To: MTB Distribution

From: Chris Jones

Date: 20 January 1981

Subject: New Installation Tools

Introduction

This MTB is a revision of MTB-459. While the purpose of the tools described is the same as that described in MTB-459, there has been a crucial change in the output of the tools. It is not necessary to have read MTB-459 to understand this MTB.

Problem Definition

When a Multics programmer has completed the development cycle, there still remains the unautomated task of getting the change installed. This involves several potentially time-consuming tasks, including ensuring that the library maintainer has the correct access to the changed segments, determining that the segments were compiled with the correct options and with the correct version of the compiler, notifying the library maintainer of the changes to be made, and other such mundane tasks. It is also necessary to fill out a set of forms: one summary (yellow) form and one or more detail (orange) forms, one for each bound segment. Every source, include, bindfile, info, etc. segment which is part of the submission must be listed. This entire task is time-consuming and subject to error, and can better be done by tools.

Requirements of a Solution

The task (or series of tasks) described above can essentially reduced to two simpler tasks:

- filling out the yellow (summary) form
- invoking some tools to do the rest of the work

The continued use of the summary form is required for two reasons:

- to provide a permanent written description of the change
- to provide a signoff mechanism for auditors and project controllers

Multics Project internal working documentation. Not to be reproduced or distributed outside the Multics Project.

The present requirements of the library installation tools are as follows:

- that the submitter provide correct access
- that the components of the submission have been compiled with the correct options and with the installed compiler
- that the data specified on the forms be correct, i.e. the components of the submission are where the forms indicate

In addition, any automated tools should provide such checks as:

- object/source correspondence
- modification of the source more recently than the object
- identically named segments in different archives

If such tools are to be useful, they must function independently of how the data to be submitted is organized and stored. Thus they must resolve links and be prepared to deal with archives as well as free-standing segments. They must also be prepared to deal with the many different types of material now submitted to the libraries, such as ec's, info segments, sources, and objects. Lastly they must provide a reasonable interface to the user, the installer, and the library tools.

Proposed Solution

This MTB proposes a set of such tools. Their assumptions are as follows:

- They incorporate the concept of an "installation directory". This directory contains all the segments required for a single installation and/or links to them. Segments may or may not be archived, at the user's option.
- These tools will not actually prepare the detailed form; instead they will build a data base (an rdc translator source segment) from which the installer can find all the information required.
- These tools are intended to automate the normal installation process and provide warnings and cross-checks wherever conveniently practical. They are not intended to provide verification that data can be translated (i.e. compiled or bound without error), or that info segments actually describe that which they purport to.

Extensions and Future Issues

Once these tools have been accepted and are in general use, an effort should be made to integrate their output data base with the library maintenance tools so that the installer's work is more completely automated. In addition, other more exotic cross-checks can be added as a need is perceived for them. These new checks could include:

- binder verification
- compilation verification
- format verification using format_pl1
- source program checks for out-of-date functions
- copyright notice verification
- STI handling

Program Descriptions

There are actually three programs described here. The first is merely a tool to set up a data segment, and is invoked rarely (usually when the library maintainer changes). The second and third programs do all of the work described previously.

create_pinst_data

This program is used to create a data segment (named pinst_data_) which contains three entries: access_name, lib_maint, and lib_desc. The first is an access name; this is a person-id which must have "r" access to all components of the installation and "s" access to the installation directory and its parent directory. The second is the name of the person to whom mail is to be sent requesting that the installation information be processed. The last is the name of the library descriptor to use when determining where the various entries in the installation directory belong. If this descriptor is the null string, it designates the default library descriptor. An example of its use is shown below (the exclamation points serve to denote items typed by the user--they do not appear during an actual invocation):

```
create_pinst_data

create_pinst_data:  Access name? !  JC.SysMaint.*

create_pinst_data:  Library maintainer? !  JAnderson.SysMaint

create_pinst_data:  Library descriptor? !
```

prepare_installation

This tool actually does most of the work described previously. It can deal with object, info, bind, pl1, alm, fortran, lisp, cds, rd, et, cobol, ted, teco, and table files, and with include files for pl1, alm, and lisp. It checks for necessary access and

sets it if requested (via the -force control argument). It checks to see that unarchived segments are identical to those with the same name in an archive. It checks for source-object correspondence both by name and by date modified. In addition, it checks for correct pl1 compilation options (unless prepare_installation is invoked with -brief, the absence of -optimize or the presence of -profile or -table is flagged as an error). Its MPM writeup follows:

