Data Driven TeamConnection user exits

Document Number TR 29.xxxx

Lee Perlov

TeamConnection/CMVC Development
IBM Software Solutions
Research Triangle Park, North Carolina

ABSTRACT

This technical report features a user exit program sample, **mue** (multiple user exits), that demonstrates the capabilities of a new TeamConnection concept: the Environment File. The mue program demonstrates how the environment file provides greater reliability and more direct access to data.

ITIRC KEYWORDS

- TeamConnection
- User Exit
- mue

ABOUT THE AUTHOR

LEE R. PERLOV

Mr. Perlov is a Staff Programmer in the TeamConnection/CMVC development group. He started working for IBM in 1985 in Gaithersburg, Md, working in the Federal Systems Division on various projects for the United States intelligence community. He then moved to RTP to work on library development and support.

Mr. Perlov received a B.S.Acc degree in Accounting from the University of Florida in 1983. He also completed two years of graduate work in the Department of Computer Science at the University of Florida.

CONTENTS

FIGURES

INTRODUCTION TO MUE

This technical report features a user exit program sample, **mue** (multiple user exits), that demonstrates the capabilities of a new TeamConnection concept: the Environment File. The mue program demonstrates how the environment file provides greater reliability and more direct access to data. This code:

- Uses the value of parameters delivered to the user exit, plus the content of the **config/multExit** configuration file, to determine whether a particular user exit needs to be run.
- Allows multiple user exits to be run and the results of the previously run user exit's return value to be considered before running the current user exit.
- The config/multExit file is very similar in format to the **config/userExit** file, making it easy to configure.

What makes mue possible is a new TeamConnection concept of user exits with version 2.0, the Environment File. Through a new field, ENV=(), in the config/userExit file, a family administrator may specify individual user exit parameters by name, to be delivered as binary data (parameter name and value) in an environment file. This file is easily parsed to allow for direct access to data, without the difficulties associated with parsing a parameter list.

DELIVERED AS SOURCE CODE

Since every family administrator has different goals, mue is delivered as source code so that the code can be customized or enhanced to fit the needs of the environment.

In order to help the family administrator take full advantage of TeamConneciton user exits, the mue program includes a significant amount of code copied directly from the TeamConnection source and the values of some critical variables used by TeamConnection user exits.

Also included in this technical report is the source code to the **viewexit.c** sample. This program displays the output of all parameters and environment file values delivered to a user exit by the TeamConnection teamcd process. When viewexit is registered in the config/multExit file for a user exit, it is possible to see the full range of what TeamConnection user exits can do for you.

This code was not written to be particularly clever or sophisticated. As such, almost any family administrator with basic C skills should be able to customize this code as needed.

INSPIRATION FOR MUE

I would like to thank Ray Morin for inspiring the development of this code. He and his team in Austin, formerly of Boca Raton, developed a very sophisticated data driven user exit architecture for a highly customized IBM Internal version of CMVC: CMVC95. Unfortunately, this architecture was more powerful, flexible and complicated than is appropriate for administering TeamConnection families. The CMVC95 user exit architecture did demonstrate what is possible with user exits and provide the inspiration for this code.

USING DATA DRIVEN USER EXITS

WHY USE MUE

Background on user exits

User exits are provided in a wide range of products. They allow a user to access data and perform actions at various stages during the execution of a tool. CMVC and TeamConnection provide 4 points in the execution of most client commands (actions) where a user exit may be called. At any of these 4 points, the CMVC/TeamConnection administrator may register a program to be executed as a user exit. CMVC/TeamConnection then passes a fixed set of positional parameters to the user exit program. A user exit program is a command followed by positional parameters, run in a command processor shell of the operating system. These user exits run on the CMVC/TeamConnection server when a user issues a CMVC/TeamConnection action.

Note: TeamConnection is a follow-on product to CMVC. The CMVC user exit architecture was migrated to TeamConnection, then extended by the enhancement described in this document.

Problem with positional parameters

A problem arises when one of the positional parameters contains something other than simple text (e.g. a newline or quote character), or when the length of the parameter list is greater than the command string length allowed by the command processor of an operating system. Unless special measures are taken, a newline ends the command line and truncates all data that follows the newline character. Also, some characters will disrupt the processing of parameter lists by the program run in the command processor shell. Further, once the line length limit of the command processor is reached all subsequent characters are truncated.

While it is possible to "escape" most characters so that they do not terminate the user exit program invocation in Unix, this is much more difficult on OS/2 and Windows NT operating systems. Also, there is no way to extend the length of the string passed to the command processor during invocation of a user exit.

If the values of the parameters will be too long for the command line that invokes the user exit, fields and/or the entire line are selectively truncated, with each truncation indicated by ellipses (...). The included user exit sample in Appendix A, “viewexit Source Code” on page 21 detects ellipses and warns that truncation has occurred. For Windows NT the limit is currently 1024, and Unix is limited to 32,000. Once the limit is reached, ellipses are inserted and the remainder of the user exit string is truncated. Given these limits, individual values are truncated to 400 characters in OS/2 and Windows NT, and 16,000 for Unix.

As a result, it is difficult to deliver all of the positional parameters on a user exit invocation successfully to the registered user exit program.

The Environment File

The envFile is a temporary file containing the name and value for all parameters specified by the administrator in the user exit definition. For each user exit invocation where parameters are specified, a temporary envFile is generated and the name of the file is passed to the user exit as the second positional parameter. Therefore, the user exit program has direct access to the data in the envFile. So, parameters with newlines, quotes, etc. that might truncate the positional parameters are delivered unaltered to the user exit through the envFile.

The name of each parameter is now known to TeamConnection. This allows the administrator to specify each parameter to be passed in the envFile by name.

One of the parameters that is of particular interest to administrators is something called "configurable fields". There are a predetermined number of configurable fields in TeamConnection. An administrator can turn these fields on or off, as well as change their names. In CMVC, all configurable fields that are specified by users (when they invoke a CMVC action) are passed as a collection of values in a single string to the user exit. The TeamConnection enhancements allow the administrator to get the value of any configurable field by name (just like any other parameter). When the administrator checks for the value of a specific configurable field in the environment file, he or she gets a zero length value if the user did not specify that configurable field when the TeamConnection action was invoked.

TeamConnection is packaged with sample user exits viewexit.c, viewexit.ksh, viewexit.cmd and teamcenv.c to demonstrate how to access the environment file. Specifically, teamcenv.c is a tool that searches the environment file for a specific parameter, returning the value of that parameter.

How mue helps

The mue program allows you to create smaller, self-contained user exit programs that are invoked separately by mue based on the conditions specified in config/multExit. It also demonstrates the capabilities of the environment file in TeamConnection.

However, specifying multiple userexits and allowing for conditional execution of these exits is not a commonly required feature. As such, mue.c is not currently packaged with TeamConnection.

HOW TO SET UP USER EXITS

The family administrator decides what actions require user exits. These are documented in the Administration Guide, and are also available through the TeamConnection "teamc report -userExitInfo" action.

The complete user exit architecture for TeamConnection is documented in the **TeamConnection Administrator's Guide**, SC34-4551. This document is included in PostScript format with the TeamConnection server. It may also be useful to compare the TeamConnection manual to **CMVC Server Administration and Installation**, SC09-1631-02.

What are userexits and multiple userexits

In TeamConnection, users issue actions through the TeamConnection client. Family administrators may register user exits for these actions in order to add their own processing on top of that already done by the TeamConnection server. Once a user exit is registered for a particular action, the registered user exit program is called each time the user issues that action. The user exit program receives a parameter list containing data necessary for user exit program processing.

For example, the family administrator can add a user exit to enforce coding standards when a user creates a new part (file). The user exit program can either change the contents of the file or abort the TeamConnection action and return an error message to the user.

The family administrator decides what parameters will be included in the envFile that will be passed to the user exit program. This information is also available from the Adminstration Guide or "teamc report -userExitInfo".

The family administrator registers a user exit program by adding a line to the config/userExit file. The format of this file is:

```
ActionName ExitId# "UserDefinedParm" UserExitProg ENV=(parameters,...)# Comments
```

These parameters are defined as follows:

Action The user action name (e.g. PartAdd)

ExitId# The point in the server's processing of the action where the user exit program will be called (0,1,2,3)

"UserParm" The user defined parameter string passed as one of the parameters in the parameter list to the user exit program. The quotes are required.

UEProg The family administrator's user exit program called by the TeamConnection server for the action and exitId#

ENV=(parameters,...) The list of parameters, specified by name, to be included in the envFile when it is passed to the user exit program

 # Comments Comments are for documentation in the config/userExit file

For example:

```
PartAdd 0 mue "PartAdd 0"
PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target)
PartDelete 0 viewexit ""
```

The family administrator writes a user exit program that will open and read the temporary envFile generated for each call to that user exit program. The sample program, viewexit, has a routine that displays all of the parameters and the values that are contained in the envFile that is passed to it. A second sample, teamcenv, allows an envFile to be searched for a particular parameter name, returning that parameter's value.

Using environment files

Here is how an envFile is delivered to a user exit program and used:

1. When the TeamConnection server is started it loads the contents of the config/userExit file,
2. A user issues an action for which a user exit program is registered,
3. The TeamConnection server gathers up the information required by this user exit, including putting the parameters specified in the ENV=() statement, and their values, into a newly created temporary envFile,
4. The user exit program is called, passing the user defined parameter string, the name of the envFile and all of the parameters defined for this user exit,
5. The user exit program opens the envFile, searches for a parameter by name, reads the length of the parameter value, then reads the value.

HOW TO SET UP MUE

The mue program demonstrates data driven logic that is based in large part on the value of parameters in the envFile. This shows how a user exit can use the envFile to perform complex tasks reliably. The code reads the userExit file, and a file specific to mue, **config/multExit**, that contains conditions for determining whether a user exit program will be executed.

The mue program has a usage message, options for verifying the configuration files and of course the default where mue is called as a user exit.

Usage message and error messages

The usage message for the mue program provides a quick and concise overview of the capabilities of this user exit program. Use **mue -?** to display this help message:

```
Multiple User Exit usage:
 1. Called from command line:
    mue -?         // Display complete help message
    mue -C [a|m|u] // Checks userexit and multexit files;
                   // where 'm' verifies config/multExit file, 'u' verifies
                   // config/userExit, and 'a' verifies all (m+u).
 2. Registered as TeamConnection user exit (\config\userExit):
    ActionName ExitID mue "ActionName ExitID" ENV=(...) # Comment
  Example: PartAdd 0 mue "PartAdd 0" ENV=(component,release)
  Registration of conditional exits in MultiUser file (\config\multExit):
    ActionName ExitID ExitName "Conditions" # comment
  Example: PartAdd 0 viewExit "RC>0,component!=NULL"
  Acceptable conditional expressions:
    envName > Number;... // Using envName (where envName is in ENV=()),
    RC      <   :        // or RC (where RC is highest return code from
            =   :        // exits).  Multiple conditionals are separated
            <>           // by commas. Numeric compares convert strings to
           == String     // numbers.  String compares do NOT allow spaces
           != NULL       // and does NOT use quotes.
           in String,String...
NOTE: "mue -C" searches config/userExit for "mue" in lower case.
```

Figure 1. Usage message from command: mue -?

Since the usage message is fairly long, taking up a full screen in most command windows, there is a shorter version that is displayed when there is an error invoking mue. For example, when entering an incorrect option such as **mue -x**:

```
Error: Incorrect number of parameters, or incorrect parameter options.
 Contact your system administrator to insure this program is being called
 by teamcd from a user exit definition.
Multiple User Exit usage:
 1. Called from command line:
    mue -?         // Display complete help message
    mue -C [a|m|u] // Checks userexit and multexit files;
                   // where 'm' verifies config/multExit file, 'u' verifies
                   // config/userExit, and 'a' verifies all (m+u).
 2. Registered as TeamConnection user exit (\config\userExit):
    ActionName ExitID mue "ActionName ExitID" ENV=(...) # Comment
```

Figure 2. Error message with "short" usage from invalid command: mue -x

Using report information

The mue program makes use of TeamConnection's ability to report available parameters for each action, including configurable fields. The TeamConnection **teamc report -userExitInfo -long** client command (with the "-long" option), reports the user exits actually configured after the list of all the available user exits. So, once the config/userExit file is verified using mue, it can be loaded and verified through teamcd.

An administrator (or anyone with super user authority) can see the list of available parameters using the TeamConnection command:

```
teamc Report -userExitInfo
```

An administrator (or anyone with super user authority) can see the list of configured user exits (with keywords to be passed in the environment file) with the TeamConnection command:

```
teamc Report -userExitInfo -long
```

Conditional Expressions

Conditional expressions are where mue provides unique flexibility. Of course a family administrator can put "if" statements at the beginning of each user exit to decide whether to do further processing. In most cases, that probably makes more sense than using mue. As I have already stated, not many administrators really need mue, but if it can save time and effort give it a try.

The left side of the equation can either use "RC", or the name of a parameter in the envFile. If "RC" is specified, the numeric value of the highest return code from previous user exits (invoked at this exitId and this action) is used. Otherwise, the envFile is read and the parameter's value is used.

The right side of the equation can either be a "string", a number, or NULL (i.e. a null terminated string of length zero). The set operator, (), allows the parameter to be compared to a list of strings.

To keep things simple, there are unique operators for strings and numbers. Leading and trailing spaces are removed.

Finally, you can use more than one expression on any line in the config/multExit file, separated by semi-colons.

Here is an example of everything at once:

```
PartAdd  2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207"
```

String comparison

Rules for string comparisons:

- There are different operators for string and numeric expressions. Strings may either be equal (==) or different (!=)
- The left side of the expression may ONLY BE a parameter name.
- The comparison is for the full length of the string
- If the right side is NULL, the environment variable is checked for a NULL value. If the user did not specify a value or no value was previously set, then there will be a null value for the parameter (on the left side).

Numeric comparison

Rules for numeric comparisons:

- Only numeric comparisons can use RC on the left side of an expression
- A parameter's value on the left side of an expression is converted to a number using sprintf. However, the limit is the compiler's maximum for a "signed int".
- The value on the right side can be positive or negative
- Numeric expressions CANNOT use NULL

String in a set

Set rules are the same as string rules, except:

- The comparison is only for a match
- Each value on the right side is separated by a comma
- One of the strings on the right must match the value of the string on the left
- NULL is not valid on the right side

Handling return codes when calling multiple exits

The data driven options include the return codes from user exits as well as the values of the parameters in the envFile. For example, if the config/multExit file contains three user exits for the TeamConnection PartAdd action on ExitID 2, the family administrator probably wants to decide whether the second and third exit should run when the first user exit returns a non-zero return code. Therefore, mue accumulates the highest value returned from any previous call (in this user exit call from teamcd). The code assumes that a higher value for a return code is a more severe error.

Verification of user exit configuration

The teamcd process verifies the contents of the config/userExit file each time the daemon is started and the values are read in. Considering the complexity of the config/multExit file, verifying that everything is correct is important. Therefore, mue has a rich set of configuration verification features.

Contents of config/multExit file

Using **mue -C a** or **mue -C m**, a family administrator can verify that:

- Each entry in the config/multExit file is syntactically correct.

  For example, a call to viewexit.exe on "PartAdd 2" must at least be formed as follows:

  ```
  PartAdd     2  viewexit ""
  ```

  Further, if there is a condition, it must be valid and in the right spot, like these:

  ```
  PartAdd     2  viewexit "RC > 0"
  PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 "
  PartAdd     2  viewexit "RC=0"
  ```

- Each call to mue in the config/userExit file is syntactically correct, since teamcd verifies that each entry in config/userExit has the basic formatting correct, such as:

  ```
  PartAdd 2 mue ""
  ```

  For mue to work correctly, the user defined parameter string must pass the action and exitID, so the format must be:

  ```
  PartAdd 2 mue "PartAdd 2"
  ```

  For the entries in the config/multExit file above, the call in config/userExit also needs to have the correct ENV=() values:

  ```
  PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target)
  ```

- There is at least one user exit call in the config/multExit file for every call to mue in config/userExit.

  For example, if there is an entry in config/userExit for "PartAdd 2", then there needs to be at least one "PartAdd 2" exit in config/multExit.

Contents of config/userExit file

Using **mue -C a** or **mue -C u**, a family administrator can verify that:

- Each line in config/userExit is syntactically correct
- Each parameter specified in ENV=() is valid

When a family administrator is updating the config/userExit file, for example adding a call to mue, it can be time consuming and frustrating to stop and restart the teamcd daemons to find out if the configuration is correct. Further, after the daemons are up and running, it is nice to know that everything is set up the way the administrator intended.

TeamConnection has added the **teamc report -userExitInfo** client command that lists all of the actions that allow user exits, as well as the regular and configurable parameters that are passed to the user exit. Of course, each parameter listed can be specified in ENV=().

Notes:

1. At least one teamcd process must be running in order for mue to call "teamc report -userExitInfo".
2. A user must have super user authority to run "teamc report -userExitInfo".

Customizing mue

As stated before, the source code is provided so that family administrators can customize mue to their needs. This is also true of viewexit.c. Typical compile options are provided in the header section of the source file, mue.c, for OS/2, Windows NT and AIX.

In order to help make this customization easier, a compile option is available to add tracing output to mue. If mue.c is compiled with `-DTRACING` then the amount of output to stderr is dramatically increased. With a normal compile, most of the output lines call a "noop" procedure instead. This too is covered in the header section of mue.c.

See “Testing configuration files” on page 14, which contains a sample output with tracing that can help you debug while making changes to mue.c.

DATA DRIVEN EXAMPLE

The config/userExit file below contains entries that direct TeamConnection to run the mue program for the following TeamConnection actions: PartAdd (exitId=0) and PartAdd (exitId=2). The other two entries have errors that will not be noticed by teamcd, but will be caught by "mue -C a". This config/userExit file was originally constructed using the tcadmin GUI tool.

```
PartAdd 0 mue "PartAdd 0"
PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target)
PartDelete 0 mue "" ENV=(partpathName)  # no user defined parameter
DefectModify 1 mue "PartModify 1" ENV=(customerName)  # UDP not match action
# udp not match action and exitid
```

Figure 3. Sample config/userExit file that invokes mue

When mue is invoked as a user exit, the following occurs:

1. The config/multExit file is scanned for user exits that match this action and exitId.
2. The conditions, if there are any, are evaluated to determine whether a specific user exit in the config/multExit file should be run.
3. Search the envFile for the value of any parameters required to evaluate conditions.
4. Run user exits whose conditions evaluate true.
5. Update accumulated return code.

Here is a config/multExit file:

```
# mue file:
# ActionName ExitId Exit "Condition(s)"
FeatureOpen 1  viewexit ""
PartAdd     0  viewexit "target==9604"
PartAdd     2  viewexit "RC > 0"
PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 "
PartAdd     2  viewexit "RC=0"
PartModify  1  failtest "RC=0"
PartModify  1  failtest "RC=0"
DefectModify 1 viewexit "RC < 2"
#PartDelete ?
```

Figure 4. Sample config/multExit file that invokes viewexit user exit

This config/multExit file is executed for "PartAdd 0" and "PartAdd 2". As a result, "viewexit" is run for "PartAdd 2", with the sample output below:

Notes:

1. The entire contents of the envFile, envgood.bin, are dumped as part of viewexit. Also, normally the environment file name is generated by tmpnam().
2. The failtest program is a short C program (not included) that always exits with a return code of 1.

```
UEProgram:               viewexit
UEParameters string:     PartAdd 2
EnvFile name:            envgood.bin
Action Parameter [1&brk.:    one
Action Parameter [2&brk.:    two
Action Parameter [3&brk.:    three
Total action parameters: 3
Printing contents of Environment File
Parameter                                Value
=======================================  =====================================
workareaname                             wa5
release                                  MOUSTRAP
workarea                                 1
target                                   9604
targetTeamcUserID
workareaType                             defect
effectiveUserID                          cmvctest
```

Figure 5. viewexit output, with envFile, for simulated PartAdd (exitId=2)

Testing configuration files

For the configuration files in “Data Driven Example” on page 13, the following is the output of checking both files using "mue -C a", with the code compiled normally and with tracing on.

Here is a sample of the normal output from executing "mue -C a":

```
Checking integrity of userexit against daemon and multexit against userexit:
 Verifying ENV from UserExit: # UserExits for PartAdd should work, but others should fail.
 Verifying ENV from UserExit: PartAdd 0 mue "PartAdd 0".
 No ENV parameters to verify.
 Verifying ENV from UserExit: PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target).
 Verifying ENV from UserExit: PartDelete 0 mue "" ENV=(partpathName)  # no user defined parameter.
 Verifying ENV from UserExit: DefectModify 1 mue "PartModify 1" ENV=(customerName, priority)
# udp not match action and exitid.
mue: 1 error(s) found in e:\anon\userexit\mue\config\userExit, comparing to "teamc report -userExitInfo".
Checking integrity of mue configuration:
Verifying from UserExit: # UserExits for PartAdd should work, but others should fail.
Verifying from UserExit: PartAdd 0 mue "PartAdd 0".
Verifying from UserExit: PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target).
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
  Comparing with multexit line: #PartDelete ?.
Completed verification of UE Parm: for...
 Action=PartAdd, ExitID=2, ExitName=mue
Verifying from UserExit: PartDelete 0 mue "" ENV=(partpathName)  # no user defined parameter.
Error: ActionName and ExitID in User Defined Parameter must match
       ActionName and ExitID parameters.
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
  Comparing with multexit line: #PartDelete ?.
Error: Could not find valid ActionName/ExitID in e:\anon\userexit\mue\config\multExit
         for Action PartDelete, ExitID 0, ExitName mue
Verifying from UserExit: DefectModify 1 mue "PartModify 1" ENV=(customerName, priority)
# udp not match action and exitid.
Error: ActionName and ExitID in User Defined Parameter must match
       ActionName and ExitID parameters.
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
  Comparing with multexit line: #PartDelete ?.
Completed verification of UE Parm: for...
 Action=DefectModify, ExitID=1, ExitName=mue
mue: 1 error(s) found in e:\anon\userexit\mue\config\userExit.
```

Figure 6. Sample output from "mue -C a"

Here is a sample of the output from of "mue -C a" when compiled with tracing:

```
Checking integrity of userexit against daemon and multexit against userexit:
 File e:\anon\userexit\mue\config\userExit open.
 Verifying ENV from UserExit: # UserExits for PartAdd should work, but others should fail.
 Comment in : # UserExits for PartAdd should work, but others should fail
 Verifying ENV from UserExit: PartAdd 0 mue "PartAdd 0".
 No environment string
 No ENV parameters to verify.
 Verifying ENV from UserExit: PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target).
 Adding 0 to EnvList: workareaname.
 Adding 1 to EnvList: target.
 Adding 0 to Ref table: partpathName.
 Adding 1 to Ref table: temporaryfileonserver.
 Adding 2 to Ref table: release.
 Adding 3 to Ref table: component.
 Adding 4 to Ref table: filetype.
 Adding 5 to Ref table: WorkAreaName.
 Adding 6 to Ref table: remarks.
 Adding 7 to Ref table: fMode.
 Adding 8 to Ref table: parentname.
 Adding 9 to Ref table: parsername.
 Adding 10 to Ref table: buildername.
 Adding 11 to Ref table: relationtoparent.
 Adding 12 to Ref table: buildparameters.
 Adding 13 to Ref table: parttype.
 Adding 14 to Ref table: parenttype.
 Adding 15 to Ref table: temporaryFlag.
 Adding 16 to Ref table: effectiveUserID.
 Adding 17 to Ref table: VerboseFlag.
 No configurable fields to add for PartAdd.
Verifying ENV from UserExit: PartDelete 0 mue "" ENV=(partpathName)  # no user defined parameter.
 Adding 0 to EnvList: partpathName.
 Rewinding file e:\anon\userexit\mue\tmpmue.ref
 Adding 0 to Ref table: partpathName.
 Adding 1 to Ref table: release.
 Adding 2 to Ref table: forceFlag.
 Adding 3 to Ref table: WorkAreaName.
 Adding 4 to Ref table: commonFlag.
 Adding 5 to Ref table: parttype.
 Adding 6 to Ref table: effectiveUserID.
 Adding 7 to Ref table: TeamcUserID.
 Adding 8 to Ref table: VerboseFlag.
 No configurable fields to add for PartDelete
Verifying ENV from UserExit: DefectModify 1 mue "PartModify 1" ENV=(customerName, priority)
# udp not match action and exitid.
 Adding 0 to EnvList: customerName.
 Adding 1 to EnvList:  priority.
 Rewinding file e:\anon\userexit\mue\tmpmue.ref
 Adding 0 to Ref table: defectname.
 Adding 1 to Ref table: newdefectname.
 Adding 2 to Ref table: severity.
 Adding 3 to Ref table: environmentname.
 Adding 4 to Ref table: prefix.
 Adding 5 to Ref table: reference.
 Adding 6 to Ref table: drivername.
 Adding 7 to Ref table: abstract.
 Adding 8 to Ref table: originator.
 Adding 9 to Ref table: answer.
 Adding 10 to Ref table: remarks.
 Adding 11 to Ref table: release.
 Adding 12 to Ref table: configFields.
```

```
 Adding 13 to Ref table: dateoflastupdate.
 Adding 14 to Ref table: effectiveUserID.
 Adding 15 to Ref table: VerboseFlag.
 Adding config 16 to Ref table: symptom.
 Adding config 17 to Ref table: phaseFound.
 Adding config 18 to Ref table: phaseInject.
 Adding config 19 to Ref table: priority.
 Adding config 20 to Ref table: target.
 Adding config 21 to Ref table: pubsImpact.
 Adding config 22 to Ref table: guiImpact.
 Adding config 23 to Ref table: duration.
 Adding config 24 to Ref table: solution.
 Adding config 25 to Ref table: customerName.
 Adding config 26 to Ref table: pmrNumber.
 Adding config 27 to Ref table: testImpact.
 Adding config 28 to Ref table: installImpact.
mue: 1 error(s) found in e:\anon\userexit\mue\config\userExit, comparing to "teamc report -userExitInfo".
Checking integrity of mue configuration:
 Rewinding file e:\anon\userexit\mue\config\userExit
 File e:\anon\userexit\mue\config\userExit open.
Verifying from UserExit: # UserExits for PartAdd should work, but others should fail.
 Comment in : # UserExits for PartAdd should work, but others should fail
Verifying from UserExit: PartAdd 0 mue "PartAdd 0".
 No environment string
Verifying from UserExit: PartAdd 2 mue "PartAdd 2" ENV=(workareaname,target).
 Adding 0 to EnvList: workareaname.
 Adding 1 to EnvList: target.
 File e:\anon\userexit\mue\config\multExit open.
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
 Verifying conditions: RC > 0
 First condition parameter is RC, defer setting.
 Operator: >.
 Second condition parameter: 0.
 Numeric compare: >.
 Return code cumulative compare: 0.
 Numeric value of 2nd condition: 0.
 Result Pass: 0 > 0.
 Exit Conditions Verified.
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
 Verifying conditions: RC=0;workareaname!=NULL;target()9604,9607,v207
 First condition parameter is RC, defer setting.
 Operator: =.
 Second condition parameter: 0.
 Numeric compare: =.
 Return code cumulative compare: 0.
 Numeric value of 2nd condition: 0.
 Result Pass: 0 = 0.
 First condition must be an envName: workareaname.
 Verifying first condition is valid envParm.
 Element match 0 in EnvList: workareaname.
 Operator: !=.
 Second condition parameter: NULL.
 String compare: !=.
 Checking for NULL.
 Result Fail, string is .
 First condition must be an envName: target.
 Verifying first condition is valid envParm.
 Element match 1 in EnvList: target.
```

```
 Operator: ().
 Second condition parameter: 9604,9607,v207.
 Set compare: 9604,9607,v207.
 Current in set: 9604.
 Current in set: 9607.
 Current in set: v207.
Result Fail,  not in set.
 Exit Conditions Verified.
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
 Verifying conditions: RC=0
 First condition parameter is RC, defer setting.
 Operator: =.
 Second condition parameter: 0.
 Numeric compare: =.
 Return code cumulative compare: 0.
 Numeric value of 2nd condition: 0.
 Result Pass: 0 = 0.
 Exit Conditions Verified.
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
  Comparing with multexit line: #PartDelete ?.
Completed verification of UE Parm: for...
 Action=PartAdd, ExitID=2, ExitName=mue
Verifying from UserExit: PartDelete 0 mue "" ENV=(partpathName)  # no user defined parameter.
Error: ActionName and ExitID in User Defined Parameter must match
       ActionName and ExitID parameters.
 Adding 0 to EnvList: partpathName.
 Rewinding file e:\anon\userexit\mue\config\multExit
 File e:\anon\userexit\mue\config\multExit open.
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
  Comparing with multexit line: #PartDelete ?.
Error: Could not find valid ActionName/ExitID in e:\anon\userexit\mue\config\multExit
         for Action PartDelete, ExitID 0, ExitName mue
Verifying from UserExit: DefectModify 1 mue "PartModify 1" ENV=(customerName, priority)
# udp not match action and exitid.
Error: ActionName and ExitID in User Defined Parameter must match
       ActionName and ExitID parameters.
 Adding 0 to EnvList: customerName.
 Adding 1 to EnvList:  priority.
 Rewinding file e:\anon\userexit\mue\config\multExit
 File e:\anon\userexit\mue\config\multExit open.
  Comparing with multexit line: # mue file:.
  Comparing with multexit line: # ActionName ExitId Exit "Condition(s)".
  Comparing with multexit line: FeatureOpen 1 viewexit "".
  Comparing with multexit line: PartAdd     0  viewexit "target==9604".
  Comparing with multexit line: PartAdd     2  viewexit "RC > 0".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0;workareaname!=NULL;target()9604,9607,v207 ".
  Comparing with multexit line: PartAdd     2  viewexit "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: PartModify  1  failtest "RC=0".
  Comparing with multexit line: DefectModify 1 viewexit "RC < 2".
 Verifying conditions: RC < 2
```

```
 First condition parameter is RC, defer setting.
 Operator: <.
 Second condition parameter: 2.
 Numeric compare: <.
 Return code cumulative compare: 0.
 Numeric value of 2nd condition: 2.
 Result Pass: 0 < 2.
 Exit Conditions Verified.
  Comparing with multexit line: #PartDelete ?.
Completed verification of UE Parm: for...
 Action=DefectModify, ExitID=1, ExitName=mue
mue: 1 error(s) found in e:\anon\userexit\mue\config\userExit.
```

APPENDIX A. VIEWEXIT SOURCE CODE

The viewexit.c program is a simple C program that provides the following features:

- Displays the contents of the parameter list that is passed from TeamConnection (in context)
- Calls a function to read and display the names of parameters (and their values) in an environment file, if one was used
- Identifies when limits of any parameter, or the entire parameter list, on the command line have been exceeded (using ellipses)

```
/*
**********************************************************************

SAMPLE NAME:  viewExit.c

USAGE:        User Exit, see <family dir>\config\UserExit for details

COMPILATION:  cc -o viewExit viewExit.c

ENVIRONMENT VARIABLES:
                       none

DESCRIPTION:  This sample user exit displays the parameter list passed to
              a user exit from a TeamConnection command.  This is the
              C program sample, there are equivalent samples for REXX and ksh.

**********************************************************************
*             IBM TeamConnection for OS/2
*                     Version 2 Release 0
*
*                       5622-717
*    (C) Copyright, IBM Corp., 1996. All Rights Reserved.
*             Licensed Materials - Property of IBM
*
*         US Government Users Restricted Rights
*     - Use, duplication or disclosure restricted by
*         GSA ADP Schedule Contract with IBM Corp.
*
*              IBM is a registered trademark of
*        International Business Machines Corporation
**********************************************************************
*
*           NOTICE TO USERS OF THE SOURCE CODE EXAMPLES
*
* INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THE SOURCE CODE
* EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS, "AS IS" WITHOUT
* WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT
* LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE
* OF THE SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS,
* IS WITH YOU.  SHOULD ANY PART OF THE SOURCE CODE EXAMPLES PROVE
* DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED DEALER) ASSUME THE ENTIRE
* COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
*
```

```
*********************************************************************
*/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

extern int errno;

/* This is based on a limit used for actions in TeamC */
#define maxParmName 40

/*-------------------------------------------------------------------------\
|  envGetFromEnvFile:
|  - Read each entry to environment file
|  - Read binary:
|    size of parameter, parameter string, size of value, value string          |
|  - Minimal error checking to simplify example
|  Note:  This routine is condensed from the sample program teamcenv.c
\-------------------------------------------------------------------------*/
int envGetFromEnvFile(char *envFileName)
{
   FILE *envFile;

   int nNameLength;
   int nValueLength;
   char parameterName[maxParmName+1]; /* allow for maximum in TeamC (15 + NULL) */
   char parameterValue[16001]; /* allow for max in TeamC 16000 for remarks + NULL */

   /* Open temporary file */
   envFile = fopen(envFileName, "rb");
   if (envFile == NULL)
   {
       fprintf(stderr,"teamcenv: Error, could not open file \"%s\"\n",
          envFileName);
       return 1;
   }

   /* Dump all attributes */
      printf("\
Parameter                              Value\n\
====================================  ====================================\n");
   fread(&nNameLength, sizeof(int), 1, envFile);
   fread(parameterName, sizeof(char), nNameLength, envFile);
   *(parameterName+nNameLength)='\0';
   fread(&nValueLength, sizeof(int), 1, envFile);
   fread(parameterValue, sizeof(char), nValueLength, envFile);
   *(parameterValue+nValueLength)='\0';

   while (!feof(envFile))
   {
      strncat(parameterName, "                                        ",
         (maxParmName - strlen(parameterName)));
      *(parameterName+maxParmName) = '\0';
      printf("%s %s\n", parameterName, parameterValue);
      fread(&nNameLength, sizeof(int), 1, envFile);
      *(parameterName+nNameLength)='\0';
      fread(parameterName, sizeof(char), nNameLength, envFile);
      fread(&nValueLength, sizeof(int), 1, envFile);
      fread(parameterValue, sizeof(char), nValueLength, envFile);
      *(parameterValue+nValueLength)='\0';
   }
```

```
   fclose(envFile);
   return 0;
}


/*-------------------------------------------------------------------------\
|  main:
|  - Print standard set of arguments at beginning of parameter list:
|    UEprogram, UEparameter and EnvFile
|  - Print rest of arguments; the positional parameters
|  - If EnvFile is not NULL, print the contents of the EnvFile
\-------------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
   int i,n;

   /* Compute the number of action parameters passed at definition     */
   int totalParms = argc - 3;

   /* Display the name of the command.                                 */
   /* It is not necessary to parse name.                               */
   printf("UEProgram:               %s\n", argv[0]);

   /* Display parameter list.                                          */
   if (strlen(argv[1]) != 0)
      printf("UEParameters string:     %s\n", argv[1]);
   else
      printf("UEParameters string is NULL\n");

   /* Display parameter env file.                                      */
   if (strlen(argv[2]) != 0)
      printf("EnvFile name:            %s\n", argv[2]);
   else
      printf("No environment file; string is NULL\n");

   /* Display parameter list.                                          */
   for (i = 3; i < argc; i++)
   {
      n = strlen(argv[i]);
      if (n == 0)
         printf("Action Parameter [%d] is NULL\n", i-2);
      else
      {
         printf("Action Parameter [%d]:    %s\n", i-2, argv[i]);

         /* Ellipses in data, checking for end (i.e. truncation)       */
         /* - Each parameter limited to 400 bytes in OS/2, Windows     */
         /*   and Windows NT, and 16000 bytes in Unix                  */
         /* - Total command string limited to 1024 bytes in OS/2,      */
         /*   Windows and Windows NT, and 32000 bytes in Unix          */
         /* - compute address of last 3 characters then compare        */
         if (strcmp(((argv[i])+n-3), "...") == 0)
         {
            printf("                         parameter %d string was \
truncated!\n", (i-1));
            /* If last parameter truncated, then entire list truncated */
            if (i == (argc - 1))
               printf("Parameter list was truncated!\n");
         }
      }
   }
   printf("Total action parameters: %d\n", totalParms);
```

```
   /* Print contents of parameter env file.                          */
   if (strlen(argv[2]) != 0)
   {
      printf("Printing contents of Environment File\n");
      envGetFromEnvFile(argv[2]);
   }

   return (0);
}

/* End of File */
```

APPENDIX B. MUE SOURCE CODE

The mue (Multiple User Exit) tool is written in C so that it can easily be ported to run on any TeamConnection server platform. Further, the rules for data driven user exits should be easy to enhance or modify.

The source code for mue.c is provided here because the comments and code should be useful to a family administrator trying to make the most of TeamConnection user exits. In many cases, this code was originally copied from the TeamConnection source code in order to better show how TeamConnection user exits are processed. Comments in the code indicate the relationship to the TeamConnection source code.

```
/*
*********************************************************************

SAMPLE NAME:  mue.c
              Multiple User Exits

USAGE:        mue ActionName ExitID ... (called as a user exit by teamcd)
              mue -C [a|m|u]            (mue server configuration check)

- Call in userExit file
  ActionName ExitID mue "uep" ENV=(...) # Calling MultiUserExit Program
  example:
     PartAdd 0 mue "PartAdd 0" ENV=(component,release) # comment
- Call in multExit file
  ActionName ExitID ExitName "uep" ENV=(...) # Calling MultiUserExit Program
  example:
     PartAdd 0 viewExit "RC>0,component!=NULL" # comment


COMPILATION:
             icc mue.c (OS/2)
             icc -DTRACING /Ti+ mue.c (OS/2 debug w/Tracing output to stderr)
             icc -D__NT__ mue.c (NT)
             icc -D__NT__ -DTRACING /Ti+ mue.c (NT w/debug ...)
             cc -D__UNIX__ -o mue mue.c  (Unix)
             cc -D__UNIX__ -DTRACING -g -o mue mue.c (Unix debug w/Tracing output to stderr)

ENVIRONMENT VARIABLES: TC_FAMILY  (all used by teamc command during "mue -C a"
                       TC_BECOME   and mue -C u)
                       TC_DBPATH
                       TC_TMP

DESCRIPTION:  This sample user exit allows you to invoke the mue utility to
              manage multiple user exits for a single action.  Conditions
              can be specified for each exit, resulting in a "data-driven"
              user exit mechanism. This proram contains a wide range of
              utilities to allow you to customize or enhance mue for your own
              uses.

CALLING HIERARCHY:
 main
    usage (-?)                    - options for calling mue
      teamcRun                    - Process exit parameters from TeamC
```

```
         mueInitFile               - Set up to read config/multExit file
         exitParseAction           - Get ActionName and ExitId from string
         stripBlanks               - Remove leading/trailing blanks and ctrls
         exitParseCall             - Parse for Action/ExitID/Exit and verify
            stripBlanks
         envInitFile               - Set up to read from envFile (argv[2])
         exitVerifyConditions      - Evaluate and verify conditions
            stripBlanks
            envRead                - Get a value from env file
            envLookupParm          - Is a parameter in environment list
            exitVerifyOrTest       - Is this a Check or actual exit call
         mueRun                    - Execute a userexit just like TeamC
      ueVerifyFile (-C a, -C m)    - Verify mue calls in userExit file
         ueInitFile                - Set up to read ocnfig/userexit file
         stripBlanks
         exitParseCall
            stripBlanks
         exitParseAction
         envParseList              - Parse ENV=()
         mueVerifyFile             - Verify mue file for a ue call to mue
            mueInitFile
            stripBlanks
            exitParseCall
            exitVerifyConditions
               stripBlanks
               envRead
               envLookupParm
               exitVerifyOrTest
      refVerifyFile (-C a, -C u)   - Verify contents of config/userExit
         ueInitFile
         stripBlanks
         exitParseCall
            stripBlanks
         envParseList
         refParseList
            refInitFile            - Get output of teamc report -userExitInfo
            stripBlanks
         refVerifyEntry            - Compare parms in conditions w/ teamc report

NOTES:        1. This utility is written in C for portability and
                 compatibility with CMVC code that will be integrated
                 with this utility by some IBM internal users.
              2. This code contains lots of hints about how TeamConnection
                 currently does things (under the covers).  It will be
                 noted where appropriate.
*********************************************************************
*             IBM TeamConnection for OS/2
*                    Version 2 Release 0
*
*                      5622-717
*    (C) Copyright, IBM Corp., 1997. All Rights Reserved.
*             Licensed Materials - Property of IBM
*
*         US Government Users Restricted Rights
*     - Use, duplication or disclosure restricted by
*         GSA ADP Schedule Contract with IBM Corp.
*
*              IBM is a registered trademark of
*        International Business Machines Corporation
*********************************************************************
*
*           NOTICE TO USERS OF THE SOURCE CODE EXAMPLES
```

```
*
* INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THE SOURCE CODE
* EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS, "AS IS" WITHOUT
* WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT
* LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE
* OF THE SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS,
* IS WITH YOU.  SHOULD ANY PART OF THE SOURCE CODE EXAMPLES PROVE
* DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED DEALER) ASSUME THE ENTIRE
* COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
*
**********************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#if defined(__OS2__) || defined(__NT__)
 #include <io.h>
#endif

/* For tracing define "TRACING" */
#ifdef TRACING
 #define TRACE fprintf
#else
 #define TRACE noop
#endif

extern int errno;

/* This is based on a limit used for acions in TeamC */
#define nMaxParmName 40
#define nMaxUEParm   1024
#ifdef __UNIX__
 #define strUEFileName "/config/userExit"
 #define strMUEFileName "/config/multExit"
#else
 #define strUEFileName "\\config\\userExit"
 #define strMUEFileName "\\config\\multExit"
#endif
#define strRefFileName "mue.ref"
#define strUEDefinedKeyword "Configured User Exits:"

#define boolean int
#define True 1
#define False 0

/* These are values used in TeamConnection */
#define maxParmName 40
#define maxArgs 64
#define STDOUT 1
#define STDERR 2
#define maxPathLen 1024
#ifdef __UNIX__
 #define OSmaxCommandLen 32000
#else
```

```
 #define OSmaxCommandLen 1024
#endif

/* Global variables: */
char msg[40000];                        /* Message buffer for UE output */
char *strTCHome = NULL;                 /* Store TC_HOME */
char *strTCTmp = NULL;                  /* Store TC_TMP */
char  strUEPath[maxPathLen]  = "";      /* Path to userexit file */
char  strMUEPath[maxPathLen] = "";      /* Path to multexit flie */
char  strEnvPath[maxPathLen] = "";      /* Path to temporary envFile */
char  strRefPath[maxPathLen] = "";      /* Path to refFile w/teamc report data*/
char  arEnvList[maxArgs][maxParmName];  /* Array of parms from userexit */
char  arRefList[maxArgs][maxParmName];  /* Array of parms from ref file */
char  cmdToBeExec[OSmaxCommandLen] = ""; /* Command executed by system() */
boolean fCheck = False;                 /* Checking files or running exit? */
FILE *fpUEFile  = (FILE *)NULL;
FILE *fpMUEFile = (FILE *)NULL;
FILE *fpEnvFile = (FILE *)NULL;
FILE *fpRefFile = (FILE *)NULL;
int    ArgC;                            /* Global for argc in main */
char **ArgV;                            /* Global poiner to argv in main */

/* Function prototypes */
void    usage(boolean fLong);
#ifdef __UNIX__
void    noop(struct __file *f, char *s, ...);
#else
void    _System noop(struct __file *f, char *s, ...);
#endif
void    stripBlanks(char *pString);
int     ueInitFile(void);
int     mueInitFile(void);
int     envRead(char *strEnvFileName, char *strEnvName, char *strEnvValue);
int     exitVerifyOrTest(int nPassed);
int     exitVerifyConditions(char *strCondition, int *nValueRC);
int     refInitFile(void);
int     envLookupParm(char *strEnvParm);
int     envParseList(char *ueInputString);
int     envVerifyList(char *ueInputString, char *actionName, int exitNum);
int     refParseList(char *strAction, int nExitId);
int     refVerifyEntry(void);
int     exitParseAction(char *strUEParm, char *strAction, int *nID);
int     exitParseCall(char *strUEParm, char *strAction, int *nID,
           char *strExit, char *strOther);
int     refVerifyFile(char *strMUECall);
int     mueVerifyFile(char *strActionName, int nExitID, char *strExitName);
int     ueVerifyFile(char *strMUECall);
int     mueRun(char *strNewCmd);
int     teamcRun(void);


/*-----------------------------------------------------------------------------\
|  usage:                                                                      |
|  - display normal use message                                                |
\-----------------------------------------------------------------------------*/
void usage(boolean fLong)
{
   fprintf(stderr, "Multiple User Exit usage:\n\
 1. Called from command line:\n\
    mue -?         // Display complete help message\n\
    mue -C [a|m|u] // Checks userexit and multexit files;\n\
                   // where 'm' verifies config/multExit file, 'u' verifies\n\
```

```
                  // config/userExit, and 'a' verifies all (m+u).\n");

      fprintf(stderr, "\
 2. Registered as TeamConnection user exit (%s):\n\
    ActionName ExitID mue \"ActionName ExitID\" ENV=(...) # Comment\n",
    strUEFileName);

   if (fLong)
   {
      fprintf(stderr, "\
  Example: PartAdd 0 mue \"PartAdd 0\" ENV=(component,release)\n");
      fprintf(stderr, "\
  Registration of conditional exits in MultiUser file (%s):\n\
    ActionName ExitID ExitName Conditions # comment\n\
  Example: PartAdd 0 viewExit \"RC>0,component!=NULL\"\n", strMUEFileName);
      fprintf(stderr, "\
  Acceptable conditional expressions:\n\
    envName > Number;... // Using envName (where envName is in ENV=()),\n\
    RC      <   :        // or RC (where RC is highest return code from\n\
            =   :        // exits).  Multiple conditionals are separated\n\
            <>           // by commas. Numeric compares convert strings to\n\
           == String     // numbers.  String compares do NOT allow spaces\n\
           != NULL       // and does NOT use quotes.\n\
           in String,String...\n");
      fprintf(stderr, "\
NOTE: \"mue -C\" searches config/userExit for \"mue\" in lower case.\n");
   }

   exit (-1);
}

/*-----------------------------------------------------------------------------\
|  noop:                                                                       |
|  - a means to control tracing                                                |
\-----------------------------------------------------------------------------*/
#ifdef __UNIX__
void noop(struct __file *f, char *s, ...)
#else
void _System noop(struct __file *f, char *s, ...)
#endif
{
   return;
}

/*-----------------------------------------------------------------------------\
|  stripBlanks:                                                                |
|  - remove leading and trailing blanks                                        |
\-----------------------------------------------------------------------------*/
void stripBlanks(char *pString)
{
   int   n = 0;
   char *p;

   /* Remove Trailing blanks (and other assorted control characters) */
   n = strlen(pString);
   if (n > 0)
   {
      p = pString + n - 1;
      while ((p >= pString) &&
             ((*p == ' ') || (*p == '\n') || (*p == '\t') || (*p == '\r') ||
              (*p == '\v') || (*p == '\f'))
            )
```

```
      {
         *p = NULL;
         p--;
      }
   }

   /* Compute leading blanks (and tabs) */
   p = pString;
   while ((*p == ' ') || (*p == '\t'))
   {
      p++;
   }
   n = p - pString;

   /* Remove leading blanks */
   if (n > 0)
   {
      p = pString;
      while (*(p+n) != NULL)
      {
         *(p) = *(p + n);
         *(p + n) = NULL; /* clean up after yourself */
         p++;
      }
      *p = NULL;
   }

   return;
}

/*-------------------------------------------------------------------------\
|  ueInitFile:                                                             |
|  - Open UserExit file for reading text                                   |
|  - Return success or failure and store file ponter globally              |
\-------------------------------------------------------------------------*/
int ueInitFile(void)
{
   struct stat statBuff;

   strcpy(strUEPath, strTCHome);
   strcat(strUEPath,strUEFileName);

   if (fpUEFile != (FILE *)NULL)
   {
      /* close to Rewind */
      TRACE(stderr," Rewinding file %s\n", strUEPath);
      fclose(fpUEFile);
      fpUEFile = (FILE *)NULL;
   }

   /* Open file */
   /* Does file exist ? */
   stat(strUEPath, &statBuff);
   if (statBuff.st_size == 0)
   {
      fprintf(stderr, "Error: user exit file \"%s\" does not exist,\n\
 or contains no values.\n", strUEPath);
      return 1;
   }

   /* Open */
   fpUEFile = fopen(strUEPath, "r");
```

```
   if (fpUEFile == (FILE *)NULL)
   {
       fprintf(stderr,"Error, could not open file \"%s\".\n\
 Please verify that you have permission to read file.\n", strUEPath);
       return 1;
   }
   TRACE(stderr," File %s open.\n", strUEPath);

   return 0;
}

/*-------------------------------------------------------------------------\
|  mueInitFile:                                                            |
|  - Open MultExit file for reading text                                   |
|  - Return success or failure and store file ponter globally              |
\-------------------------------------------------------------------------*/
int mueInitFile(void)
{
   struct stat statBuff;

   strcpy(strMUEPath, strTCHome);
   strcat(strMUEPath,strMUEFileName);

   if (fpMUEFile != (FILE *)NULL)
   {
      /* close to Rewind */
      TRACE(stderr," Rewinding file %s\n", strMUEPath);
      fclose(fpMUEFile);
      fpMUEFile = (FILE *)NULL;
   }

   /* Does file exist ? */
   stat(strMUEPath, &statBuff);
   if (statBuff.st_size == 0)
   {
      fprintf(stderr, "Error:  User exit file \"%s\" does not exist,\n\
 or contains no values.\n", strMUEPath);
      return 1;
   }

   /* Open */
   fpMUEFile = fopen(strMUEPath, "r");
   if (fpMUEFile == (FILE *)NULL)
   {
       fprintf(stderr,"Error: Could not open file \"%s\".\n\
Please verify that you have permission to read file.\n", strMUEPath);
       return 1;
   }
   TRACE(stderr," File %s open.\n", strMUEPath);

   return 0;
}


/*-------------------------------------------------------------------------\
|  envInitFile:                                                            |
|  - Open temporary binary file for reading binary data                    |
|  - Return success or failure and store file ponter globally              |
|  - temporary file automatically deleted when by teamcd                   |
\-------------------------------------------------------------------------*/
int envInitFile(char *strEnvFileName)
{
```

```
   struct stat statBuff;

   if (strlen(strEnvFileName) == 0)
   {
      TRACE(stderr," No envfile to open\n");
      fpEnvFile = (FILE *)NULL;
      return 0;
   }

   if (fpEnvFile != (FILE *)NULL)
   {
      /* close to Rewind */
      TRACE(stderr," Rewinding file %s\n", strEnvFileName);
      fclose(fpEnvFile);
      fpEnvFile = (FILE *)NULL;
   }

   /* Open temporary environment file (complete path supplied) */
   fpEnvFile = fopen(strEnvFileName, "rb");
   if (fpEnvFile == NULL)
   {
       fprintf(stderr,"Error: Could not open file \"%s\"\n",
          strEnvFileName);
       return 1;
   }

   stat(strEnvFileName, &statBuff);
   if (statBuff.st_size == 0)
   {
      fprintf(stderr, "Error: Environment file \"%s\" contains no values\n",
         strEnvFileName);
      return 1;
   }

   return 0;
}

/*-----------------------------------------------------------------------------\
|  refInitFile:                                                                |
|  - Perform "teamc report -userExitInfo", storing result in mue.ref           |
|  * This code puts the temporary file in the same directory used by the       |
|    teamcd daemon.                                                            |
\-----------------------------------------------------------------------------*/
int refInitFile(void)
{
   struct stat statBuff;
   char strLine[nMaxUEParm];
   int nRC;

   if (fpRefFile == (FILE *)NULL)
   {
      /* Generate Path for temporary reference file */
      strcpy(strRefPath, strTCTmp);
      strcat(strRefPath, strRefFileName);

      /* Open temporary reference file (in TC temporary directory) */
#ifdef __UNIX__
      sprintf(strLine, "teamc report -userExitInfo > %s 2>/dev/null",
         strRefPath);
#else
      sprintf(strLine, "teamc report -userExitInfo > %s 2>nul",
```

```
         strRefPath);
#endif

      nRC = system(strLine);
      if (nRC)
      {
         fprintf(stderr, "Error: Could not run teamc report -userExitInfo.\n\
 Please verify that:\n\
 - teamc is on the path,\n\
 - TC_FAMILY and TC_BECOME are set,\n\
 - the family is running,\n\
 - you are a superuser.\n");
         return 1;
      }
   }
   else
   {
      /* close to Rewind */
      TRACE(stderr," Rewinding file %s\n", strRefPath);
      fclose(fpRefFile);
      fpRefFile = (FILE *)NULL;
   }

   fpRefFile = fopen(strRefPath, "r");
   if (fpRefFile == (FILE *)NULL)
   {
      fprintf(stderr,"Error: Could not open file \"%s\".\n\
 Please verify that:\n\
 - there is space on the filesystem,\n\
 - you have permission to write a file to the current directory,\n\
 - the file does not already exit in read-only mode.\n",
         strRefPath);
      return 1;
   }

   stat(strRefPath, &statBuff);
   if (statBuff.st_size == 0)
   {
      fprintf(stderr, "Error: Reference file \"%s\" contains no values\n",
         strRefFileName);
      return 1;
   }

   return 0;
}

/*-------------------------------------------------------------------------\
|  envRead:                                                                |
|  - Read each entry to environment file                                   |
|  - Read binary:                                                          |
|    size of parameter, parameter string, size of value, value string      |
|  - Minimal error checking to simplify example                            |
|  Note:  This routine is condensed from the sample program teamcenv.c     |
\-------------------------------------------------------------------------*/
int envRead(char *strEnvFileName, char *strEnvName, char *strEnvValue)
{
   int  nNameLength;
   int  nValueLength;
   int  nRC;
   char strInName[maxParmName+1]; /* allow for maximum in TeamC (15 + NULL) */
   char strInValue[16001]; /* allow for max in TeamC 16000 for remarks + NULL */
```

```
   /* Open file */
   nRC = envInitFile(strEnvFileName);
   if (nRC)
   {
      return nRC;
   }

   fread(&nNameLength, sizeof(int), 1, fpEnvFile);
   fread(strInName, sizeof(char), nNameLength, fpEnvFile);
   *(strInName+nNameLength)='\0';
   fread(&nValueLength, sizeof(int), 1, fpEnvFile);
   fread(strInValue, sizeof(char), nValueLength, fpEnvFile);
   *(strInValue+nValueLength)='\0';

   while ((strcmp(strEnvName, strInName) != 0) && (!feof(fpEnvFile)))
   {
      fread(&nNameLength, sizeof(int), 1, fpEnvFile);
      fread(strInName, sizeof(char), nNameLength, fpEnvFile);
      *(strInName+nNameLength)='\0';
      fread(&nValueLength, sizeof(int), 1, fpEnvFile);
      fread(strInValue, sizeof(char), nValueLength, fpEnvFile);
      *(strInValue+nValueLength)='\0';
   }

   if (!feof(fpEnvFile))
   {
      /* Return the value of the parameter */
      strcpy(strEnvValue, strInValue);
      return 0;
   }
   else
   {
      fprintf(stderr, "Warning: Parameter \"%s\" not found.\n\
 user did not ener a value, returning NULL.\n", strEnvName);
      strcpy(strEnvValue, "");
      return 1;
   }
}

/*-------------------------------------------------------------------------\
|  envLookupParm:                                                          |
|  Look up a specific variable to see if it is in the list                 |
\-------------------------------------------------------------------------*/
int envLookupParm(char *strEnvParm)
{
   int i = 0;

   /* Find string in table */
   for (i=0;
        ((i<maxArgs) && (arEnvList[i][0] != NULL) &&
         (strcmp(&(arEnvList[i][0]), strEnvParm)!=0)
        );
        i++);

   if (strcmp(&(arEnvList[i][0]), strEnvParm)==0)
   {
      TRACE(stderr," Element match %d in EnvList: %s.\n", i, &(arEnvList[i][0]));
      return 0;
   }
   else
   {
```

```
      fprintf(stderr, "\
Error: Requested envName (%s) not provided through userexit file.\n", strEnvParm);
      return 1;
   }
}

/*-------------------------------------------------------------------------\
|  envParseList:                                                           |
|  Parse the contents of the ENV=(...) clause, if present.                 |
|  This function:                                                          |
|  - parses all fields in ENV=()                                           |
|  * The elements in arEnvList are verified as part of the condition       |
|  * This routine is derived from teamcd daemon                            |
\-------------------------------------------------------------------------*/
int envParseList(char *ueInputString)
{
   char *parmList = NULL, *p = NULL;
   int i = 0;
   char keyStartString[]="ENV=(";
   char keyEndString[]=")";
   char keySeparator[]=",";

   /* Clear EnvList table */
   for(i=0;i<maxArgs;i++)
   {
      arEnvList[i][0] = NULL;
   }

   /* Parsing ENV list */
   parmList = strstr(ueInputString, keyStartString);

   if (parmList == NULL)
   {
      TRACE(stderr," No environment string\n");
      return 1;
   }
   parmList = parmList + strlen(keyStartString);

   p = strstr(parmList, keyEndString);
   if (p == NULL)
   {
      fprintf(stderr," Error: No ending for ENV list: %s\n", ueInputString);
      return 1;
   }

   /* Discard user comment */
   *(p) = '\0';

   /*-------------------------------------\
   | Parse parameters in list             |
   | - Check for empty list               |
   | - Create environment variable for    |
   |   each parameter                     |
   \-------------------------------------*/

   for(i=0; (*parmList != '\0') && i < maxArgs; i++)
   {
      p = strstr(parmList, keySeparator);
      /* If key separator not found, at end of string */
      if (p == NULL)
      {
         p = parmList + strlen(parmList);
```

```
      }

      /* