Using CMVC for Products with Multiple National Language Versions (NLVs)

Document Number TR 29.2183

Sara Schwiesow, Jay Cagle, Angel Rivera

IBM Software Solutions Applications Enabling
Research Triangle Park, North Carolina
Copyright (C) 1996 IBM

ABSTRACT

This technical report describes how two software products use Configuration Management and Version Control (CMVC) to manage their multiple National Language Versions (NLVs). It includes an introduction to CMVC and National Language Support (NLS) enablement, and describes how the ImagePlus Workstation Program/2 (IWPM/2) and VisualAge Generator products use CMVC.

This technical report also provides recommendations and considerations for using CMVC, which are derived from experience with IWPM/2 and VisualAge Generator. In addition, this report describes OS/2 REXX tools that facilitate handling translatable files. It discusses the use of these tools in an end-to-end translation shipment scenario. This report also gives information on how to access the source code of the tools.

ITIRC KEYWORDS

CMVC

National Language Version

NLV

NLS

multiple NLS versions

ABOUT THE AUTHORS

SARA L. SCHWIESOW

Sara is a Senior Associate Programmer at IBM's Software Solutions Application Enabling (SWS AE) development lab in Research Triangle Park, North Carolina. She joined IBM in 1992 after receiving a B.A. in Computer Science and Music Performance from the University of Wisconsin at Madison.

Sara is a key member of the lab's Centralized NLS team. She has provided translation planning, tools support, and shipment services for twelve releases of six product families, and has extensive experience using CMVC with translation shipments.

Sara can be reached via email at sschwies@raleigh.ibm.com.

JAY E. CAGLE

Jay is a Senior Associate Programmer in the VisualAge Generator product area of the IBM SWS AE development lab in Research Triangle Park, North Carolina. He joined IBM in 1991 after receiving an M.S. in Computer Science from the University of Wisconsin at Madison, and a B.S. in Computer Science from Washington University.

Jay is a programmer for the VisualAge Generator Developer product. He played a key role in designing and implementing the CMVC library structure for the VisualAge Generator area. He has been involved with the translation of several VisualAge Generator product releases, and he has written many tools to assist in translation processing and testing.

Jay can be reached via email at jcagle@raleigh.ibm.com.

ANGEL RIVERA

Angel is a Staff Programmer in the IBM SWS AE development lab in Research Triangle Park, North Carolina. He joined IBM in 1989 after graduating with an M.S. in Electrical Engineering from the University of Texas at Austin, and a B.S. in Electronic Systems Engineering from the Instituto Tecnológico y de Estudios Superiores de Monterrey, México.

Angel is the development team lead for the CMVC product. He is regarded as an expert in the development and support of library systems.

Angel can be reached via email at arivera@raleigh.ibm.com.

CONTENTS

ABSTRACT . . . iii
ITIRC KEYWORDS . . . iii

ABOUT THE AUTHORS . . . v
Sara L. Schwiesow . . . v
Jay E. Cagle . . . v
Angel Rivera . . . v

Figures . . . ix

Overview . . . 1

Purpose . . . 3
Audience . . . 3
Assumptions . . . 4

Some CMVC Concepts . . . 5
Overview of CMVC . . . 5
Components and Releases . . . 5
Files . . . 5
Naming Files in CMVC . . . 5
Linking Files between Several Releases . . . 6
File Changes . . . 6
Levels . . . 7
Code Extraction . . . 7
Further Reading for CMVC . . . 7

Some NLS Concepts . . . 9
Primary Language . . . 9
Separating PII from Source Code . . . 9
IBM Language Codes . . . 9
Further Reading for NLS . . . 9

Example Usage of CMVC . . . 11

IWPM/2 . . . 13

VisualAge Generator 15

Recommendations and Considerations 17

Recommendations 19
Store Translatable Primary Language Files Separately 19
Always Use Lower Case 19
Use "8.3" File Names Across Platforms 20
Reflect the Structure of Your Project in CMVC 20
Use CMVC Releases with Tracks and Levels 20
Follow Recommended Basic Extraction Process for Code Building 21
Use the Tools Discussed in this Technical Report 21

Considerations 23
Few Releases vs. Many Releases 23
Using Few Releases 23
Using Many Releases 23
Separated Primary Language vs. Integrated Primary Language 23
Treating the Primary Language as Separate 23
Treating the Primary Language like another National Language 24

Appendixes 25

Appendix A. Description of Tools and How to Get Them 27
Description of Tools 27
chkpii 27
chglist 27
nlsext 28
procdrop 29
Obtaining the Tools 29
Obtaining chkpii 29
Obtaining the Other Tools 30

Appendix B. Translation Shipment Scenario 33
Background 33
Organizational Structure 33
Making a Freeze Level of a Product 33
Information About CMVC for this Scenario 34
Tools Used in this Scenario 34
Sequence of Activities 34
Development Freezes the PII 34

Development Creates Output Lists 34
Centralized NLS Extracts Files for Shipment 35
Centralized NLS Sends Translation Shipment to Translation Centers 36
Centralized NLS Receives Translation Shipment from Translation Centers 36
Centralized NLS Checks in Translated Source Files to CMVC 36
Development Builds the Translated Source Files 37

Appendix C. Glossary of Selected Acronyms 39

Appendix D. Copyrights, Trademarks and Service Marks 41

FIGURES

1. Base Names and Path Names 6
2. Languages, Release Names, and Size of Extracted Files for IWPM/2 14
3. Languages, Release Names, and Size of Extracted Files for VisualAge Generator . 16

OVERVIEW

PURPOSE

This technical report provides recommendations, considerations, and tools to administrators and users of Configuration Management and Version Control (CMVC) who are using CMVC to manage their product's multiple National Language versions (NLVs).