Name: prepare_installation

The prepare_installation command builds an rdc source segment containing all the information needed by a library maintainer to install a system change. It works with a directory called the installation directory. All of the components of an installation either must be in this directory, or have links in this directory to them. prepare_installation figures out which segments are to be installed, ensures that both source and object segments are present, that the object segments were created after their corresponding source segments were last modified, and that the correct compilation options were used in the case of pl1 compilations. It creates its output in a segment in the installation directory. If the installation directory's entryname is "foo", the segment will have the name "foo.install". If foo.install already exists, it will be overwritten.

The output of prepare_installation is an ascii segment which is later parsed by a reduction compiler translator. Its syntax is described below. After this phase is complete, the user has the choice of editting this segment to fill in unknown library names, specify addnames, add comments, etc. When this step is completed, another program is invoked which checks this segment, assigns a unique ID to this submission, and sends mail to the library maintainer informing him or her of the pending installation.

The ascii segment output by prepare_installation is in a form which can be translated by a reduction compiler translator. It consists of a header followed by a series of statements describing files to be installed and where they are to be installed.

The format of the header is particularly rigid. This means that when you are editting the segment, you should not change the header. While changing the header may not cause any problems, it could, and there is no reason to change it anyway, so don't touch it! The header consists of five statements in a specified order. See the example below for the form of the header. The meaning of most of these fields should be obvious. The ID statement contains a template ID (the string "XXXXXXXXXXXX") which is overwritten by the submit_installation command when it assigns

the ID. The Descriptor statement has the name of the descriptor which was used to process the submission.

After the first five statements of the header (which are always present), there is an optional group of statements which tell the names of any ascii segments which convey instructions to the installer (e.g. those specified by -add_name and -misc). The syntax of these statements is:

```
<info_stmt> ::= <keyword>: <pathname>;
<keyword> ::= Addname_info | Delete_info | Misc_info
```

The body of the ascii segment consists of sequences of statements which describe what goes where in this submission. A BNF description of it is:

```
<body> ::= <group> [, <group>] ...
<group> ::= <in_stmt> [<install_group> [, <install_group]]
<install_group> ::= { <install_stmt> [<from_stmt>]
          [<addnames_stmt>] | <delete_stmt> }
<in_stmt> ::= In: <pathname>;
<install_stmt> ::= install: <name>;
<from_stmt> ::= from: <name>;
<delete_stmt> ::= delete: <name> [, <name>]...;
<addnames_stmt> ::= addnames: <name> [, <name>]...;
```

The prepare_installation tool will write the "In", "install", and "from" statements. The user will have to add any "addnames" and "delete" statements which are necessary (normally none will be necessary). In addition, if prepare_installation is unable to find a given segment in any library, it will output an "In" statement with a name of "****", as an aid in denoting which library names must be supplied by the user. This would most likely occur when a new segment is being added to a library.

Usage

```
prepare_installation {path} {-control_args}
```

where:

1. path
   is the pathname of the installation directory. This is the directory where all sources, archives, objects, etc. (and/or links to them) are located. The "add", "delete", and "misc" ascii text segments (described below) are also in this directory.

It is also the directory into which the rdc source segment describing the installation will be put.

2. control_args
can be one or more of the following:

-force, -fc
If this control argument is used, prepare_installation will try to ensure that <pinst_data_$access_name> has the required access by modifying the ACLs (as necessary) of the installation directory, its parent, and the segments in the installation directory which are to be installed.

-check, -ck
specifies that no rdc source segment will be created

-brief, -bf
specifies that warning messages about invalid compiler options and date mismatch between the source and object will not be printed.

-add_name ename
specifies a free-form ascii text segment containing instructions to the installer concerning names which are to be added, deleted, or changed.

-misc ename
specifies a free-form ascii text segment which contains miscellaneous instructions to the installer.

-delete ename, -dl ename
specifies a free-form ascii text segment containing instructions to the installer concerning segments to be deleted as part of this installation.

-access_name ACCESS_NAME, -an ACCESS_NAME
gives an access name list to be used instead of the one in pinst_data_$access_name_list.

-lib_desc Descriptor_name
specifies a library descriptor to be used instead of the one in pinst_data_$lib_desc

Example

```
ls -a -sh example>**

Segments = 1, Lengths = 1.

r w    1  misc_instructions

Directories = 3.

sma  incl
sma  o
sma  s

Links = 7.

incl.archive  >udd>m>clj>pinst>example>incl>incl.archive
get_io_segs  >udd>m>clj>pinst>example>o>get_io_segs
iom_manager  >udd>m>clj>pinst>example>o>iom_manager
print  >udd>m>clj>pinst>example>o>print
get_io_segs.pl1  >udd>m>clj>pinst>example>s>get_io_segs.pl1
print.alm  >udd>m>clj>pinst>example>s>print.alm
iom_manager.alm  >udd>m>clj>pinst>example>s>iom_manager.alm

r 09:05 0.192 0

prepare_installation example -misc misc_instructions
print was found in more than one library.  Pick one.
Enter a number from 1 to 3.
1        >ldd>sss>object>bound_fscom1_.archive::print
2        >sss>bound_fscom1_
3        >ldd>bos>object>print
3
r 09:08 50.668 1075

pr example>example.install

                    example.install     12/22/80  0908.5 est Mon


Version: 1;
ID: XXXXXXXXXXX;
Submitter: CLJones.Multics;
Date: "12/22/80  0908.1 est Mon";
Descriptor: multics_libraries_;
Misc_info: >user_dir_dir>Multics>CLJones>pinst>example>misc_instructions;

In: >ldd>hard>source>i7.archive::iom_manager.alm;
install: iom_manager.alm;
```

```
In: >ldd>bos>source>print.alm;
install: print.alm;

In: >ldd>hard>source>g1.archive::get_io_segs.pl1;
install: get_io_segs.pl1;

In: >ldd>bos>object>print;
install: print;

In: >ldd>hard>bc>bound_iom_wired.archive::iom_manager;
install: iom_manager;

In: >ldd>hard>bc>bound_temp_1.archive::get_io_segs;
install: get_io_segs;

In: >ldd>bos>include>bosequ.incl.alm;
install: bosequ.incl.alm;
from: incl.archive;

In: >ldd>bos>include>bos_tv.incl.alm;
install: bos_tv.incl.alm;
from: incl.archive;


r 09:08 0.249 16
```

submit installation

Name: submit_installation

The submit_installation command finishes the job of preparing a submission. It is invoked after prepare_installation has completed and any editing has been done on the rdc source segment. It checks the rdc source segment for correct syntax, and if there are no errors, assigns a unique ID number to the submission, and sends mail to the library maintainer.

Usage

submit_installation {path} {-control_args}

where:

1. path
   is the pathname of the installation directory. This is the directory where the rdc source segment is located. If this

argument is omitted, the current working directory is assumed instead.

2. control_args
can be the following

-lib_maint Person_id
gives an installer name to be used instead of the one in pinst_data_$lib_maint.

Example

This example is the second half of the submission prepared in the example given for prepare_installation, above. After submit_installation has run (and sent mail to the library maintainer), the beginning of the installation segment is printed to show that the ID field has been updated.

```
submit_installation example
r 09:08 0.623 36

pr example>example.install 1 5
Version: 1;
ID: 15          ;
Submitter: CLJones.Multics;
Date: "12/22/80  0908.1 est Mon";
Descriptor: multics_libraries_;
r 09:09 0.185 1
```