How to build and package the CMVC client for Windows 3.1

Document Number TR 29.2244

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 Windows 3.1 (16-bit).

ITIRC KEYWORDS

CMVC

Client for Windows

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 Windows 3.1 . . . 3
Disclaimer . . . 3
Bibliography . . . 3

Overall process with CMVC . . . 5
Families, Components and Releases . . . 5
Process model . . . 6
Project lead creates components and releases . . . 6
Handling defects and features . . . 7
Developers work with files using tracks (Code and Unit Testing) . . . 7
Preparing the code for Function Verification Test (FVT) . . . 7
Backing out a faulty FVT driver . . . 9
Dealing with Tracks/Features during FVT . . . 9
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 . . . 12
Consolidating TCP/IP include files into one directory . . . 12
Creation of the Build/Ship/Package tree . . . 13
Copy the files from Software Installer . . . 13
Estimated time for building and packaging the CMVC Client for . . . 13

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 . . . 15

Files that need to be modified due to changes in 16
How to create a bitmap to be used by the Software Installer 17
Files that need to be modified when adding new files 18
Preparation for the Build 18
Process Tracks, Levels and Level Members 19
Extraction of the files related to the installation 19
Cleanup of the Build tree 19
Extraction of the source files 20
Copy Files into other directories 22
Building the code 22
Considerations before building 23
Build the Message Catalog 24
Start Turbo C++ 24
Known building problems 29
Performing a quick sanity test 29

Packaging the CMVC GUI 31
Miscellaneous notes on Software Installer 31
Overall packaging process 31
Moving files into the Ship directory 32
Compress the files to be shipped 33
Keep back levels of the MAP files 33
Prepare CMVCWIN installation files 33
Minor changes (no changes to Copyright information) 33
Major changes (to Copyright information) 34
Copy files into diskettes 35
Perform installation test from diskettes. 36
Make a backup copy of the installable diskette images 36

Post-Build activities 37
Establish level as baseline with *.map files 37

Copyrights, Trademarks and Service Marks 39

Appendix A. Build and package checklist 43

FIGURES

1. CMVC Clients for Windows, Build and Package Checklist 44

BUILDING AND PACKAGING THE CMVC CLIENT FOR WINDOWS 3.1

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 Windows 3.1 (16-bit). 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 43.

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 Windows 3.1. This section describes the overall process for handling the files and the different CMVC objects associated with the CMVC Client for Windows 3.1.

FAMILIES, COMPONENTS AND RELEASES

In CMVC 2.3 the CMVC Client for Windows 3.1 (16-bit) was developed and it had a different look and feel with respect to the GUI for UNIX. 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.

The CMVC Client for Windows is developed and maintained in the "octo" family.

The component for the GUI in the octo family is called gui and the release is windows2.3 .

The component for the tools to install the client in the octo family is called installWindows and the release is windowsInst2.3 .

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 two separate releases, it might be possible to use only one release for all the code (GUI and installation tools), which could minimize the overhead associated with tracks and levels. However, we found that having two releases gives us also much needed flexibility for handling situations where we could fine tune the installation tools at a different pace than the GUI code.

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 4 MB of memory.

At least 100 MB of DASD:

- The DASD for the software requirements is: 55 MB
- The DASD for the source, object, executables, compressed images, etc. is: 45 MB

SOFTWARE REQUIREMENTS

Note: The build procedures described in later sections assume that all the software has been installed in the C: drive.

The software requirements are:

PC-DOS 6.0, or later (takes 3 MB)

Windows 3.1, or later (takes 9 MB)

TCP/IP for DOS 2.1, or later (takes 15 MB)

Include the Programmer's Toolkit.

Install it in its default directory: c:\TCPDOS

Borland Turbo C++ 3.0 or later (takes 22 MB)

Install it in its default directory: c:\TCWIN

After installing it, to create the additional icons for tools, run:

```
c:\tcwin\bin\groups groups.b$$
```

Software Installer for Windows 1.2 or later (takes 3 MB)

Install it in the directory: c:\INSTWIN. Do not use the default directory IBBWIN.

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 (Netview DM, etc.) to initiate the install.

CMVC Client for Windows (takes 3 MB)

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
2. Primary Partition (Drive C, using FAT) with all the software requisites.

   DOS

   Windows 3.1

   TCP/IP for DOS

   Borland Turbo C++ 3.

   Software Installer for Windows

   CMVC Client for Windows.
3. Extended Partition with the following drive assignments:

   a. Drive D: OS/2 2.1 in FAT.

      CMVC Client OS/2 is installed here, and it is used to do the initial download of the files for the Client for Windows. Drive E: Data in FAT.

   b. Drive H: Source code for CMVC Windows, in FAT.

      This is the drive used for keeping the source and the executable files for the client.
4. In your CONFIG.SYS the LASTDRIVE variable for NFS is Z.

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.

Consolidating TCP/IP include files into one directory

Due to the space limitations in Turbo C++ in the field that specifies the paths for the include files in the project file, do the following (only once, you do not need to repeated every time there is a build):

1. Copy the TCP/IP include files for WSA into RSA:

   ```
   copy c:\tcpdos\include\wsa\ .h c:\tcpdos\include\rsa
   ```

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 DOS.
2. Change to the directory h:\.
3. Use CMVC to extract the file "mk_dir.bat" from release "windowsInst2.3".
4. Execute the batch 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:\IBBWIN, but in the build host is called C:INSTWIN) into the Package directory. Do the following:

1. Change to the directory h:\.
2. Use CMVC to extract the file "cp_swins.bat" from release "windowsInst2.3".
3. Execute the batch file, and the files will be copied.

ESTIMATED TIME FOR BUILDING AND PACKAGING THE CMVC CLIENT FOR

Windows

The approximate time needed to perform a complete build of the GUI and the packaging 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:	.5
Perform the build:	.5
Package the files:	.5
Perform the installation test:	.5

TOTAL	2.

BUILDING THE CMVC CLIENT

This chapter describes how to build the CMVC 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)
maps	contains maps of the executables and DLLs
output	contains object files, DLLs and EXE
package	used for Software Installer
savedskf	used for CMVCWIN PACKAGE for CMVCBETA
ship	shippable files (no compressed yet)
shipstuf	contains read.me
sloc	count of lines of code
src	source code

Note: The <top_build_directory> used in this document is h:\CMVCWIN.

LOCATION OF THE SOURCE CODE

The source code for Version 2.3 is in the "octo" family:

release: windows2.3 -> executables and DLLs
release: windowsInst2.3 -> build and packaging tools

The ID team members are responsible for the maintenance of the Help file "src\ui\fhwz.hlp" which is part of the release "windows2.3".

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 "windows2.3"

The following files must be changed in release "windows2.3" (the actual code) in order to update the Copyright information and the Title:

src\ui\fhwzmain.c

This is the only file that uses the "@(#)" in static char.

src\ui\fhwz.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.

src\ui\fhwzmri.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.

src\ui\fhwzdlg.dlg

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

Look for the keyword "Level".

src\cmd\fhwzcmd.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.

src\cmd\cmvc.msg

This file contains the message "msgTestCatalog" which contains the VRML 2.3.0.x.

Release "windowsInst2.3"

The following files must be changed in release "windowsInst2.3" (the files for packaging) in order to update the Copyright information and the Title:

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.

package\cmwin23.bmp

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

package\copyrite.txt

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

package\cmvcwin.icf

Look for the term "service level is".

package\cmvcwin.pkg

Look for the keyword "SERVICELEVEL" and change the date starting from the entry for "CMVCWIN.iCF".

package\epfwiirc.rc

Look for the keyword "INST_WINDOW_TITLETEXT".

