped . . . 24
Prepare CMVCOS2 installation files . . . 25
Minor changes (no changes to Copyright information) . . . 25
Major changes such as changes to Copyright information . . . 26
Copy files into diskettes . . . 26
Perform installation test from diskettes. . . . 27
Make a backup copy of the installable diskette images . . . 27

Post-Build activities . . . 29
Establish a Level as a baseline (and include the *.map files) . . . 29

Copyrights, Trademarks and Service Marks . . . 31

Appendix A. Build and package checklist . . . 35

FIGURES

1. CMVC Client for OS/2, Build and Package Checklist 36

BUILDING AND PACKAGING THE CMVC CLIENT FOR OS/2

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 Client for OS/2. Of course, we use CMVC itself to develop and maintain CMVC.

A quick status checklist with a summary of the build and package activites is provided in the appendix, Appendix A, “Build and package checklist” on page 35.

In response to requests from several CMVC customers for real examples of how to use CMVC, this technical report was converted from an internal process document and it is made available as a reference material to the CMVC users. It does not contain confidential data.

DISCLAIMER

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.

BIBLIOGRAPHY

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

SC 9-1596- 1 IBM CMVC Client Installation and Configuration
SC 9-1597- 1 IBM CMVC User's Reference
SC 9-1631- 2 IBM CMVC Server Administration and Installation
SC 9-1633- IBM CMVC Concepts
SC 9-1635- 1 IBM CMVC Commands Reference

OVERALL PROCESS WITH CMVC

CMVC is used to manage the software configuration and the file changes for the CMVC Client for OS/2. This section describes the overall process for handling the files and the different CMVC objects associated with the CMVC Client for OS/2.

FAMILIES, COMPONENTS AND RELEASES

In CMVC 2.2, the CMVC client for OS/2 was developed and consisted of line commands and a graphical user interface (GUI).

Then, in CMVC 2.3, after the CMVC Client for Windows 16-bit was developed, we received very good feedback on the usability model that was used in this GUI for Windows and we based the CMVC GUI for OS/2 in this model, which is different than the CMVC GUI for OS/2, version 2.2. In fact, the CMVC GUI for OS/2 and the GUI for OS/2 for the product TeamConnection (the successor for CMVC) uses the same source code but different conditional compilation options.

The CMVC 2.3 product is developed and maintained in the "toro" family; the TeamConnection product is developed and maintained in the "octo" family. Thus, we have the code for the CMVC line commands in the "toro" family and the code for the CMVC GUI in the "octo" family.

The component for the line commands in the toro family is called cmvcPCclient and the release is Pcclient2.3 .

The component for the tools to install the client in the toro family is called cmvcOS2inst and the release is OS2inst2.3 .

The component for the GUI in the octo family is called gui and the release is gui2.3nls .

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 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 GUI and there were frequent code changes, and thus, we could continue at slower pace of change with the line commands.

In fact, the initial changes to the GUI required frequent CMVC Levels to integrate all the tracks for the file changes, whereas, there were fewer levels for the changes in the line commands and the installation tools. Also, once the actual code is frozen, we needed to fine tune the installation tools and the READ.ME files, and having different releases gave us the flexibility to do this.

For business reasons, the CMVC version/release 2.3.0 was the only one that we did, and we just provide full refreshes of the release; that is, the latest refresh supersedes the previous one, and there are no deltas to be delivered and applied. We identify these refreses by using the CMVC Level as the 4th identifier from left to right; for example, our latest refresh for the code was Level 18 or refresh 2.3.0.18 . Thus, we did not encounter the situation where we had to create a new release (such as one for version 2.4 or 3.1) and we did not have to maintain the links of files between multiple releases.

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 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 rather have many small levels that 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, and level. Depending on the stage of the release, it might be necessary to add the approval subprocess.

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. Once the defects are accepted, they will be in 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, link-edit, test, debug and check-in the files affected by the release.

   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/link), do not need to be checked-out: it is enough just to extract them.

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.

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 9.

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. ;p. 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 that the prerequisite track is not ready to be integrated).
8. If starting a build from scratch, then the builder will extract the Full Level 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.

Backing out a faulty FVT driver

If during the integration activities of the preparation for an FVT driver, a problem is found, it could be necessary to make some file changes and prepare again the FVT driver to ensure that it compiles and links fine. 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 8.

