CMU-ITC-84-022

Conventions for ITC System Source Code

Editor: WJHansen

Property of IBM Corporation

Information Technology Center
Carnegie-Mellon University
Pittsburgh PA 15213

Introduction

'footprints in the sands of time'
Longfellow

Software will be the ITC's major footprints. This note describes a few conventions for identifying that software and storing it centrally, so future explorers can hope to find the right beach. The note incorporates the following documents:

Using RCS Remotely, David S. H. Rosenthal, ITC, April, 1984

Makefiles for lsh and wmlib, WJHansen, ITC, April, 1984

Sample program and itc.h, John Howard, ITC, April, 1984

For complete information on the rcs commands, see the manual entries for the rcs system, Walter Tichy, Purdue University; they are generally distributed with this document.

Contents

Program Contents

A standard ITC source file begins like this:

```
/*  module-name - brief description on one line
*/

/* Author: WJHansen
 *
 *  more commentary,
 *  including usage instructions
 *  and other facts
 *
 */

#include "itc.h"
private char rcsid[] = "$Header$";
private char IBMid[] =
        "Property of IBM Corporation";

/*
        $Log$
*/
```

'*More commentary*' should serve as the first source of documentation of the program, module, or library entry.

The file included by **#include "itc.h"** is in the **Examples** section below. It defines a few items of general utility, especially '**private**', which is used by the next two lines. **Private** indicates that this identifier will not be exported so there will be no name conflicts when several modules are linked together.

Each time the file is checked in, RCS emends the *$Header$* to describe the latest revision:

The contents of *IBMid* will appear in both source and object.

Log will also be replaced appropriately by RCS.

For a makefile the initial commentary includes name, author, "Property of IBM Corporation", and$Header$. A shell script has the same commentary, but at the end, following a "exit 0" line.

Library header files (*.h) must not include *itc.h* nor define *rcsid* and *IBMid*. Their inital commentary includes name, author, "Property of IBM Corporation", and $Header$. In documentation and help files the attribution and ownership lines go at the beginning while $Header$ and Log go at the end. See, for example, the end of this file.

Tailoring code to specific environments

To prepare code that can be executed on more than one machine or operating system, a number of compile time variables are defined by the **rinse** mechanism for running Makefiles. As described below, **rinse** should be used in place of **make** for all compilations.

Surround VAX-specific code with

```
        #ifdef VAX
            . . .
        #endif VAX
```

Enclose SUN-specific code with

```
        #ifdef SUN
            . . .
        #endif SUN
```

The Makefile rule for making this program then needs -D$(MACHINE) in CFLAGS; see the discussion of Makefiles below.

"#include" lines should be written with quotes instead of angles for all but files that are part of the system distribution. For example:

```
        #include "usergraphics.h"
```

This prevents precedence problems when installing pieces of the system. See the Makefile for **wmlib** in the appendix.

Note: Makefiles should not refer to **usergraphics.a** but should use
${DESTDIR}${LIBDIR}/libitc.a
instead.

Central Source Storage System

ITC system source files are stored on a cental source host, currently the '**linus**' Vax. From here they are retrieved for review and modification or to be recompiled for installation on servers.

Each piece of source in the ITC system is part of some 'component' of the system. Each component is managed as a collection of files stored together in a directory on the central source host. (A component's directory may have subdirectories.) The directory for some particular source file can be found by use of the **rcsfind** command.

Source files are stored in subdirectories of */user/source* on linus, but ordinary mortals can only write these files through the agency of remote RCS. This is done by using **rcslink** to link a local subdirectory to one of the central source subdirectories and then copying files back and forth with **rco** and **rci**. When a newly released facility is to be installed, **rinse** is done for it on server **mother** from whence it is copied to other servers.

For ITC source, the principal central source subdirectories of interest are
linus:/user/source/usr/local/xxx/RCS
where values for xxx are the names of component facilities of the system: wm, xyzzy, itclogo, wmlib, aledit, or whatever. For example, this document

is part of the *itcrcs* component and its RCS source file is
linus:/user/source/usr/local/bin/itcsrcs/RCS/itcsource.d,v
(The final ',v' denotes a file managed by the RCS system. It contains both the source itself and information about all former versions of the file.)