How to create a bitmap to be used by the Software Installer

The following activities need to be done from Win-OS/2, because the CLIP tool is an OS/2 tool:

1. Use one of the existing bitmaps (such as package/cmwin23.bmp) and edit it using Paint Brush from Windows.
2. The Text Fonts are:

   Top portion: Times New Roman, Bold Italics, Size 16 (Size 18)

   Bottom portion: Times New Roman, Normal, Size 8
3. Use the lazo tool in Pain Brush to select the image that needs to be processed and clipped.

You need process the image with CLIP via the clipboard, otherwise the default image created by Paint Brush is too large.

4. Copy the desired portion of the image into the clipboard.

   Try to select the minimum rectangle that includes all the text.

5. Invoke the CLIP tool from OS/2.
6. The clipboard contents will be pasted automatically into CLIP.

   This will create a clipped version of the original image.

   Specify a "PM Bit Map" with 16 colors when dealing with this image.

7. Exit CLIP, and by doing this, the clipboard image will be clipped.
8. Now from Paint Brush, create a new file and paste the contents of the clipboard.

   Notice that at this time, the image is smaller.

9. Save the file as h:\cmvcwin\package\cmwin23.bmp 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 Client for Windows, release "windows2.3":

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 project file (.PRJ) to ensure proper compilation.

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

- cp_ship.bat
- do_compr.bat
- cp_compr.bat
- cp_2disk.bat

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

You can use a working version of the CMVC Client for Windows or the CMVC Client for OS/2 (use the Boot Manager feature to start in OS/2) to extract the source files as follows:

1. Start the CMVC client with the family **octo**.
2. Bring up the Files window and select the files that you want to extract from the release **windowsInst2.3.**.
3. One usual task is to extract the current versions of the files that were modified recently. One query that is helpful in this task is:

   releaseName in ('windowsInst2.3') order by lastUpdate DESC

   The specification of DESC is important because the files that have changed recently will appear at the top, and because of the limitations of the CMVC Client OS/2 (which only shows the first 120 files!) the files that you are looking for are at the top of the list.

   It is recommended to add a task in the task list with this query, because it could be a frequent task.
4. From the list of files, select the ones to be extracted.
5. Select the action "extract" and specify:

   The directory: h:\cmvcwin.

   Select OK to start the extract of the files.

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

Cleanup of the Build tree

The batch files used from now on 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:\.

2. Execute the batch command **cl_all** to cleanup the source, the output, the ship and the compress directories.
3. There are 2 other batch files that perform a subset of the cleaning activities specified in cl_all:

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

   The batch 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.

Extraction of the source files

Extraction using Release Extract

You have to use the CMVC Client OS/2 (use the Boot Manager feature to start in OS/2) to extract the source files as follows:

1. Start the CMVC client with the family **octo**.
2. Bring up the Release window and select the release **windows2.3**.
3. Select the action "extract" and specify:

   Extract committed versions.

   The name and directory of the NFS server in an OS/2 server (There is no NFS server in IBM TCP/IP for DOS).

   host: angel
   directory: h:\cmvcwin

   Select OK to start the extract of the files for the release.

You can proceed to section “Copy Files into other directories” on page 22 to copy some files into other directories.

Extraction using Level Extract

You have to use the CMVC Client OS/2 (use the Boot Manager feature to start in OS/2) to extract the source files as follows:

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

2. Bring up the Level window and select the level that you want to extract from the release **windows2.3**.
3. Select the action "extract" and specify:

   Extract committed versions.

   The name and directory of the NFS server in an OS/2 server (There is no NFS server in IBM TCP/IP for DOS).

   ```
   host:      angel
   directory: h:\cmvcwin
   ```

   Select OK to start the extract of the files for the release.

You can proceed to section “Copy Files into other directories” on page 22 to copy some files into other directories.

Extraction using File Extract

You can use a working version of the CMVC Client for Windows or the CMVC Client for OS/2 (use the Boot Manager feature to start in OS/2) to extract the source files as follows:

1. Start the CMVC client with the family **octo**.
2. Bring up the Files window and select the files that you want to extract from the release **windows2.3**.
3. One usual task is to extract the current versions of the files that were modified recently. One query that is helpful in this task is:

   ```
   releaseName in ('windows2.3') order by lastUpdate DESC
   ```

   The specification of DESC is important because the files that have changed recently will appear at the top, and because of the limitations of the CMVC Client OS/2 (which only shows the first 120 files!) the files that you are looking for are at the top of the list.