DEALING WITH TRACKS/FEATURES DURING FVT

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

   In turn, this event moves the defect or feature to the "verify" state.

2. 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 is modified, the defect or feature moves into the "complete" state.

3. 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.

OVERALL PREREQUISITES

HARDWARE REQUIREMENTS

The hardware requirements are:

At least 16 MB of memory.

At least 150 MB of DASD:

- The DASD for the software requirements is: 110 MB
- The DASD for the source, object, executables, compressed images, etc... is: 40 MB

SOFTWARE REQUIREMENTS

The software requirements are:

OS/2 2.1, or later (takes 30 MB), with REXX.

TCP/IP for OS/2 2.0, or later (takes 10 MB), with NFS server and client.

Include the Programmer's Toolkit.

TCP/IP for OS/2 1.2.1 (takes 4 MB)

Install only the Programmer's Toolkit. Need to use ICAT from diskette B-1 in order to install it. DO NOT ALLOW ICAT TO CHANGE THE CONFIG.SYS TO UPDATE THE SET INCLUDE AND SET LIB (the build procedure will setup the proper values).

Used only during the build task

To be used only for building this client, but not to use it

The CMVC Client/2 was originally built for TCP/IP for OS/2, V1.2.1 and the TCPH32 package, and not for TCP/IP V2.0. However, the TCP/IP V2.0 can be used to run the client, once is built.

IBM Developer's Toolkit for OS/2 2.1 or later (takes 23 MB)

IBM C Set++ 2.0 (C/C++ Tools) or later (takes 29 MB)

Double Byte Character Set (DBCS) routines for OS/2 from Japan (takes 1 MB, using the OS2DBCSB package).

Create the DBCS directory in the proper drive.

Just copy the files from disk 1 into the DBCS directory.

Add TCPH32 package from OS2TOOLS.

Software Installer for OS/2 (takes 4 MB)

If the application uses Software Installer (both Windows or OS2), then the application is automatically enabled for a CID installation. The users have to configure their distribution manager to initiate the install.

CMVC Client OS/2 (takes 6 MB)

CMVC Client DOS/Windows (takes 3 MB)

This client could be used if the CMVC Client OS/2 cannot be used.

CUAPAINT package from the OS2TOOLS conferencing disk (IBM internal)

SAVEDSKF and LOADDSKF tools from the OS2TOOLS conferencing disk.

CONFIGURATION OF THE HARD DISKS

The current build host (hostname: angel.raleigh.ibm.com) has Boot Manager and has the following configuration:

1. Boot Manager

   This host is used also to boot on Windows 3.1 in order to build and package the CMVC Client for Windows.

2. Primary Partition (Drive C, using FAT) with:

   Software Installer for OS/2.

   OS2DBCSB package, with Double Byte Character Set routines for OS/2.

3. Extended Partition with the following drive assignments:

   a. Drive D (FAT) with:

      OS/2 itself.

      TCP/IP for OS/2, with NFS server and client.

      TCPH32 package

      CMVC Client OS/2 is installed here.

   b. Drive F (HPFS) with the development tools.

      C/C++ Tools

   c. Drive H (FAT) with:

      This is the drive used for keeping the source and the executable files for the client.

ONE-TIME ONLY ACTIVITIES

There are some activities that need to be done only once, and there is no need to repeat them for every build.

Export the H: drive

In order to perform an NFS extract, it is important to export the H: drive as follows:

1. Edit the file D:\TCPIP\ETC\EXPORTS
2. Add the following line to export the E: drive to the host carcps21.raleigh.ibm.com where the CMVC families reside:

   e:\ carcps21.raleigh.ibm.com
3. Save the file.

Install requested OS2DBCSB package from DTCP at YMTVM1

Due to the DBCS enablement routines used with the CMVC Client OS/2, it is necessary to build the code with the DBCS routines for OS/2 that are available from Japan. To request this package, issue the following command:

```
REQUEST OS2DBCSB FROM DTCO AT YMTVM1
```

Install the code (2 diskettes) in one flat directory called C:\DBCS

Install requested TCPH32 package from OS2TOOLS

Because the original CMVC Client OS/2 V2.2 code was based on TCP/IP 1.2, it is necessary to build the code with the TCPH32 package from OS2TOOLS.