By convention, names of central source directories are shortened by omitting the initial 'linus:/user/source/' and the final '/RCS/'. Thus the central sources for *itcrcs* are said to be in *usr/local/bin/itcrcs*

The source for a component includes any appropriate library and help files. Thus the source of **usergraphics.h** is in usr/local/bin/wm/ In a few cases in the past, there has been no appropriate component directory so files have been stored directly in *usr/local/bin* or whatever. Such cases should be eliminated.

In some cases ITC'ers correct files that are properly part of the 4.2BSD sources and not ours. These can go under source directories that are not in *usr/local.* Consider **login**, which differs considerably between 4.1c and 4.2 and has new ITC versions for both. The original 4.1c version is not stored; the 4.2 version is in *bin/,* the revised 4.1c version can go in *old/login,* the latest version in *usr/local/bin/login/.*

Remote RCS Commands

This section details each of the ITC command files for remote access to RCS sources. Note the following:

- Most of these commands must be executed when the working directory contains a file 'RemoteRcsDir'. Create this file with the **rcslink** command.
- They are shell command files and parse switches primitively. Therefore write only one switch letter per dash. For instance, use "-R -L" and not "-RL".
- The commands exit with value 0 if successful and a small positive integer for failure.
- Rco, rci, and rrcs adjust the mode of each file to accord with whether it is locked to you. If you do not have a lock, the file is read-only. Setting a lock changes to read-write.

rcsfind *name*

searches a list of all files stored in the system sources on linus. Returns a table of filename and directory for entries whose filename contains the string *name*. For example, **rcsfind wm** produces a list that includes:

```
newmat.c      usr/lib/lib2648
windowman.c usr/local/bin/emacs
wm.c          usr/local/bin/wm
wmdev.c               usr/local/bin/pressdvi
```

(in each case the filename contains the string 'wm')

name may be a sequence of patterns for egrep. Any file meeting any of the patterns will be printed. The patterns may not include dollar-sign, apostrophe, double-quote, or exclamation-point.

File names of entries in usr/local all have an initial blank so they will be at the beginning of the file and will match first.

rcsfind exits with value zero.

rcslink [-c] *remotename*

creates the file RemoteRcsDir in the current directory. This file serves as a link between this directory and the corresponding directory in the central system source files.

rcslink without an argument reports the name of the "linked" directory.

In general for the ITC, *remotename* is of the form

usr/local/...

note that there is no leading slash.

-c

This switch should be given only the first time a system directory is referenced. It creates the system directory.

Error exit values
1 – syntax error (Message also printed)
2 – this directory already linked (file RemoteRcsDir exists)
3 – there is no central source directory with the given name

rcsls [–{*ls switches*}]
performs an *ls* on the remote RCS directory linked to the current directory. Normal *ls* switches may be specified on the command line, but file names may not (directories would be inappropriate, wildcards would be expanded on the wrong machine).

Error exit values
1 – tried to give a non-switch argument
3 – **rcslink** needs to be done

rco [–l] *filename ...*
copies files from linus to the current directory. The remote equivalent of **co**, and operates in exactly the same way except that it must be invoked in a local directory which has a RemoteRcsDir.

–l

This switch specifies that the file is to be locked for modification by the requestor.

To check out all the files from the directory on linus, you can write

rco ‘rcsls‘

the backquotes around **rcsls** cause it to be executed and replaced by the list of filenames it returns.

When **rco** prints the message '*done*', it means that the file has been copied from an internal rcs form to source form. But the file is still on linus. **Rco** is really done only when you get the prompt for the next command.

Error exit values
1 – could not create temporary directory on source host
3 – **rcslink** needs to be done

rci [–l] [–t*descriptionfile*] *filename ...*
copies files from the local directory to linus central sources. The remote equivalent of **ci**, and operates in exactly the same way except that it must be invoked in a local directory which has a RemoteRcsDir. The number of characters in the list of filenames is limited; at least fifteen to twenty files can usually be handled.

–l
: This switch specifies that the file is to remain locked for modification by the requestor.

–t*descriptionfile*
: Uses *descriptionfile* as the description for *filename.* Descriptions are stored in the RCS file and can be retrieved with the **rrcs** command. (In an early version of this document I claimed this parameter was required and ignored. It is now not required and never was ignored. It would be good to replace the bogus descriptions stored when I thought it was ignored.)

Rci will prompt you for a log message to be recorded as the effect of this revision; type in a description of your changes. Terminate the description with a final line that contains only a period. The log message can also be supplied as the standard input to **rci**, but the –m switch does not work..

Do not put shell delimiters (double quote, apostrophe, semi–colon, exclamation, or dollar sign) in the log message. However, if you get to where **rci** wants the log entry and suddenly decide not to release the files yet, you can abort by writing a log entry containing an apostrophe. (Yes, this is crude.)

If the file has not been changed an attempt to unlock it by checking it in will fail. The best way to unlock it is with the –u option of **rrcs**.

Error exit values:
- 1 – cannot create temporary directory on source host
- 2 – the checkin operation failed on the source host for at least one file (as per message); others may have been checked in
- 3 – **rcslink** is needed
- 4 – a file to be checked in is non–existent; no checkins have been attempted
- 5 – a file to be checked in is non–existent; no checkins have been attempted

rrcs [–l] [–u] *filenames*
: Set options for *filenames.* The remote equivalent of **rcs**, and operates in exactly the same way except that it must be invoked in a local directory which has a RemoteRcsDir.

(The –t option is not implemented, but can be done by logging in to **linus** and using **rcs** itself.)

–l locks the listed files.

-u unlocks the listed files.

Error exit values:
3 – **rcslink** is needed
5 – the -t switch is not implemented

rrlog [-L] [-R] [-h] [-I] [-l] *filenames*
Give information about *filenames.* The remote equivalent of **rlog**, but must be invoked in a local directory which has a RemoteRcsDir.

-L
only list info for files that are locked (to anybody)

-R
only list the name of the file. Use "-R -L" to get a list of files locked (to anybody). ("-RL" and "-LR" are not the same as "-R -L".

-h
limit the amount of information for each file. (This amount is usually enough.)

-I
gives a table of filenames, people who have locked revisions, and which revisions are locked.

-l
gives a long listing, though it does omit information about revisions that are not locked. Ignored if -R is set.

Error exit values:
3 – **rcslink** is needed

rinse [-h*hostname*] [*–Ttempdir*] [*options*] [*directory*]
performs '**make** *options*' for the system sources from the *directory.* **Rinse** sets environment variables that are used in compilations. For example, the MACHINE variable which specifies what machine this compilation is for. See the complete list in the section on Makefiles and Installation.

options
Can be any options that could be passed to make. Usual default is to make the first entry in Makefile; if -h*hostname* is given, 'install' is always done.

directory
Specifies a subdirectory on linus containing sys-

tem sources; no leading slash, but must contain a slash to distinguish it from an *option*. Example: *usr/local/bin/wm/* If no *directory* is given, the current directory is used and files are not copied from the central system sources. If –h*hostname* is given without *directory*, **rinse** uses the directory established in RemoteRcsDir by **rcslink.**

–h*hostname*
Performs the compilation and installation on host *hostname*. If no *directory* is given, the one set by **rcslink** is used. The 'install' option is always appended as the last of *options*. As a special case, installation on **mother** also does a postnews to *itc.rinsed* to make a record of the system change.

–T*tempdir*
specifies a name for the temporary directory. The default is */usr/tmp*.

Two cases are most common:

1) On a non-server machine you have used **rco** to get some sources and now want to do a 'make'. Use 'rinse' because this sets environment variables needed for the compilations. Since no directory is given, the current one will be used.

2) You wish to install on server HIM the system component whose sources are in *usr/local/bin/foo*. Say
rinse –hHIM usr/local/bin/foo
(the *option* defaults to 'install'). If the current working directory has been linked to usr/local/bin/foo, you can just say
rinse –hHIM

delete or move a file
Sometimes files are created in the wrong place. The only current way to delete files is to send mail to the source code administrator containing the name of the file to delete or move.

"is in use"
If you get the message that an RCS file is in use, it means either that it is actively now being checked in or out or someone has hung during a checkin or checkout process. The in-use flag is a file of zero length whose name begins and ends with a comma. To free up a file locked due to a hang, ask the source code administrator to delete its comma-comma file.