   It is recommended to add a task in the task list with this query, because it could be a frequent task.
4. From the list of files, select the ones to be extracted.
5. Select the action "extract" and specify:

   The directory: h:\cmvcwin.

   Select OK to start the extract of the files.

You can proceed to section “Copy Files into other directories” on page 22 to copy some files into other directories.

Copy Files into other directories

1. When the release extract ends, copy the following files which need to be in different directories, due to the limitations in Turbo C++ which cannot look into another source directory to find out the rest of the source files. You can perform the batch command file **cp_files.bat**.

   Note: This exec also will delete obsolete project files.

BUILDING THE CODE

To build the CMVC Client for Windows do the following steps which are described in greater detail later on in this chapter:

1. take into account the type of driver that is being built, such as beta.
2. copy files into other directories: **cp_files**
3. create the message catalog "cmvc.cat": **mk_cat**
4. use Turbo C++ to compile the code.
5. build the following cmvcwin\src\cmd project files:
   a. ni.prj
   b. nls.prj
   c. client.prj
   d. fhwzcmd.prj
6. build the following cmvcwin\src\ui project files:
   a. fhwzmri.prj
   b. fhwzmain.prj

      You need to compile first the file: fhwmcst.c

Notes:

1. Before you start the build process, you need to boot your workstation under DOS and then change the directory to h:\CMVCWIN.
2. Disregard the warning "Can't create h:\cmvcwin\src*.prj" when opening a new project. This warning happens because the project files have been extracted from CMVC as read-only files.

Considerations before building

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

Making a BETA driver

Making a Development driver

Making a GOLD driver

Making a BETA driver (BETARELEASE code)

To provide a feature that will not allow the beta code to work after a certain date (currently is July-1-94), it is necessary to compile the FHWZMAIN.PRJ with a define that will tag it as Beta code.

To do this, bring the FHWZMAIN.PRJ in Turbo C++ (as described in section “Building the rest of the project files in \src\ui” on page 28) and select the "Options" pulldown, choose "Code generation ...", and then add to the "Defines" field at the bottom of the dialog "Code Generation Options" the following (note the semicolon before the word):

```
;BETARELEASE
```

The code thus compiled will have the word BETA added to the title of the main window.

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

Making a Development driver (with ASSERT)

To compile with the ASSERT function activated, it is necessary to compile the FHWZMAIN.PRJ with a define that will tag it as Developer's code.

To do this, bring the FHWZMAIN.PRJ in Turbo C++ (as described in section “Building the rest of the project files in \src\ui” on page 28) and select the "Options" pulldown, choose "Code generation ...", and then add to the "Defines" field at the bottom of the dialog "Code Generation Options" the following (note the semicolon before the word):

```
;DEVELOPMENT
```

The code thus compiled will ...

Making the Gold code

To compile for the GOLD level of code, it is necessary to compile without the BETA and the ASSERT considerations.

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

Build the Message Catalog

1. From the DOS prompt, perform the batch command file **mk_cat.bat**.

   The command "mkcat" creates the cmvc_m.h file, and the command "gencat" creates the cmvc.cat message catalog.

   Note: This batch file deletes the previous cmvc.cat and cmvc.dat files to avoid the situation where the gencat command will fail if any of these files are present (run time R601 - null pointer assignment).

Start Turbo C++

1. Start Windows. From the DOS prompt, enter:

   win

2. From the "Program Manager", open the "Turbo C++ 3.1" group.
3. From the "Turbo C++ 3.1" group select the icon for "Turbo C++". Double click on it to start Turbo C++.

Building the Project file \src\cmd\ni.prj

This sequence of steps is used also for each of the project files that are listed in “Building the rest of the project files in \src\cmd” on page 25.

1. Select the "Project" pulldown and then select "Open Project...".
2. From the "Open Project File" select the project file (see below) to be compiled.

   a. Leave the File Name as "*.prj".

   b. Change the path to:

      h:\cmvcwin\src\cmd

   c. Then you will see the following project files:

      ni.prj
      nls.prj
      client.prj
      fhwzcmd.prj

d. Select the project file "ni.prj" and then click on OK.

   You will see a window called "Project: xxxxx" where xxxxx is the name of the project file. This window lists all the files that are part of the project file.

3. Change the specifications of the directories for the project file. Select the "Options" pulldown and choose "Directories..."

4. From the "Directories" dialog, ensure that the directories are as follows.

   Note: Due to lack of space, this document shows each directory in a separate line, but actually, the entries for each category are concatenated into a single line.

   Include Directories:
   c:\tcwin\include;
   c:\tcwin\owl\include;
   h:\cmvcwin\src\cmd\inc;
   c:\tcpdos\include;
   c:\tcpdos\include\rsa;

   Library Directories:
   c:\tcwin\lib;
   c:\tcwin\owl\lib;
   h:\cmvcwin\output;

   Output Directory:
   h:\cmvcwin\output;

   You may have to use the combo box to see the additional entries in order to select the proper one.

   When ready, click on the "OK" button.

5. Select the "Compile" pulldown and choose "Make"

   The status of the compilation will be shown in a window called "Compile Status". The "Status" must be "Success".

Building the rest of the project files in \src\cmd

Perform the sequence of steps described in “Building the Project file \src\cmd\ni.prj” on page 24 for each one of the following project files in cmvcwin\src\cmd:

Note: Disregard the warning "Can't create h:\cmvcwin\src*.prj" when opening a new project. This warning happens because the project files have been extracted from CMVC as read-only files.

1. nls.prj

   The "Directories" for this project file must be:

Include Directories:
  c:\tcwin\include;
  c:\tcwin\owl\include;
  h:\cmvcwin\src\cmd\inc;
  c:\tcpdos\include;
  c:\tcpdos\include\rsa;

Library Directories:
  c:\tcwin\lib;
  c:\tcwin\owl\lib;
  h:\cmvcwin\output;

Output Directory:
  h:\cmvcwin\output;

2. client.prj

   The "Directories" for this project file must be:

   Include Directories:
     c:\tcwin\include;
     c:\tcwin\owl\include;
     h:\cmvcwin\src\cmd\inc;
     h:\cmvcwin\src\ui;
     c:\tcpdos\include;
     c:\tcpdos\include\rsa;

   Library Directories:
     c:\tcwin\lib;
     c:\tcwin\owl\lib;
     h:\cmvcwin\output;

   Output Directory:
   h:\cmvcwin\output;

3. fhwzcmd.prj

   The "Directories" for this project file must be:

   Include Directories:
     c:\tcwin\include;
     c:\tcwin\owl\include;
     h:\cmvcwin\src\cmd\inc;
     c:\tcpdos\include;

   Library Directories:
     c:\tcwin\lib;
     c:\tcwin\owl\lib;
     c:\tcpdos\lib;
     h:\cmvcwin\output;

   Output Directory:
     h:\cmvcwin\output;

Building the Project file \src\ui\fhwzmri.prj

This sequence of steps is used also for each of the project files that are listed in “Building the rest of the project files in \src\ui” on page 28.

1. Select the "Project" pulldown and then select "Open Project...".
2. From the "Open Project File" select the project file (see below) to be compiled.
   a. Leave the File Name as "*.prj".
   b. Change the path to:

      h:\cmvcwin\src\ui

   c. Then you will see the following project files:

      fhwzmri.prj
      fhwzmain.prj

   d. Select the project file "fhwzmri.prj" and then click on OK.

      You will see a window called "Project: xxxxx" where xxxxx is the name of the project file. This window lists all the files that are part of the project file.
3. Change the specifications of the directories for the project file. Select the "Options" pulldown and choose "Directories..."
4. From the "Directories" dialog, ensure that the directories are:

   Include Directories:
   c:\tcwin\include;
   c:\tcwin\owl\include;
   h:\cmvcwin\output;

   Library Directories:
   c:\tcwin\lib;
   c:\tcwin\owl\lib;
   h:\cmvcwin\output;

   Output Directory:
   h:\cmvcwin\output;

   You may have to use the combo box to see the additional entries in order to select the proper one.

   When ready, click on the "OK" button.
5. Select the "Compile" pulldown and choose "Make"

   The status of the compilation will be shown in a window called "Compile Status". The "Status" must be "Success".

Be alert

If the Statistics show only around a total of 9,000 lines being processed and the status is "Success", then the build IS NOT OK, because there was something wrong with one of the *.rch files because the total number of lines must bet greater than 22,000.

Sometimes it is necessary to delete some blank lines that are extra that the compiler cannot deal with in "fhwzdef.rch". In this case, edit the file, and make again the project file.

Building the rest of the project files in \src\ui

Perform the sequence of steps described in “Building the Project file \src\ui\fhwzmri.prj” on page 27 for each one of the following project files in cmvcwin\src\ui:

1. fhwzmain.prj

To be processed differently ...

It is necessary to compile first the file "fhwmcst.c", and then to make the whole project. Go to the "Project: fhwzmain" window, select the file "fhwmcst.c", then select the pulldown "Compile" and choose the action "Compile"; this compile has to be successful, although with warnings. Then, build the project file: select "Compile" then choose "Make", and the compile should be successful.

The "Directories" for this project file must be:

```
Include Directories:
  c:\tcwin\include;
  c:\tcwin\owl\include;
  h:\cmvcwin\output;
  h:\cmvcwin\src\ui;

Library Directories:
  c:\tcwin\lib;
  c:\tcwin\owl\lib;
  h:\cmvcwin\output;

Output Directory:
  h:\cmvcwin\output;
```

Now you can exit Turbo C++ and proceed with step “Packaging the code” on page 31.

Known building problems

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

The gencat utility does not work - run time R6001 error.

The problem is that the cmvc.dat and cmvc.cat files exist already. Thus, delete them, and perform mkcat and then gencat. This is accomplished by running the batch file "mk_cat.bat".

When double-clicking the icon, an empty dialog window with an I is shown and the OK button, then the CMVC main window is shown without icons and without menu bar.

The problem is that the FHWZMRI.DLL was not built correctly due to a problem with the compilation of the FHWZMRI.PRJ. Usually the culprit is the file FHWZDEF.RCH, in which only 1 or 2 blank lines should be there, but no more (find a large chunk of empty lines, and delete some of them). The "Compile Status" for FHWZMRI.PRJ should have a value of the Total number of Lines (in the Statistics section) greater than 22,000.

There are errors when linking FHWZMAIN.PRJ, because some mcs* routines are not found.

In the window "Project: fhwzmain.prj" look for the file fhwmcst.c (in the bottom 3rd of the list of files), select it, then select the pulldown "Compile" and choose the option "Compile"; the compilation will be successful. Then, select the pulldown "Compile" and choose the option "Make" and this time the overall build should be successful.

Performing a quick sanity test

If you have already installed the CMVC Client for Windows code in C:\CMVCWIN, then you could copy the executable and the DLLs from the h:\CMVCWIN\OUTPUT directory by performing the batch file **cp_inst**.

After this step, you could invoke CMVCWIN 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 GUI

This chapter describes how to package the CMVC GUI.