1. Unpack the TCPH32 code into a single diskette.
2. Copy the code from the diskette into the directory where TCP/IP 1.2.1 is installed, such as D:\TCPIP121.
3. Do not update the SET INCLUDE of the CONFIG.SYS to include TCPIP121. The makecmvc.cmd will do it.

Creation of the Build/Ship/Package tree

To create the Build/Ship/Package tree (to be called just "Build tree"), do the following:

1. Boot the workstation in OS/2.
2. Change to the directory H:\.
3. Use CMVC to extract the file "mk_dir.cmd" from release "OS2inst2.3".
4. Execute the command file, and the tree will be created.

Copy the files from Software Installer

Copy all the necessary files from the Software Installer directory (by default is C:\IBBOS2, but in the build host is called C:INSTOS2) into the Package directory. Do the following:

1. Open the Software Installer folder.
2. Double-click on the icon "Rename Software Installer".
3. Specify "CMV" as the 3 letter prefix.
4. Specify H:\CMVCOS2\PACKAGE as the target directory.
5. Press Enter to start the copying process.

ESTIMATED TIME FOR BUILDING AND PACKAGING THE CMVC CLIENT FOR OS/2

The approximate time needed to perform a complete build of the line commands, of the GUI and the packaging of both is shown below. This time assumes that this is not the first build, 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 for line commands:	.5
Extract the installation/source files for the GUI:	.5
Perform the build for the line commands:	.5
Perform the build for the GUI:	.5
Package the files:	.5
Perform the installation test:	.5

TOTAL	3.

BUILDING THE CMVC CLIENT

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

BUILD/SHIP/PACKAGE DIRECTORY STRUCTURE

The Build/Ship/Package directory structure is as follows:

```
<top_build_directory>
      compress              shippable files (compressed)
      gui                   source code for the GUI
      maps                   contains link maps
      os2                   contains the executables for OS/2 line commands
      package               used for Software Installer
      samples               samples using line commands
      savedskf             used for CMVCOS2 PACKAGE for CMVCBETA
      ship                  shippable files (no compressed yet)
     shipstuf              contains  read.me
      sloc                  count of lines of code
      src                   source code for the line commands
```

Note: The <top_build_directory> used in this document is H:\CMVCOS2.

LOCATION OF THE SOURCE CODE

The source code for the CMVC line commands for OS/2, Version 2.3, is in the "toro" family:

```
family:      toro
release:     PCclient2.3 -> executables for line commands
release:     OS2inst2.3   -> build and packaging tools
```

The source code for the CMVC GUI for OS/2, Version 2.3, is in the "octo" family:

```
family:      octo
release:     gui2.3nls         -> executable for GUI
```

The ID team members are responsible for the maintenance of the Help file "cmvc.hlp" which is part of the release "gui2.3" in the "octo" family.

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 these service information.

Release "PCclient2.3"

The following files must be changed in release "PCclient2.3" (the actual code) in order to update the Copyright information and the Title:

*.c and *.h

h:\cmvcos2\src\inc\whatcmd.h
h:\cmvcos2\src\cat\cmvc.msg

These are the files that use the "@(#)" in static char

*.def

h:\cmvcos2\src\ui\simple\cmd\cmvc.def

The change is in the DESCRIPTION field; be careful in not to exceed the character limit; also, all needs to be specified in one single line.

Release "gui2.3nls"

The following files must be changed in release "gui2.3nls" (the actual code) in order to update the Copyright information and the Title:

*.c and *.h

h:\cmvcos2\gui\src\inc\whatcmd.h

These are the files that use the "@(#)" in static char

*.def

h:\cmvcos2\gui\src\tcmerge\cmvccomp.def
h:\cmvcos2\gui\src\ui\cmvc.def

The change is in the DESCRIPTION field; be careful in not to exceed the character limit; also, all needs to be specified in one single line.

*.dlg

h:\cmvcos2\gui\src\ui\fhoabout.dlg

This file contains the First Panel that appears with the Copyright, which is the same panel for "Help -> Product Information".