Examples

Reading Existing Code

Suppose you wish to read the code for the **itclogo** program. By running

```
rcsfind itclogo
```

you find that the source is in *usr/local/bin/itclogo*

You make a working directory somewhere on your file space:

```
mkdir /tmp/itclogo
cd /tmp/itclogo
```

Now you link to the central source directory for **itclogo**

```
rcslink usr/local/bin/itclogo
```

and check out a file via

```
rco itclogo.c
```

To check out all the files you can use **rcsls** as a parameter. Delimit it with back-quotes:

```
rco `rcsls`
```

The result will be some read-only files in the current directory. When you have finished reading the code, the entire directory and all its contents may be removed. No remote RCS commands are needed to do this (if you have removed all locks by checking in any files you checked out with locks.)

Changing Existing Code

Suppose that your reading reveals some recondite bug in the **itclogo** command. You wish to check out a writable version of some file, change it, test the modified command, and have your fixes installed permanently. You are starting from the state of having read-only versions of all the files, obtained as described above.

– Obtain a writable version of (say) *itclogo.c* by executing

```
rm itclogo.c
rco -l itclogo.c
```

(If someone else already has *itclogo.c* checked out, this will fail with a message.)

– Edit the file, compile the program, test it. Use

```
rinse
```

to compile it.

– When you are confident of your fix, check *itclogo.c* back in by executing

```
rci itclogo.c
```

You will be asked for a log message describing the fix you have made.

After checking the code in, you must install it on the file servers. It is sufficient to install it on *mother* because */usr/local* is copied nightly from there to *father.* You need to have the server execute a **rinse** of the directory containing the revised facility. This can be done for *usr/local/bin/itclogo* by

```
rinse -hmother
```

when executed in the directory **rcslink**'ed to *itclogo.*

Storing New Code

Suppose you have created a new command called *panacea*, the answer to everyone's problems at the ITC. The source consists of a set of files in a directory somewhere in your file system. It must include a **Makefile** with options for *install* and *system*, as described below. You should also create a description file *df* for each file *f*, describing how this file contributes to the entire package. The description for the Makefile should describe the overall function of the package.

To check in and install *panacea* do the following:

–Create a remote RCS directory:
rcslink –c usr/local/bin/panacea

–Checkin each file by executing for each:
rci –tdf_i f_i

–Install the facility on the server:
rinse –hmother

Deleting source files

Later you decide that panacea is in fact not a solution but an infinite source of new problems. You decide to delete it. The only way to delete source files at present is to send mail to the system source administrator.

Makefiles and Installation

The Makefiles of ITC source serve two functions: installation and system construction. You execute **make** on the Makefile to install a change on our servers, mother, father, and linus. The system-build staff employs the same Makefile with the 'system' option to build an entirely new instance of the system.

When you take extra care to check the Makefile, you are being of tremendous service to the people who have to build the system.

Every directory of source on linus must contain a file called 'Makefile' (capital 'M') which describes how to make the facility from the source pieces. Among the rules of Makefile must be ones which (a) install the facility on servers and (b) build the system from a virgin 4.2 system. To describe these rules, let me first describe the installation process.

A component is installed when it has been placed in the public files provided on the servers. Thus **wm** is installed by placing its object module */usr/local/bin/wm* and a few libraries on father, mother, and any other servers. Note that this is a step beyond placing the source for **wm** into *usr/local/bin/wm/* on linus. Using the **rinse** command, the installation is performed on the server: **rinse** copies the sources, and invokes **make** to recompile the sources and move them to the appropriate directory on the server.

There is a standard command **install** which performs the copy for installation. See the example Makefile's below and the documentation of *install*(8) in the Sun System Manager's Manual. This command should be used instead of **cp** because it handles some conditions better.

Among the rules of each Makefile must be ones for 'install' and 'system'. The install rule should depend on compilation of the component and should copy the object module to the appropriate directory using the **install** command. Saying '**make install**' is expected to install all the files generated by this source directory. For lowest level directories, '**make system**' is exactly the same.

There are also non-terminal directories: *usr/local, usr/local/bin,* ... . The '**make install**' operation in one of these directories installs only the facilities whose source is in the directory itself. The '**make system**' option installs both these local facilities and any components in subordinate directories. For example, '**make system**' in *usr/local* is expected to invoke '**make system**' for each of *usr/local/doc, usr/local/help, usr/local/include,* and so on. The latter will in turn invoke '**make system**' for all their subordinate directories.

At present, an ITC system consists of Unix 4.2 BSD augmented with files in */usr/local/*... . In particular, all facilities go in one of the directories

/usr/local/bin
/usr/local/lib
/usr/local/include
/usr/local/help

/usr/local/doc
/usr/local/fonts

In the future, these directory names may change, so macro names have been defined to refer to these directories. Makefiles should use only the macro names to refer to directories. The names corresponding to the above directories are BINDIR, LIBDIR, INCLDIR, HELPDIR, DOCDIR, and FONTDIR.

The only sure check on the Makefiles is to begin with a bare 4.2 BSD and execute the Makefiles to build the system. Keep this in mind when writing a Makefile so you are sure that all target files are described by the Makefile. (Sometimes the system build will be done into a mounted file structure or some other subordinate directory. For this reason, all references to directories should begin with ${DESTDIR}, a macro which ordinarily is mapped into the null string.)

When you take extra care to check the Makefile, you are being of tremendous service to the people who have to build the system.

When you create a new component and put it in a subdirectory, you are responsible for also putting the name of the subdirectory in the SUBDIR list in the Makefile for the parent. For example, when source directory *usr/local/bin/wm/* was created, an entry was made in the Makefile in *usr/local/bin/.*

Mention every file in the subdirectory in its Makefile. If the file is included for archival purposes only, mention it in a macro definition of IGNORE and give a subsequent comment explaining it's importance.

A typical Makefile that installs one command, say **lsh**, will have this pair of rules:

```
install : lsh
        ${INSTALL} lsh ${DESTDIR}${BINDIR}

system: install
```

Note: ${INSTALL} is used in case some special treatment is needed or we need to use another version of **install**. The Makefile itself should not define INSTALL. Similarly the Makefile should use and not define CC and RINSE.

${DESTDIR} is left undefined. When new systems are being constructed in may be valuable to define DESTDIR so the old system is not overwritten. The Makefile should not define DESTDIR.

Note: ${BINDIR} is used instead of */usr/local/bin* because we may someday wish to change the name of the directory for our binaries. Six such variables are defined; all installs should be into one of these directories or a subdirectory:

BINDIR	/usr/local/bin
LIBDIR	/usr/local/lib

INCLDIR	/usr/local/include
HELPDIR	/usr/local/help
DOCDIR	/usr/local/doc
FONTDIR	/usr/local/fonts

If a Makefile installs multiple modules it can have a rule of this form:

```
install: ${COMMANDS}
        for i in ${COMMANDS}; do ✓
           (${INSTALL} $$i ${DESTDIR}${BINDIR}); ✓
        done
```

In a number of cases, subdirectories of one of the source directories are needed. These are created by the Makefile that puts something in them. In fact, even at the top levels, the Makefiles create the subdirectories they need. Thus the Makefile in *usr/local/bin* creates the directory /usr/local/bin. The 'make system' for *usr/local* performs **make system** in its subdirectories in the order

doc help include lib bin

Each of these can insert material only in subdirectories created by ones that are made earlier.

Even if overwriting the current directory tree, the Makefile in *usr/local* removes the file ${DESTDIR}${LIBDIR}/libitc.a so it will be completely refreshed. Thus the order of making subdirectories of *usr/local/bin* is important; a facility can only depend on library entries that it makes itself or that have been made by preceding subdirectories.

When adding facilities to libitc.a, note that file names to **ar** are limited (eight letters work, and possibly fourteen). The best strategy I found was to put a number of entry points in one file and give that file a short name. The entry point names can be as long as desired.

If your source is adapted for multiple environments via #ifdef's, you need to cause those variables to be appropriately defined by proper CFLAGS in the Makefile. To enable tests for machine type (VAX or SUN), include

-D${MACHINE}

in the definition of CFLAGS. Similarly, if testing for operating system, include

-D${OS}

The following variables are added to the definitions in your Makefile by **rinse**:

MACHINE – "VAX" "SUN=1" "SUN=2"
HOSTTYPE – SUNUSER SUNHOST VAXHOST VICEHOST
OS – BSD42 BSD41c
LJOBS – "" or "-ljobs"
(To get the 4.2 jobs compatibility package for 4.1c)
BINDIR, LIBDIR, INCLDIR, HELPDIR, DOCDIR, FONTDIR
CC – cc
RINSE – /usr/local/bin/rinse

INSTALL –install

Our version of **make** uses only the Bourne shell, **sh**, so **make** rules must use **sh** syntax. In particular, the higher level Makefiles in *usr/local/...* use "**for** ... **do** ... **done**" rather than "**foreach** ... **end**".

Examples

Example program – hello.c from usr/local/doc/

```
/* hello – minimum C test
 *
 *       Author: John H. Howard
 *
 *       Illustrates the standard structure of ITC code
 */

#include "itc.h"
private char rcsid[] = "$Header: hello.c,v 1.2 84/03/16 11:25:12 jhh Exp $";
private char IBMid[] = "Property of IBM Corporation";

/* $Log:      hello.c,v $
 * Revision 1.2  84/03/16  11:25:12  jhh
 * Add "there" to message, remove trash quote from log.
 *
 * Revision 1.1  84/03/16  09:55:28  jhh
 * Initial revision */

#include <stdio.h>

main(argc, argv)
in int argc;           /* number of args including name */
in char *argv[];               /* argument array */
{
        printf("Hello there, world!\n");
}
```

Standard ITC header file – itc.h in usr/local/include/

```
/* itc.h – ITC standard header file
 *
 *      Author: John H. Howard
 *      Property of IBM Corporation
 *
 *      defines a few constants used throughout ITC code
 *
 *      $Header$
 */

/* $Log$
*/


#ifndef     _ITC
#define     _ITC

/* C language patches */
typedef     int boolean;    /* to declare switches */
#define     private static  /* for variables local to a module */

/* parameter usage description tags */
#define     in              /* argument not modified */
#define     out             /* initial value not used; arg modified */
#define     inout           /* initial value used and arg modified */

/* very commonly used values */
#define     TRUE        1       /* for boolean variables */
#define     FALSE       0       /* ditto
#define     NULL        0       /* to point nowhere */

#endif _ITC
```

Makefile for a command – Makefile in usr/local/bin/lsh/

```
#   lsh  – window manager file organizer
#
#       Author: WJHansen, ITC
#       Property of IBM Corporation
#
# Installs /usr/local/bin/lsh
# and     /usr/local/help/lsh.n
#
# This file cannot be released until after 'wmlib.*'
# $Header: itcsource.d,v 1.4 84/09/12 22:56:24 wjh Exp $

PARTS=lsh.o lsbody.o makemenu.o
CFLAGS = -c -I${DESTDIR}${INCLDIR} -D$(MACHINE)

lsh: $(PARTS)
        ${CC} $(PARTS) ${DESTDIR}${LIBDIR}/libitc.a $(LJOBS) -o lsh

lsh.o: lsh.c lsh.h
        ${CC} $(CFLAGS) lsh.c

lsbody.o: lsbody.c lsh.h
        ${CC} $(CFLAGS) lsbody.c

makemenu.o: makemenu.c
        ${CC} $(CFLAGS) makemenu.c

install: lsh
        ${INSTALL} lsh $(DESTDIR)$(BINDIR)
        ${INSTALL} lsh.n $(DESTDIR)$(HELPDIR)

system: install
```

Makefile for a library **–wmlib** in **usr/local/lib/wmlib**

```
#  wmlib –library of routines for using raw windows.
#       See wmpgm.c for examples of use.
#
# Author: WJHansen, ITC
# Property of IBM Corporation
#
#       Installs wmlib.o into itclib.a
#       Also installs wmlib.h in /usr/local/include/
#
# $Header$

CFLAGS = -c -I${DESTDIR}${INCLDIR} -D$(MACHINE)

wmlib.o: wmlib.c
        ${CC} $(CFLAGS) wmlib.c

wmpgm:      wmpgm.o
        ${CC}  wmpgm.o  ${DESTDIR}${LIBDIR}/libitc.a  $(LJOBS)  -o
wmpgm

wmpgm.o: wmpgm.c
        ${CC} $(CFLAGS) wmpgm.c

install: wmlib.o
        ${INSTALL} wmlib.h $(DESTDIR)$(INCLDIR)
        ar r $(DESTDIR)$(LIBDIR)/libitc.a wmlib.o
        ranlib $(DESTDIR)$(LIBDIR)/libitc.a

system: install
```

In a prior version of this file, I did a recursive make of **wmpgm** in order that *wmlib.h* could already have been installed in */usr/local/include.* This is no longer necessary because I use

#include "wmlib.h"

in *wmpgm.c* so the local version is taken rather than the one from */usr/local/include.*