MISCELLANEOUS NOTES ON SOFTWARE INSTALLER

The following files are created by Software Installer in the WINDOWS directory of the workstation when installing CMVC Client for Windows. 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:

epfwis.ini

epfwhcnf.cnf

epfwcat.pkg

OVERALL PACKAGING PROCESS

To package the CMVC Client for Windows do the following steps which are described in greater detail later on in this chapter:

1. Gather the files to be shipped: **cp_ship**
2. Compress the files to be shipped: **do_compr**
3. Copy the compressed files to be shipped: **cp_compr**
4. Prepare the CMVCWIN catalog, package and description file for Software Installer for Windows.
5. Build the installation files for the complete package.
6. Copy the installation files and the compressed files into diskettes: **cp_2disk**
7. Perform installation test from diskettes.
8. Place installable diskette images in CMVCBETA and in CARADPC LAN.

Packaging the code

MOVING FILES INTO THE SHIP DIRECTORY

Once the executable for CMVC Client for Windows 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 Client for Windows 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 CMVCWIN.PKG and CMVCWIN.ICF information).

1. Reboot the workstation under native DOS.
2. Perform the batch command file **cp_ship.bat** which will copy into the h:\cmvcwin\ship directory the following files that were produced by the build just recently done, such as:

   h:\cmvcwin\output\fhwzcmd.dll
   h:\cmvcwin\output\fhwzmri.dll
   h:\cmvcwin\output\fhwzmain.exe
   h:\cmvcwin\src\ui\cmvc.cat

3. ... and the following files that are provided by the CMVC development team but are not built everytime:

   h:\cmvcwin\src\ui\cmvc.ini
   h:\cmvcwin\shipstuf\read.me

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

   h:\cmvcwin\shipstuf\fhwz.hlp
   h:\cmvcwin\shipstuf\cmvccmd.inf

5. ... and the following files that were not built by the us, but that were provided by another parties:

   Container (From RTP CUA Development Group)

   h:\cmvcwin\src\ui\fclcnrw.dll
   h:\cmvcwin\src\ui\fcldlgw.dll
   h:\cmvcwin\src\ui\fcldrcw.dll
   h:\cmvcwin\src\ui\fclnbw.dll

   IPF for Windows (From Singapor Lab, thru RTP-ID):

   h:\cmvcwin\shipstuf\hmhelp.hlp
   h:\cmvcwin\src\ui\ipf.dll
   h:\cmvcwin\src\ui\libipfx.dll
   h:\cmvcwin\src\ui\view.exe
   h:\cmvcwin\src\ui\viewh.hlp

6. From the prompt, list the contents of the SHIP directory to find out the amount of DASD that the code takes. For Version 2.3 is approximately 2.7 MB. This data is important because the package\cmvcwin.pkg file needs it.

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 440K for the Software Installer files in diskette 1).

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

KEEP BACK LEVELS OF THE MAP FILES

It is important to keep handy at least one back level of the MAP files from the OUTPUT directory. 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.

PREPARE CMVCWIN INSTALLATION FILES

Follow the instructions in the Software Installer manual, "Guide to Getting Started with the Windows Version", Version 1 Release 2, Appendix A "Step-by-Step Instructions for Enabling a Product for Installation". A brief summary of these steps are shown below.

The following activities will require more environment space and thus, it is necessary to get out of Windows and to delete 2 environment variables, which will be recover later.

Minor changes (no changes to Copyright information)

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

1. Exit Windows.
2. Change to the directory h:\CMVCWIN\PACKAGE:
3. Modify the CMVCWIN installation files:

   package\cmvcwin.icf
   package\cmvcwin.pkg
   package\cmvcwin.dsc

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

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

Major changes (to Copyright information)

This is used when updating the Copyright information.

1. Exit Windows.
2. Make a backup copy of the existing AUTOEXEC file.
3. Copy the file E:\CMVCWIN\AUTOEXEC.BLD as AUTOEXEC.BAT.

   This will provide a smaller environment space, which is needed for further activities.
4. From the DOS prompt issue AUTOEXEC to pick up the new AUTOEXEC.
5. Change to the directory h:\CMVCWIN\PACKAGE:
6. Modify the CMVCWIN installation files:

   package\cmvcwin.icf
   package\cmvcwin.pkg
   package\cmvcwin.dsc

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

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

7. Modify the EPFWIIRC.RC resource file, if adding/changing any of the windows used during installation.

8. Change directory to h:\cmvcwin\package and perform the command h:\CMVCWIN\DO_BLDRC.BAT which will invoke BLDRC.BAT to build the resource file and to bind it to EPFIDLDS.EXE which is used during the installation process.

   DO_BLDRC

9. Perform the command h:\CMVCWIN\DO_BLDIN.BAT which will invoke BLDINST.BAT to prepare all the runtime files from Software Installer to create INSTALL.IN_. This batch file specifies the files COPYRITE.TXT and EPFIUPCK.EXE in order to add them to the installation process.

   DO_BLDIN

   Note: Watch out for the first line in the screen, if it says "Out of environment space", then the result will not be valid and it will be necessary to delete more environment variables in order to obtain more space.

10. Replace the original AUTOEXEC.BAT from AUTOEXEC.WIN.

11. Reboot the workstation to re-establish the environment variables to their original state.

12. Test by running h:\cmvcwin\package\EPFIDLDS.EXE from the Program Manager, File->Run pulldown.

COPY FILES INTO DISKETTES

1. Prepare 2 diskettes (1.4 MB) as follows:

Diskette Label	Description in Outside Label
CMVCWIN1	CMVC Client for Windows, Disk 1 of 2
CMVCWIN2	CMVC Client for Windows, Disk 2 of 2

   You need to use the DOS 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:\CMVCWIN.

3. Perform the batch file **cp_2disk.bat** to copy the files from the package directory into the diskettes.

PERFORM INSTALLATION TEST FROM DISKETTES.

Install the client from diskettes into C:\CMVCWIN.

1. You can proceed with the installation from native Windows or from a Win-OS/2 Full Screen.
2. Insert diskette 1 in drive A: and from the "File" pull-down of the Program Manager for Windows, select "Run".
3. Enter:

   A:\INSTALL
4. 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. Reboot the workstation in OS/2 mode.
2. Change the directory to h:\CMVCWIN\SAVEDSKF
3. Create a subdirectory to store the image; the suggested format for the name is:

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

   savedskf a: h:\cmvcwin\savedskf\SUBDIR\cmvcwin1.dsk /n
5. Perform the following with the diskette 2: which should be inserted in the A: drive.

   savedskf a: h:\cmvcwin\savedskf\SUBDIR\cmvcwin2.dsk /n

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

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

POST-BUILD ACTIVITIES

ESTABLISH LEVEL AS BASELINE WITH *.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 "windows2.3".
3. Create a level and ensure that the type is NOT "development"; appropriate types could be "alpha", "beta", "FVT", "SVT" or "gold".
4. Store the *.MAP files in the OUTPUT directory using the track.

   It is important to keep handy at least one back level of the MAP files from the OUTPUT directory. 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.

   In order to copy these files, go the subdirectory in MAPS and perform the following command:

   ```
   h:\cmvcwin\cp_maps
   ```

   This command will copy the maps from the following directories:

   OUTPUT

   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.
5. Complete the fix records, add the track as a level member to the level just created.
6. 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, TCP/IP for DOS, PC-DOS	IBM Corporation
Microsoft Windows	Microsoft Corporation
Borland Turbo C++	Borland International, Inc.
UNIX	UNIX System Laboratories, Inc.

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 Clients for Windows, Build and Package Checklist

Activity	**Windows (host:_____)**
(optional) clean all directories: cl_all	
clean OBJ, DLL, EXE: cl_misc	
extract full latest committed level	
extract delta latest uncommitted level	
copy needed files: cp_files	
make message catalog: mk_cat	
build the CMD code: ni.prj	
build the CMD code: nls.prj	
build the CMD code: client.prj	
build the CMD code: fhwzcmd.prj	
build the GUI code: fhwzmri.prj	
build the GUI code: compile fhwmcst.c	
build the GUI code: fhwzmain.prj	
copy DLL and EXE: cp_ship	
compress the code: do_compr	
copy the compressed code: cp_compr	
(optional) build the installation files	
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