This report begins with an introduction of CMVC and NLS enablement in “Some CMVC Concepts” on page 5 and “Some NLS Concepts” on page 9.

Then it describes how two products use CMVC:

IWPM/2 is a small product with fifteen NLVs that uses two releases in CMVC. See “IWPM/2” on page 13.

VisualAge Generator is a large product with five NLVs that uses six releases in CMVC. See “VisualAge Generator” on page 15.

Based on experience with these products, this report provides recommendations and considerations on how to best use CMVC with any product having multiple NLVs. See “Recommendations” on page 19 and “Considerations” on page 23. For any product using CMVC, these chapters contain important information about manipulating the source files that are used in creating product NLVs.

In the appendixes this report describes some OS/2 REXX tools that IWPM/2 and VisualAge Generator use to facilitate handling NLS related files (see Appendix A, “Description of Tools and How to Get Them” on page 27). Finally, the report details use of the tools in an end-to-end scenario (see Appendix B, “Translation Shipment Scenario” on page 33).

AUDIENCE

The audience for this technical report is:

CMVC family administrators and CMVC project leaders.

Software developers who use CMVC to:

- Track program code
- Track translatable material, also called Program Integrated Information (PII)
- Track the subsequent translated material

ASSUMPTIONS

The following requirements must be met before applying the methods and tools described in this technical report:

Use of CMVC Version 2 for the software configuration and change control.

It is assumed that the reader has some basic knowledge of CMVC. However, a brief description of the relevant CMVC concepts will be provided when appropriate.

Separation of the translatable material from the program code.

See “Some NLS Concepts” on page 9 for more information.

SOME CMVC CONCEPTS

This section provides a brief introduction to CMVC and to some of the basic concepts in this technical report.

OVERVIEW OF CMVC

Configuration Management Version Control (CMVC) allows software developers to better manage their software development process. Problem tracking and change control are integrated with configuration management and version control; project communication is provided by a notification facility; and change history is recorded so that previous work can be recreated.

COMPONENTS AND RELEASES

Because the primary purpose of CMVC is to track the changes to files, CMVC provides two different ways to group these files:

CMVC Components are logical entities that group files according to category. For example, a product could have four components: **base**, **install**, **msg**, and **help**. Each component has an owner; and a set of people can do certain CMVC tasks against the files in each component.

CMVC Releases are logical entities that group files according to how they are packaged for delivery. For example, a product could have two releases, **rel10jpn** and **rel20jpn**. Each release would include files that belong to one or more components. Release **rel10jpn** could have some files that are not included in **rel20jpn** and vice versa.

FILES

Naming Files in CMVC

CMVC Files have attributes of a base name and a path name. A path name can consist of a set of directory names and a base name, or just a base name. If a file name includes directory names, CMVC will create the necessary directory structure when the file is extracted from CMVC.

The following table illustrates the concept of base names and path names.

Figure 1. Base Names and Path Names		
File Name in CMVC	**Base Name of File**	**Path Name of File**
client.dlg	client.dlg	client.dlg
src/server.msg	server.msg	src/server.msg

Names for CMVC files are case-sensitive. The following examples show valid CMVC file names that each refer to a different file. The last file name illustrates a long name that is supported on the PC only by OS/2 HPFS directories.

file1.msg
FILE1.MSG
file1.MSG
File1.msg
src\readme.txt
SRC\readme.txt
src\README.txt
src\thisFileNameIs.Long

Linking Files between Several Releases

CMVC provides a function that allows sharing files between releases. For example, **file1.msg** can be shared between **rel10jpn** and **rel20jpn**. This function is called "linking".

The files of a release are linked "vertically" to another release when one release is the "parent" and the other is the "child". For example, if you have finished with **rel10jpn**, you can create the next release, **rel20jpn**, and link all the files from the parent **rel10jpn** to the child **rel20jpn**.

The files of a release are linked "horizontally" when both releases are "peers", as when a project has several releases for different products that are shipped at the same time. For example, if you have a release **client10enu** for the client product, then you can create a peer release **server10enu** for the server product, and link the files you wish to share from one peer release to the other.

FILE CHANGES

A CMVC release has two methods of tracking the file changes:

1. Version control without associating the file change to a defect or feature.

   The user does not specify a defect or feature when changing a file. The file changes are associated only with the release.

2. Version control by associating the file change to a **CMVC Defect** or a **CMVC Feature**.

   Every time a user changes a file in a given release, the user provides the name of the defect or feature by using a **CMVC Track**, which is the association between a given defect/feature and a given release.

LEVELS

The builder of the application controlled by the CMVC release will usually want to use baselines of the code at certain points of time. A baseline is group of file changes that can be extracted together in a repeatable manner.

A baseline is established by using a **CMVC Level**, which is a collection of tracks that are committed at the same time. Most often, only the most recent level is extracted; but it is also useful to be able to extract an earlier level of a release.

CODE EXTRACTION

There are two ways to extract files from CMVC:

Using File -extract, which will extract one file at a time by using the TCP/IP socket interface.
Using Release -extract or Level -extract, which will extract many files at a time, either by using the Network File System (NFS) or by extracting into a local file system.

For extracting a large number of files, extracting releases or levels is better than extracting files. The reason is you can avoid the overhead that is associated with the processing of an individual File -extract.

The default method for extracting releases and levels is NFS. However, NFS inherently has some security exposures. Due to the unreliability of NFS in OS/2 and Windows, it might be necessary to extract code without using NFS.

CMVC provides an alternative to NFS for extracting releases and levels. In this scenario, files are extracted into a local file system (local with respect to the CMVC server). For more information, see the entries about NOMOUNT in the file /usr/lpp/cmvc/doc/README.pubs in the CMVC administrator's UNIX installation.

FURTHER READING FOR CMVC

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

GC09-1630-00 CMVC MARKETING BROCHURE
GC09-1853-00 CMVC: A Customer's Perspective

SC09-1596-01 *IBM CMVC Client Installation and Configuration*
SC09-1597-01 *IBM CMVC User's Reference*
SC09-1631-02 *CMVC Server Administration and Installation*
SC09-1633-00 *IBM CMVC Concepts*
SC09-1635-01 *IBM CMVC Commands Reference*
GG24-4178-00 *Did you say CMVC?*

If you have questions on operational issues with NLS and CMVC, consult technical report 29.2169 "How to use CMVC with National Language Support (NLS) and Double-Byte Character Sets (DBCS)". If you have installed CMVC, a copy of this technical report can be found in /usr/lpp/cmvc/doc/README.nls_dbcs.

SOME NLS CONCEPTS

PRIMARY LANGUAGE

In this technical report, the term **primary language** refers to the language used by the developers. In the examples and scenario (see “Example Usage of CMVC” on page 11 and Appendix B, “Translation Shipment Scenario” on page 33), the primary language is U.S. English; but the methods described in this report are applicable to products developed in any primary language. It is also possible that the primary language is not the native language of some or all of the developers.

SEPARATING PII FROM SOURCE CODE

One of the fundamental concepts for the national language enablement of code is separating the translatable material, or PII, from the program code. Translatable material includes messages, helps, window elements, and prompts. In this report, the term **primary language file** refers to the separated PII.

By separating the translatable material from the program code, the process of localizing the final application for another national language is greatly simplified. Only the files that contain the translatable information are provided to the translators, so there is no chance of corrupting the program code during translation. Furthermore, there is no need to recompile the whole code set for each national language.

IBM LANGUAGE CODES

This technical report uses the three-letter IBM language codes to identify different national languages. For example, "ENU" denotes U.S. English, and "JPN" denotes Japanese.

FURTHER READING FOR NLS

For more information on NLS enablement, consult the following manual:

SE09-8001-02 National Language Design Guide Volume 1

For more information on the IBM language codes, consult the "Language codes" chapter in the following manual:

SE09-8002-02 National Language Design Guide Volume 2

This manual also provides many other guidelines for implementing NLS.

EXAMPLE USAGE OF CMVC

This section contains examples of how CMVC is used in two products that have multiple NLVs.

The description of each product is a snapshot of the point when this technical report was written. It is possible that some of the characteristics mentioned here have changed since then.

IWPM/2

IWPM/2 is a part of the ImagePlus family of products. It is the client front-end for the VisualInfo scanner server software.

PII separation: All PII is separated from the program code into translatable primary language files.

Number of non-translatable files: 354

Number of translatable primary language files: 27

Number of CMVC releases per released product version: 2

Number of languages including primary: 15

Path names:

bin
evn2/bld/src
include
lib
make
obj
res
src
src/enu, src/jpn, src/nor, etc.

Location of primary language files: All primary language files have a path of **src/enu.** This path contains only primary language files.

Case: Names of files, components, and releases are all in lower case.

Language source file names: Language source files have the same base name across all languages; however, they have different path names for each language. For example, an English file could be named **src/enu/file1.msg**; this file translated into Japanese would be named **src/jpn/file1.msg**.

Built file names: Built files have the same base name across all languages; however, they have different path names for each language. Built files are stored in CMVC.

Links: Files are linked vertically from one released product version to the next, for example, from **iwpm21enu** to **iwpm211enu**.

Extract and build method: All languages and the program code are extracted at the same time. Only committed levels are extracted for builds.

Components:

iwpm (parent)
base
conf
install
nlviwpm

Release information:

Figure 2. Languages, Release Names, and Size of Extracted Files for IWPM/2

Language	Release Name	Size of extracted files
English & Program Code	iwpm21enu	13 068 KB
Arabic	iwpm21nlv	850 KB
Brazilian Portuguese	iwpm21nlv	724 KB
Danish	iwpm21nlv	710 KB
Dutch	iwpm21nlv	735 KB
French	iwpm21nlv	909 KB
Finnish	iwpm21nlv	898 KB
German	iwpm21nlv	932 KB
Hebrew	iwpm21nlv	950 KB
Italian	iwpm21nlv	932 KB
Japanese	iwpm21nlv	917 KB
Korean	iwpm21nlv	894 KB
Norwegian	iwpm21nlv	892 KB
Spanish	iwpm21nlv	923 KB
Swedish	iwpm21nlv	873 KB
Total extracted product source and executables		25 207 KB

VISUALAGE GENERATOR

VisualAge Generator is a comprehensive fourth generation language environment for building applications for client/server models, as well as for traditional stand-alone applications. The applications created in the development environment are targeted for multiple runtime platforms.

VisualAge Generator is a family of separately orderable products. The information given below is for the Developer product. For the complete family, the total number of non-translatable files is much larger. The total number of translatable files is only slightly larger; most of the translation is within the Developer product.

PII separation: All PII is separated from the program code into translatable primary language files.

Number of non-translatable files: 1,700

Number of translatable primary language files: 3,200

Number of CMVC releases per released product version: 6

Number of languages including primary: 5

Path names: Files have only a base name; they do not have a path name.

Location of primary language files: All primary language files are located in a separate release, **dev.20e**. This release contains only primary language files.

Case: Names of files, components, and releases are all in lower case.

Language source file names: Language source files have the same base name and path name across all languages; however, they are stored in separate releases. For example, an English file could be named **file1.msg** in the **dev.20e** release; this file translated into Japanese would be named **file1.msg** in the **dev.20j** release.

Built file names: Built file names contain a single letter language identifier. Built files are not stored in CMVC.

Links: Files are linked vertically from one released product version to the next, for example, from **dev.20e** to **dev.21e**. Files are also linked horizontally in order to share files between products in the same released product version, for example, from **dev.20e** to **gen.20e**.

Extract and build method: Each language is extracted separately. The non-translatable files are also extracted separately. Only committed levels are extracted for builds. Once extracted, the NLV builds are invoked separately, but can be executed concurrently.

Components:

cmd.doc	dev.help	dev.rcd	intro.doc
cmn.doc	dev.lby	dev.samp	libgde.doc
cmnlicense.doc	dev.lgc	dev.sql	license.doc
cmnplan.doc	dev.lib	dev.stg	migapps.doc
dev	dev.lst	dev.svc	msg.doc
dev.app	dev.map	dev.tbl	plan.doc
dev.build	dev.mce	dev.utl	refsum.doc
dev.cmn	dev.misc	develop.doc	run400.doc
dev.csp	dev.msg	devinstall.doc	runmvs.doc
dev.dl1	dev.parse	esf.doc	runos2aix.doc
dev.dte	dev.prc	guigde.doc	runvse.doc
dev.esf	dev.prj	file.lst	trans.doc
dev.glb	dev.psb	install	tut

Release information:

Figure 3. Languages, Release Names, and Size of Extracted Files for VisualAge Generator

Language	Release Name	Size of extracted files
Program Code	dev.20	40 MB
Brazilian Portuguese	dev.20p	11 MB
English	dev.20e	10 MB
Japanese	dev.20j	22 MB
Korean	dev.20k	21 MB
Simplified Chinese	dev.20c	15 MB
Total extracted product source and executables		119 MB

RECOMMENDATIONS AND CONSIDERATIONS

RECOMMENDATIONS

The following are recommendations for all products, independent of size or complexity.

STORE TRANSLATABLE PRIMARY LANGUAGE FILES SEPARATELY

Store all primary language files in a separate place, and store only the primary language files in that place.

You can achieve this separation by any of the following:

Storing the primary language files in a separate component

Storing the primary language files in a separate release

Using a different path name for the primary language files than the path names used for the program code files

ALWAYS USE LOWER CASE

Use only lower case for all objects in CMVC, such as file names, component names, release names, and defect names. In this way you greatly simplify file manipulation and avoid confusion.

File names in CMVC are case-sensitive. The UNIX users of CMVC are familiar with this convention; however, because file names in OS/2 and Windows are not case-sensitive, the users of these platforms need to take this convention into account. See section “Naming Files in CMVC” on page 5 for more details.

The following problems occur when users on OS/2 or Windows CMVC clients use mixed-case file names:

Developers cannot find files or overlook files in sorted file lists.

When displaying files, the case-sensitive CMVC sort engine gives a different sort order than does the case-insensitive OS/2 or Windows sort engine (e.g., with the **dir** command).

Developers cannot find files or overlook files when searching for files.

CMVC searches only for the specified case, whereas OS/2 and Windows search for either upper or lower case.

Developers overwrite files upon extraction.

CMVC treats file names with the same characters and different capitalizations as different files, whereas OS/2 and Windows do not.

USE "8.3" FILE NAMES ACROSS PLATFORMS

If you are developing for multiple environments, such as AIX, OS/2, and Windows, use no more than eight characters for the file name and no more than three characters for the file extension.

The FAT file systems of DOS can only handle file names that follow the 8.3 naming convention. If you use a long file name in AIX, CMVC cannot extract it to a FAT file system on OS/2 or Windows.

REFLECT THE STRUCTURE OF YOUR PROJECT IN CMVC

Match the CMVC releases and components to the structure of your product. In this way you can change or add product function with minimal rework in CMVC.

For example, all IWPM/2 version 2.1 product function is developed in the **iwpm21enu** release. If a function name changed or a function needed to be added, only a modification of a component name or a creation of a new component would need to be performed.

USE CMVC RELEASES WITH TRACKS AND LEVELS

For non-prototype development, use releases configured with the track subprocess. In this way, every file change can be identified by a defect or feature. See “File Changes” on page 6. Also use releases configured with the level subprocess to provide baselines of the code. See “Levels” on page 7.

Furthermore, have only one uncommitted level active per release, where all the latest changes of the files are incorporated. If you have multiple uncommitted levels active at any given time, you might have problems with file dependencies when you try to commit a level.

For example, you might have a level called **960805** that you want to commit. If you have more than one uncommitted level, you might have another active level called **960729.** You could have file changes in level **960805** that have a dependency on file changes in level **960729.** Thus, you could not commit level **960805** before the pending changes in level **960729** are also committed.

FOLLOW RECOMMENDED BASIC EXTRACTION PROCESS FOR CODE BUILDING

There are several approaches for extracting the code to be built (compiled and linked). The following basic extraction process provides a consistent baseline for the builds:

1. Ensure that the target directory is suitable (for example, there are no files with pending changes to be checked in) and that it has the proper permissions.
2. Do a full Level -extract of the most recent committed level.

   This is identical to using Release -extract with committed versions.
3. Do a delta Level -extract of the uncommitted level.

   This will extract only the changes that have been placed on tracks that are level members of the uncommitted level. All other new file changes that are not in the level will not be extracted (however, they would be extracted if you used Release -extract with current versions).

USE THE TOOLS DISCUSSED IN THIS TECHNICAL REPORT

If you are using the CMVC OS/2 client, use the OS/2 tools which are discussed in Appendix A, “Description of Tools and How to Get Them” on page 27. Appendix B, “Translation Shipment Scenario” on page 33, shows how you might use these tools to facilitate handling your translatable files.

CONSIDERATIONS

The following are issues that all products need to address. What approach you use will depend on the size and complexity of your product.

FEW RELEASES VS. MANY RELEASES

Using Few Releases

If you use few releases, you will have low overhead for track and level manipulation. For example, only one track is needed to check in translated files for all languages to the **iwpm21nlv** release. Consequently, only one level is needed to commit all languages other than the primary language.

With few releases, however, you will need to extract all languages stored in the same release at the same time. If your product is large, the space needed for all languages may prohibit this approach.

Using Many Releases

If you use many releases, you can efficiently extract only the language or languages you want.

With many releases, however, you will have high overhead in maintaining levels and tracks for each release. If your product has many languages, the overhead needed to manipulate tracks and levels may prohibit having a release for each language.

SEPARATED PRIMARY LANGUAGE VS. INTEGRATED PRIMARY LANGUAGE

Treating the Primary Language as Separate

If you treat the primary language as separate from the translated languages, you can add languages with minimal library manipulation. For example, languages are uniquely identified in the setup of IWPM/2 by a path name of **src**/**xxx**, where **xxx** is the IBM three-letter language identifier. If a new language is added to the set of translations, its translated files are simply checked in with a path name of **src**/**xxx**.

When you treat the primary language as separate from the translated languages, however, you do lose a useful parallelism between an NLV and its library structure. In the setup of IWPM/2, there is an awkward one-to-many relationship between a release associated with a language and the language files stored in a release. For example, the release **iwpm21nlv** contains all languages except the primary language, which is contained in the release

iwpm21enu. For example, suppose there is an English file named **src/enu/readme.txt** in the release **iwpm21enu**. It is translated to Japanese, and the Japanese file is named **src/jpn/readme.txt** in the release **iwpm21nlv**. A Brazilian Portuguese version of the file is also in release **iwpm21nlv**, but it is named **src/ptb/readme.txt**.

You also foster a mindset of "Work on the primary language first, then go back and do NLS". This mindset is detrimental when a product is trying to properly enable a product for NLS from the beginning.

Treating the Primary Language like another National Language

If you treat the primary language version as just another NLV, you create a clean one-to-one relationship between the NLV and the location of the NLV. For example, in the setup of VisualAge Generator, each NLV is handled in its own release, including the primary language NLV. In this case, the English file is named **server.msg** in the release **dev.20e.** It is translated to Japanese, and the Japanese file is named **server.msg** in the release **dev.20j.** The Brazilian Portuguese version of the file is named **server.msg** in the release **dev.20p.**

You also help reach one of IBM's strategic goals, which is to have developers intuitively thinking globally.

When you treat the primary language version as just another NLV, however, you do ignore the fact that development really does go on in the program code and just one language. You might lose some amount of time and space in treating the primary language version as just another NLV.

APPENDIXES

APPENDIX A. DESCRIPTION OF TOOLS AND HOW TO GET THEM

DESCRIPTION OF TOOLS

chkpii

The **chkpii** tool verifies that translated OS/2 and Windows PII files are in the correct format before you use them in the product build process. The tool validates the translated PII files against the primary language PII files.

The **chkpii** tool can also validate primary language PII files alone. The tool will find lines longer than 255 characters, will check whether all IPF tags have ending tags, and will check tags against known IPF tags. It will also check the formatting of files with double-quoted string structure.

The **chkpii** tool can validate BookMaster-tagged files as well. For those tags that are the same as IPF, normal checking will be done. However, the BookMaster-only tags will cause **chkpii** to flag them as errors. It is still helpful to use this tool against BookMaster-tagged files, but the error reports will be somewhat larger.

Because the **chkpii** tool is corporate and widely available, its syntax and example usage is not discussed in this report. For more information, consult the documentation and help that comes with **chkpii**. For information on getting **chkpii,** see “Obtaining the Tools” on page 29.

chglist

The **chglist** tool is used to produce a list of files that have changed over a given period. The user may provide file name and component specifications, as well as starting and ending levels. The output of **chglist** is a list of files that changed, including the path name of the file; the final version number of the file as of the specified committed level; the component of the file; and the type of change made. For example, here is one line in an output list that **chglist** created:

```
src/enu/epfisinc.pkg   1.3   install   delta
```

This tool is run against the release of the primary language to produce a list of files to be sent to the translation centers. The list is used as input to the **nlsext** and **procdrop** tools.

Syntax of tool:

```
chglist {all|delta} -release <rel> [-startlevel <level>]
          [-endlevel <level>] [-path <paths>]
          [-comp <components>] > <outfile>
```

all — lists all source files of the primary language that have changed, even if the only activity in this release is a link.

delta — lists only files with more activity than a link in this release.

-release <rel> — primary language CMVC release to process.

-startlevel <level> — limits the output to files that were changed after <level> was committed. Changes in <level> are not included.

-endlevel <level> — limits the output to files that were changed before <level> was committed. This includes changes made within <level>.

-path <paths> — limits the output to files that have the specified paths.

-comp <components> — limits the output to files that belong in the specified components.

<outfile> — name of the output file to contain the list of source files of the primary language that have changed.

nlsext

The **nlsext** tool reads the file list produced by **chglist** and extracts the changed files from CMVC so they may be shipped to the translation centers.

Syntax of tool:

```
nlsext -file <list-file> [-dir <extract-dir>]
```

-file <list-file> — name of the file that contains the list of source files of the primary language that have changed.

-dir <extract-dir> — name of the target directory for the extracted files.

procdrop

The **procdrop** tool is used to process the files returned from the translation centers. This shipment of files is referred to as an NLS "drop".

This tool reads the list of changed files in the primary language produced by the **chglist** tool, and checks in the returned translated files to the corresponding NLV release. If a file was new in the primary language release, the file is created in the NLV release. The **procdrop** tool also duplicates in the NLV release any links the primary language version of the file has.

Syntax of tool:

```
procdrop -list <lists> -primRelId <xxx> -nlvRelId <xxx> -lang <lang>
            [-dir <dir>] { -defect <def> | -feature <fea> }
```

-list <lists> names of the files that contain the lists
of the source files to process.

-primRelId <id> suffix identifier for primary release name.
Identifier must be at the end of the release name,
and must be the same length as the nlvRelId.

-nlvRelId <id> suffix identifier for NLV release name.
Identifier must be at the end of the release name,
and must be the same length as the primRelId.

-lang <lang> identifier for NLV language being processed.
Usually the same length as primRelId and nlvRelId.

-dir <dir> directory in which to find the NLV source files.

-defect <def> CMVC defect name for processing the NLV source files.

-feature <fea> CMVC feature name for processing the NLV source files.

OBTAINING THE TOOLS

Obtaining chkpii

Obtain **chkpii** from the corporate NLSTOOLS disk on VM. Use these steps:

1. From the VM ready prompt, type toolcat nlstools.
2. Search on **chkpii**.
3. Get the package.
4. Follow the package's instructions for unpacking and installation.

Obtaining the Other Tools

The other tools described in this technical report can be downloaded as follows:

From the IBM intranet (only for IBM employees).

From the Internet (open to everyone).

IBM Intranet

Web Home Page: You can access the CMVC Service/Development Home Page at:
http://keithp.raleigh.ibm.com/~cmvcsupt

From the index at the top of the page, select the section "NLS tools".

FTP: You can download the code from our internal FTP site, by doing:

1. ftp keithp.raleigh.ibm.com
2. login as 'anonymous' and for password give your email address.
3. cd pub/cmvc/fixes
4. binary
5. get chglist.cmd
6. get nlsext.cmd
7. get procdrop.cmd
8. quit

Internet

Web Home Page: Not available.

FTP: You can download the code from our external FTP site, by doing:

1. ftp ftp.software.ibm.com
2. login as 'anonymous' and for password give your email address.
3. cd ps/products/cmvc/fixes
4. binary
5. get chglist.cmd
6. get nlsext.cmd
7. get procdrop.cmd

8. quit

APPENDIX B. TRANSLATION SHIPMENT SCENARIO

BACKGROUND

Below is an outline of a typical translation shipment scenario using CMVC and the tools described in this technical report. It is meant to give you ideas of how you can perform translation shipments in your own organization. Your organization could use this scenario as a model, modifying it as necessary.

Organizational Structure

In the scenario below, the term **development** refers to programmers, technical writers, and build personnel for a particular project; the term **centralized NLS** refers to NLS planners and NLS technical support personnel.

In the SWS AE development laboratory, the centralized NLS team provides services to all lab products for common, specialized activities that all products need. These activities include:

Library manipulation of translated files
File shipping and receiving
Translation tool guidance and expertise

Making a Freeze Level of a Product

Most products have several iterations of build (compilation and linking) activities during the development cycle. In the scenario below, development creates a **build level** for each build, tracks are added as level members, the level is extracted, the code is built, and if the build is successful, then the level is committed. If there are problems with the build, then the tracks with files that do not compile properly can be removed as level members, the files can be fixed, the tracks can be added again as level members, and then the build process is repeated.

As part of the development cycle, products perform translation shipments. Before a product ships translatable material to the translation centers, a particular level needs to be documented as being the **freeze level**. In this way, it is easy to know exactly what files (and what versions of the files) were shipped.

In order to easily identify a freeze level, development uses a different naming convention for a freeze level than it uses for a build level. The name of the freeze level has a descriptive name that can be recognized as the translation shipment snapshot, for example, **nlsdrop2**. Development also synchronizes a freeze level with an already existing build level.

First, a build level is chosen to also be the freeze level. Immediately after the build level has been committed, the freeze level is created and committed. The freeze level will have no explicit level members. In cases such as this, CMVC implicitly gives a level exactly the same level members as the most recently committed level before it.

An alternative is to use customized values for the level type. The CMVC administrator can customize the values of the entries for **leveltype** in the **config.ld** file to include values such as **build** and **freeze**. You can change the level type at any time, regardless of whether the level is committed or uncommitted.

Information About CMVC for this Scenario

The CMVC OS/2 client 2.3.0.2 was used in this scenario, with a CMVC 2.3.0.14 server.

The CMVC process for the releases is **track_level**.

The state of CMVC is as described in section "IWPM/2" on page 13 above, with the addition that a freeze level called **nlsdrop1** has already been committed.

The translation shipment cycle for the previous NLS cycle, **nlsdrop1**, has been completed.

The translation shipment cycle for the current NLS cycle, **nlsdrop2**, is described below.

Tools Used in this Scenario

The tools described in this report assist in handling the language files. See Appendix A, "Description of Tools and How to Get Them" on page 27 for more information.

SEQUENCE OF ACTIVITIES

Development Freezes the PII

Prerequisites:

All source files of the primary language (English) have been spell-checked and proofed. Also, **chkpii** has been run against them, and any errors found by the tool have been corrected in CMVC.

All files have been checked into CMVC.

All tracks to be included in the build and freeze levels are in the **integrate** state.

Development has performed a build and has created, committed, and completed a build level.

Activity: Development creates and commits a freeze level in CMVC.

Development Creates Output Lists

Prerequisites:

There are one or more levels associated with the shipments for translation.

Activity: Development runs **chglist.cmd**.

Example input to tool:

```
chglist delta -release iwpm21enu -startlevel nlsdrop1 -endlevel nlsdrop2
        -path src/enu/  > nlsdrop2.lis
```

Example output of tool:

```
iwpm21enu
src/enu/epfisinc.pkg   1.3   install   delta
src/enu/evn.icf        1.4   install   delta
src/enu/evn2.pkg       1.6   install   delta
src/enu/evn2conf.pkg   1.7   install   delta
src/enu/evn2cus.dlg    1.3   conf      delta
src/enu/evn2cus.rc     1.4   conf      delta
src/enu/evn2us.dlg     1.4   base      delta
src/enu/evn2us.ipf     1.6   base      create
src/enu/evn2cus.ipf    1.6   conf      create
src/enu/evn2us.rc      1.7   base      delta
src/enu/evnart.pkg     1.6   install   delta
src/enu/evnscdbs.pkg   1.7   install   delta
src/enu/iirc.rc        1.4   install   delta
src/enu/info1.txt      1.2   install   delta
```

Centralized NLS Extracts Files for Shipment

Prerequisites:

Output lists from **chglist.cmd** are available and are valid.

Activity: Centralized NLS uses **nlsext.cmd** to extract files to a LAN directory.

Example input to tool:

```
nlsext -file cmd\nlsdrop2.lis
```

Example output of tool:

```
1/14: file -extract src/enu/epfisinc.pkg -release iwpm21enu -version 1.3
2/14: file -extract src/enu/evn.icf -release iwpm21enu -version 1.4
3/14: file -extract src/enu/evn2.pkg -release iwpm21enu -version 1.6
4/14: file -extract src/enu/evn2conf.pkg -release iwpm21enu -version 1.7
5/14: file -extract src/enu/evn2cus.dlg -release iwpm21enu -version 1.3
6/14: file -extract src/enu/evn2cus.ipf -release iwpm21enu -version 1.6
7/14: file -extract src/enu/evn2cus.rc -release iwpm21enu -version 1.4
8/14: file -extract src/enu/evn2us.dlg -release iwpm21enu -version 1.4
9/14: file -extract src/enu/evn2us.ipf -release iwpm21enu -version 1.6
```

```
1 /14: file -extract src/enu/evn2us.rc -release iwpm21enu -version 1.7
11/14: file -extract src/enu/evnart.pkg -release iwpm21enu -version 1.6
12/14: file -extract src/enu/evnscdbs.pkg -release iwpm21enu -version 1.7
13/14: file -extract src/enu/iirc.rc -release iwpm21enu -version 1.4
14/14: file -extract src/enu/info1.txt -release iwpm21enu -version 1.2

All files extracted without error.
```

Centralized NLS Sends Translation Shipment to Translation Centers

Prerequisites:

The files to be translated were extracted using **nlsext.cmd**.

Activity: Centralized NLS packages the files into one large file and sends it to the appropriate translation centers.

Centralized NLS Receives Translation Shipment from Translation Centers

Prerequisites:

The translation centers have translated all the files, spell-checked and proofed them, run **chkpii**, fixed any errors, and have returned the translated files in one large packed file.

Activity: Centralized NLS unloads the large packed file onto the LAN.

Centralized NLS Checks in Translated Source Files to CMVC

Prerequisites:

Centralized NLS has confirmed that translated files have been processed with **chkpii** and errors have been fixed.

Activity: Centralized NLS uses **procdrop.cmd** to check translated files into CMVC.

Example input to tool:

```
procdrop -list t:\nlsdrops\iwpm\21base2\cmd\nlsdrop2.lis -prim enu -nlv
         nlv -lang jpn -feature 984
```

Example output of tool:

```
Getting common file information for release iwpm21enu...
Getting common file information for release iwpm21nlv...
1/1 1/14: file -lock src/jpn/epfisinc.pkg -release iwpm21nlv
1/1 1/14: file -checkin src/jpn/epfisinc.pkg -release iwpm21nlv      -feature 984

1/1 2/14: file -lock src/jpn/evn.icf -release iwpm21nlv
1/1 2/14: file -checkin src/jpn/evn.icf -release iwpm21nlv      -feature 984

1/1 3/14: file -lock src/jpn/evn2.pkg -release iwpm21nlv
1/1 3/14: file -checkin src/jpn/evn2.pkg -release iwpm21nlv      -feature 984
```

```
1/1 4/14: file -lock src/jpn/evn2conf.pkg -release iwpm21nlv
1/1 4/14: file -checkin src/jpn/evn2conf.pkg -release iwpm21nlv      -feature 984

1/1 5/14: file -lock src/jpn/evn2cus.dlg -release iwpm21nlv
1/1 5/14: file -checkin src/jpn/evn2cus.dlg -release iwpm21nlv      -feature 984

1/1 6/14: file -lock src/jpn/evn2cus.rc -release iwpm21nlv
1/1 6/14: file -checkin src/jpn/evn2cus.rc -release iwpm21nlv      -feature 984

1/1 7/14: file -lock src/jpn/evn2us.dlg -release iwpm21nlv
1/1 7/14: file -checkin src/jpn/evn2us.dlg -release iwpm21nlv      -feature 984

1/1 8/14: file -create src/jpn/evn2us.ipf -release iwpm21nlv -component base
-feature 984

1/1 9/14: file -create src/jpn/evn2cus.ipf -release iwpm21nlv -component conf
-feature 984

1/1 1 /14: file -lock src/jpn/evn2us.rc -release iwpm21nlv
1/1 1 /14: file -checkin src/jpn/evn2us.rc -release iwpm21nlv      -feature 984

1/1 11/14: file -lock src/jpn/evnart.pkg -release iwpm21nlv
1/1 11/14: file -checkin src/jpn/evnart.pkg -release iwpm21nlv      -feature 984

1/1 12/14: file -lock src/jpn/evnscdbs.pkg -release iwpm21nlv
1/1 12/14: file -checkin src/jpn/evnscdbs.pkg -release iwpm21nlv      -feature 984

1/1 13/14: file -lock src/jpn/iirc.rc -release iwpm21nlv
1/1 13/14: file -checkin src/jpn/iirc.rc -release iwpm21nlv      -feature 984

1/1 14/14: file -lock src/jpn/info1.txt -release iwpm21nlv
1/1 14/14: file -checkin src/jpn/info1.txt -release iwpm21nlv      -feature 984


All actions completed successfully.
```

Development Builds the Translated Source Files

Prerequisites:

All the translated files have been successfully checked in to CMVC.

Activity: Development creates a build level, adds the tracks with the translated files as level members, extracts the files, builds the files and if the build is successful, commits the build level.

APPENDIX C. GLOSSARY OF SELECTED ACRONYMS

CMVC Configuration Management and Version Control

DBCS Double-Byte Character Set

FAT File Allocation Table

FTP File Transfer Protocol

HPFS High Performance File System

ITIRC IBM Technical Information Retrieval Center

IWPM/2 ImagePlus Workstation Program/2

LAN Local Area Network

NFS Network File System

NLS National Language Support

NLV National Language Version

PII Program Integrated Information

REXX Restructured Extended Executor Language

SWS AE Software Solutions Application Enabling

TCP/IP Transmission Control Protocol / Internet Protocol

TR Technical Report

APPENDIX D. 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, AIX, VisualAge Generator, ImagePlus, CMVC, TeamConnection	IBM Corporation
UNIX, USL	UNIX System Laboratories, Inc.
Microsoft, Windows	Microsoft Corporation
NetLS, Network Licensing System	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
OSF, OSF Motif	Open Software Foundation, Inc.
PostScript	Adobe Systems Incorporated
INFORMIX	Informix Inc.
ORACLE	Oracle Corp.
SYBASE	Sybase Inc.
Sun, SunOS, OpenWindows, Solaris	Sun Microsystems Inc.
HP, HP-UX, SoftBench	Hewlett-Packard Company
X Window System	Massachusetts Institute of Technology

END OF DOCUMENT