A system Makefile – in usr/local/bin

```
#       Makefile  for usr/local/bin – source storage for ITC programs
#
#       Author: WJHansen, ITC
#       Property of IBM Corporation
#
# $Header: itcsource.d,v 1.4 84/09/12 22:56:24 wjh Exp $

CFLAGS=   -O -D$(MACHINE) -I$(DESTDIR)$(INCLDIR)

# Shell scripts that need only be installed and are never removed.
#
SCRIPT=    print

# C programs that live in the current directory and do not need
# explicit make lines.
#
STD=dd pr prmail kermit


# Subdirectories that have source for commands that must be made
# when we 'make system'.  Note that there is order dependency:
# wm must be made before others because it creates library entries
# used by others.
#
SUBDIR = wm xyzzy Mail advise dvisherpa emacs filesys grok ✓
      head indent itcrcs logo lsh make pic pressdvi preview ✓
      rcs style telnet tftp troff itclogo


# 6670 header library files (used for print)
#
LIB6670=listings portrait

#
# Man files
MANS=dd.1 pr.1 kermit.1

all:    ${STD} $(NSTD)

${STD}:
        ${CC} ${CFLAGS} -o $@ $@.c



install: all
        for i in ${STD} ${NSTD}; do ✓
                (${INSTALL} $$i ${DESTDIR}${BINDIR}); done
        for i in ${SCRIPT}; do ✓
                (${INSTALL} -c $$i.sh ${DESTDIR}${BINDIR}); done
        -mkdir ${DESTDIR}${LIBDIR}/667
        -chmod 777 ${DESTDIR}${LIBDIR}/6670
        for i in ${LIB6670}; do ✓
```

```
		(${INSTALL} -c $$i.header ✓
			${DESTDIR}${LIBDIR}6670); done
	for i in ${MANS}; do ✓
		(${INSTALL} -c $$i ${DESTDIR}${HELPDIR}); done
	for i in ${MANS}; do ✓
		(${INSTALL} -c $$i ${DESTDIR}/usr/man/man1); done


# Note that "usr/local/bin/" in the next rule
# refers to the central source directories
system: makedir install
	for i in ${SUBDIR}; do ✓
		($(RINSE) system usr/local/bin/$$i); done

makedir:
	-mkdir ${DESTDIR}${BINDIR}
	-chmod 777 ${DESTDIR}${BINDIR}
```

Checklist for Makefiles

Makefiles need special attention because some features will have no impact until we try to make an entire system. Please aid the system-build staff by checking each Makefile against these items.

–Name is "Makefile".
–Is a member of SUBDIR list in parent.

– Initial comment contains name of directory, Author, "Property of IBM Corporation", description of the component, and $Header$.

–CFLAGS includes -D${MACHINE} if needed.
–CFLAGS has -I${DESTDIR}${INCLDIR}.

–Uses ${DESTDIR}${LIBDIR}/libitc.a (rather than usergraphics.a).

–Does not define CC, INSTALL, RINSE.
–Uses CC, RINSE, INSTALL rather than program names.
– Calls programs from ${DESTDIR}${BINDIR} and not from /usr/local/bin.

–Has rules for 'install' and 'system' something like these for *panacea*

```
      install: ${ALL}
              ${INSTALL} panacea ${DESTDIR}${BINDIR}
      system: install
```

–Omits -s on the INSTALL.

–Prefixes mkdir and chmod with dash.
–Targets mkdir to ${DESTDIR}${xxxxDIR}/...
–After a mkdir, does chmod 777 on same directory.

– Mentions all files in the subdirectory (even if some are listed in the IGNORE list).
–In high level directories uses lists named STD, NSTD, SCRIPT, SUBDIR, MANS, MKSDIR (see Makefile for *usr/local/bin*).

```
$Log: itcsource.d,v $
Revision 1.4  84/09/12  22:56:24  wjh
. changed rinse -t to -T. pass all unrecognized rinse args to make. -h implies
install, but other rinses do not. sourcedir is taken from RemoteRcsDir where
this makes sense

Revision 1.3  84/09/03  21:55:27  wjh
. Minor revisions throughout.

Revision 1.2  84/05/30  15:54:05  wjh
 . complete revision

Revision 1.1  84/05/15  15:12:27  wjh
Preliminary version.


$Header: itcsource.d,v 1.4 84/09/12 22:56:24 wjh Exp $
```