Release "OS2inst2.3"

The following files must be changed in release "OS2inst2.3" (the files for packaging) in order to update the Copyright information and the Title:

h:\cmvcos2\shipstuf\cm23read.scr --> read.me

The ID team members are responsible for this file. The output file must be renamed "READ.ME" when downloaded into the workstation.

h:\cmvcos2\package\cmos223.bmp

See next section on how to prepare this file. This file is imbedded in the IIRC.RC file.

h:\cmvcos2\package\copyrite.txt

Note: Do not use the T editor for this file. Use the E editor instead (provided by OS/2). The T editor does not handle very well the End-Of-File and Software Installer displays it as an ugly fat vertical character.

h:\cmvcos2\package\cmvcos2.icf

Look for term "The service level is".

h:\cmvcos2\package\cmvcos2.pkg

Look for keyword "SERVICELEVEL"; also update the date starting from entry for "CMVCOS2.ICF".

h:\cmvcos2\package\iirc.rc

Look for keyword "INST_WINDOW_TITLETEXT".

How to create a bitmap to be used by the Software

Installer

The existing CMVC 2.2 icon was used for CMVC, and thus, there was no need to modify it.

To update the version-release-modification of the bitmap that will be used with Software

Installer, you can use CUAPAINT (which can be requested from the tools disk OS2TOOLS as package CUAPAINT) as follows:

1. Use one of the existing bitmaps (such as package\cmos223.bmp) and edit it by using CUAPAINT.
2. Use the arrow tool to select the image that needs to be processed and clipped.
3. Use the proper text styles, Font Style:

   Top text: Courier, Bold Italics, Size 14 (size 16)

   Bottom text: Courier, Normal, Size 8
4. Select color black, 8th position in first row from color tool.
5. Save the file in the package directory and perform a checkin to store the file in CMVC.

Files that need to be modified when adding new files

This section is applicable when adding new files (or deleting obsolete files) that are related to the CMVC OS/2 client, release "OS2inst2.3":

If the new file is related to the source code and does not affect the overall packaging scheme, then modify the following files:

- The appropriate make file to ensure proper compilation.

If the new file affects the packaging scheme, then modify the following files:

- h:\cmvcos2\cp_ship.cmd
- h:\cmvcos2\do_compr.cmd
- h:\cmvcos2\cp_compr.cmd
- h:\cmvcos2\cp_2disk.cmd
- h:\cmvcos2\package\cmvcos2.pkg

PREPARATION FOR THE 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 6.

Extraction of the files related to the installation

process

To extract the source files do the following:

1. Setup the environment variables:

   ```
   set CMVC_FAMILY=toro
   set CMVC_BECOME=build
   ```

2. Enter the following line command:

   ```
   release -extract OS2inst2.3 -node angel -root e:\cmvcos2 -crlf
   ```

You can proceed to section “Extraction of the source files” on page 20 to extract the source files.

Cleanup of the Build tree

The command files used in this section and the next ones are extracted in the previous step, see “Extraction of the files related to the installation.”

Perform the following steps to cleanup the Build tree:

1. Change to the directory H:\CMVCOS2
2. Execute the command **cl_all** to cleanup the source, the object, the output, the ship and the compress directories.
3. There are 3 other command files that perform a subset of the cleaning activities specified in cl_all:

   The command **cl_src** cleanups the source directories.

   In turn, this command invokes the following commands:

   – The command **cl_srcl** does the cleanup for the line commands.

   – The command **cl_srcg** does the cleanup for the GUI.

   The command **cl_obj** cleanups the object/executable directories.

   In turn, this command invokes the following commands:

   – The command **cl_objl** does the cleanup for the line commands.

   – The command **cl_objg** does the cleanup for the GUI.

   The command **cl_misc** cleanups the output, the ship and the compress directories.

We wanted flexibility when deleting files from the build tree; that is the reason for the many cleaning command files.

Cleanup of the object modules to force an entire build

To delete the object and executable (and other type of generated modules) to force an entire build without the need to do a drastic complete cleanup of the build tree, run one of the following commands:

The command **cl_obj** does the clean up of the object and executable directories for both the line commands and the GUI.

The command **cl_objl** does the cleanup for the line commands.

The command **cl_objg** does the cleanup for the GUI.

Extraction of the source files

To extract the source files, execute one of the following commands:

The command **do_extr** extracts the source code for both the line commands and the GUI.

The command **do_extrl** does the extract for the line commands. Use the directory h:\cmvcos2

The command **do_extrg** does the extract for the GUI. Use the directory h:\cmvcos2\gui

This step only copies the necessary files from F:

BUILDING THE CODE

Building the Message Catalog

Unlike the DOS/Windows build process, the message catalog is built as part of the normal build process, and it is done when building the line commands.

Notes on things that look bad but are not

When performing the build, you may encounter some warning or error messages that might scare you (specially if this is your first build); however, the following warning/messages are not harmful to the build process and you should not be alarmed:

LIB0004: Warning (module_name): Unable to locate module

At first sight, it seems that the LIB command was not able to find the module_name.OBJ file and thus, it was not placed inside the library (in src\obj, such as LIBNLS.LIB).

However, this message will always happens when the library is built for the first time and this message indicates that the mentioned module_name does not exist yet in the library (the operation is -+module_name, to delete first and then to insert).

Known building problems

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

None.

Starting the build

1. Ensure that you are in H:\CMVCOS2
2. Open two OS/2 windows. Due to conflicts with environment variables between the build of the line commands and the GUI, it is recommended to have two OS/2 windows, one to build the line commands and the other for the GUI.
3. In one window, build the line commands: **do_bldl**
4. In the other window, build the GUI: **do_bldg**

Performing a quick sanity test

You can copy the executable and the DLLs by running one of the following:

The command **cp_inst** copies the code for both the line commands and the GUI.

The command **cp_instl** does the copy for the line commands.

The command **cp_instg** does the copy for the GUI.

After this step, you could invoke CMVCOS2 and verify that the code (new or fixed) is behaving as expected. If you find a problem now, you could stop at this point and talk to the developers to fix the problem.

PACKAGING THE CMVC LINE COMMANDS AND GUI

This chapter describes how to package the CMVC line commands and GUI.

MISCELLANEOUS NOTES ON SOFTWARE INSTALLER

The following files are created by Software Installer in the OS2\SYSTEM directory of the workstation when installing CMVC OS/2 client. These files contain information on the packages installed in the system and will override new setups; thus, if the user is experiencing problems when installing a new version of the client on top of an existing one and the directory/drive is different, then delete these files:

epfis.ini

epfihcnf.cnf

epficat.pkg

OVERALL PACKAGING PROCESS

Once the code is built and a quick sanity test is performed on it, then it is ready to be packaged. The overall sequence is the following:

1. Execute the overall packaging command: **do_pack.cmd** which does the following:

   Gather the files to be shipped: **cp_ship**

   Compress the files to be shipped: **do_compr** and **cp_compr**

2. Prepare the CMVCOS2 catalog, package and description file for Software Installer for OS/2.

3. Build the installation files for the complete package: **do_bldin**

   Test by running from the cmvcos2\package directory: **cmvidlds**

4. Copy the installation files and the compressed files into diskettes: **cp_2disk**

5. Perform installation test from diskettes.

6. Place installable diskette images in the public places such as the CMVC Development Home Page.

MOVING FILES INTO THE SHIP DIRECTORY

Once the executable for CMVC Client OS/2 are available, do the following steps to copy these executables into a Ship directory together with other files that need to be part of the whole package.

The purpose of the Ship directory is to provide a staging area where all the CMVC OS/2 client files will reside. This provides a good opportunity to obtain the total size of the overall files and this helps in determining how much space the end-user will need (to update the CMVCOS2.PKG and CMVCOS2.ICF information).

1. Perform the command file **cp_ship** which will copy into the h:\cmvcos2\ship directory the following files that were produced by the build just recently done, such as:

   h:\cmvcos2\gui\cmvc.exe
   h:\cmvcos2\gui\cmvcres.dll
   h:\cmvcos2\gui\dcomp.exe
   h:\cmvcos2\os2\access.exe
   h:\cmvcos2\os2\<the rest of the CMVC line commands>

2. ... and the following files that are provided by the CMVC development team but are not built every time:

   h:\cmvcos2\shipstuf\cmvc23.ini
   h:\cmvcos2\shipstuf\cmvc.log
   h:\cmvcos2\shipstuf\cmvcmbs.dll

3. ... and the following Help/Book files that are provided by the CMVC ID team:

   h:\cmvcos2\shipstuf\cmvc.hlp
   h:\cmvcos2\shipstuf\cmvccmd.inf
   h:\cmvcos2\shipstuf\getstart.doc
   h:\cmvcos2\shipstuf\read.me
   h:\cmvcos2\shipstuf\cid.doc

4. From the prompt, list the contents of the SHIP directory to find out the amount of DASD that the code takes:

   dirsize .\ship

   For Version 2.3 is approximately 4.7 MB. This data is important because the files package\cmvcos2.pkg and package\cmvcos2.icf

COMPRESS THE FILES TO BE SHIPPED

The purpose of the Compress directory is to provide a staging area where only the compressed files will reside. This provides a good opportunity to obtain the total size of the overall compressed files and this helps in determining how many diskettes would be needed (add 690K for the Software Installer files in diskette 1).

1. Perform the command file **do_compr** which will compress the files from the Ship directory and store the compressed files in the Compress directory h:\compress.
2. Perform the command file **cp_compr** which will copy the compressed files from the Compress directory into the Package directory h:\package.

PREPARE CMVCOS2 INSTALLATION FILES

Follow the instructions in the Software Installer on-line manual, "Software Installer Reference", Section "Step-by-Step Instructions for Using Software Installer", A brief summary of these steps are shown below.

Minor changes (no changes to Copyright information)

This is used when updating the date and time, etc...

1. Change to the directory H:\CMVCOS2\PACKAGE:
2. Modify the CMVCOS2 installation files:

   package\cmvcos2.icf
   package\cmvcos2.pkg
   package\cmvcos2.dsc

   You should need to verify the following in the CMVCOS2.PKG file:

   a. The DATE and TIME fields are set to the appropriate time.
   b. The SERVICELEVEL field should have the correct service level.
   c. The size of the component should be rounded up from the real value.
   d. If BETA code, then scan the file for the comments with the string "BETA" and uncomment them and comment the ones with the string "GOLD".
   e. If GOLD code, then scan the file for the comments with the string "GOLD" and uncomment them and comment the ones with the string "BETA".

   You should need to verify the following in the CMVCOS2.ICF file:

   a. The VRM field should have the correct version, release and modification values, such as '020300' for V2 R3 M0.

Major changes such as changes to Copyright information

This is used when updating the Copyright information.

1. Change to the directory H:\CMVCOS2\PACKAGE:
2. Modify the CMVCOS2 installation files:

   package\cmvcos2.icf
   package\cmvcos2.pkg
   package\cmvcos2.dsc

   You should need to verify the following in the CMVCOS2.PKG file:

   a. The DATE and TIME fields are set to the appropriate time.

   b. The SERVICELEVEL field should have the correct service level.

   c. The size of the component should be rounded up from the real value.

   d. If BETA code, then scan the file for the comments with the string "BETA" and uncomment them and comment the ones with the string "GOLD".

   e. If GOLD code, then scan the file for the comments with the string "GOLD" and uncomment them and comment the ones with the string "BETA".

   You should need to verify the following in the CMVCOS2.ICF file:

   a. The VRM field should have the correct version, release and modification values, such as '020300' for V2 R3 M0.
3. Modify the IIRC.RC resource file, if adding/changing any of the windows used during installation.
4. Perform **do_bldin** which will do the following:

   Perform BLDRC.CMD to build the resource file and to bind it to EPFIDLDS.EXE which is used during the installation process.

   Perform BLDINST.CMD to prepare all the runtime files from Software Installer to create INSTALL.IN_. It is necessary to specify the COPYRITE.TXT file in order to show it during the installation process.

   BLDINST COPYRITE.TXT
5. Test the initial installation window by executing from the cmvcos2\package directory:

   CMVIDLDS

COPY FILES INTO DISKETTES

1. Prepare 3 diskettes (1.4 MB) as follows:

Diskette Label	Description in Outside Label
CMVCOS21	CMVC Client OS/2, Disk 1 of 3
CMVCOS22	CMVC Client OS/2, Disk 2 of 3
CMVCOS23	CMVC Client OS/2, Disk 3 of 3

You need to use the OS/2 LABEL command to set the internal label of the diskettes.

Note: If you are reusing diskettes that contain the client, delete the contents to avoid contamination of the new stuff with the old stuff.

2. Change the directory to H:\CMVCOS2.
3. Perform the command file **cp_2disk** to copy the files from the package directory into the diskettes.

PERFORM INSTALLATION TEST FROM DISKETTES.

Install the client from diskettes into C:\CMVCOS2.

1. Insert diskette 1 in drive A:. From an OS/2 Window, enter the following:
2. Enter:

   A:\INSTALL
3. Follow the instructions provided by the installation program.

MAKE A BACKUP COPY OF THE INSTALLABLE DISKETTE IMAGES

This section is only applicable if the quality of the driver qualifies to be an official BETA driver.

To make a backup copy of the installable diskette images, do the following:

1. Change the directory to H:\CMVCOS2\SAVEDSKF

   SAVEDSKF is an IBM internal tool that creates an image from a diskette. The counterpart tool, LOADDSKF is used to expand that diskette image into a diskette.
2. Create a subdirectory to store the image; the suggested format for the name is:

   XXXXmmDD
   ++++ ---> Type, such as BETA, GOLD, DRIV
   ++ ---> Month
   ++ ---> Day
3. Perform the following with the diskette 1, which should be inserted in the A: drive.

   savedskf a: h:\cmvcos2\savedskf\SUBDIR\cmvcos21.dsk /n

   where SUBDIR is the name of the subdirectory created in a previous step (above).

4. Perform the following with the diskette 2: which should be inserted in the A: drive.

   ```
   savedskf  a:  h:\cmvcos2\savedskf\SUBDIR\cmvcos22.dsk  /n
   ```

5. Perform the following with the diskette 3: which should be inserted in the A: drive.

   ```
   savedskf  a:  h:\cmvcos2\savedskf\SUBDIR\cmvcos22.dsk  /n
   ```

You can now place these installable images in a public place, such as the CMVC Development Home Page.

POST-BUILD ACTIVITIES

ESTABLISH A LEVEL AS A BASELINE (AND INCLUDE THE *.MAP FILES)

If the installation is successful and the driver is considered to be "shippable" (for internal or external usage), then perform the following post-build activities in order to have a good baseline of code for maintenance and for subsequent development activities:

1. Commit any outstanding levels and ensure that their type is "development".
2. Create a defect to store the maps associated with the build.

   Create a track for that defect on the release.
3. Create a level and ensure that the type is NOT "development"; appropriate types could be "alpha", "beta", "FVT", "SVT" or "gold".
4. Create a directory to store the map files, such as:

   h:\cmvcos2\maps\driv1123

   Month and Date
   Type of drive (DRIV, BETA, GOLD)

   It is important to keep handy at least one back level of the MAP files from the build directories. It is suggested to use the MAPS directory and create subdirectories that include both the type of driver and the date, such as BETA0406 or GOLD0502; then copy the MAP files into that subdirectory.
5. Change the directory to the maps directory.
6. From the maps directory, invoke the following command to store the *.MAP files:

   e:\cmvcos2\cp_maps

   Note: These are the only output files that are stored back in CMVC. The reason behind this exception is that the map files are needed for debugging (in case of abends) and they must correspond to the executables and the DLLs.
7. Complete the fix records, add the track as a level member to the level just created.
8. Process the level to completion.

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
IBM, OS/2, CMVC, Software Installer, C Set++	IBM Corporation
IBM, OS/2, CMVC, Software Installer, TCP/IP for DOS, PC-DOS	IBM Corporation
Microsoft Windows	Microsoft Corporation

APPENDIX A. BUILD AND PACKAGE CHECKLIST

This appendix contains a checklist that provides a quick status of the build and package activities that are described in detail in this document.

To actually use the checklist, you can make a photocopy of it and do the following:

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)
5. Use a minus sign to indicate that the activity is not applicable.

Figure 1. CMVC Client for OS/2, Build and Package Checklist

Activity	**OS/2** **(host: ___)**
(optional) clean all directories: cl_all	
clean OBJ, DLL, EXE: cl_obj	
extract full latest committed level	
extract delta latest uncommitted level	
extract levels	
build the GUI code: do_bldg	
build the CLI code: do_bldl	
copy DLL and EXE: cp_ship	
compress the code: do_compr	
copy the compressed code: cp_compr	
build the installation files: do_bldin	
copy the files into disks: cp_2disk	
perform installation test	
use SAVEDSKF to create disk images	
place installable images in CMVCBETA and LAN	
copy MAPS	

END OF DOCUMENT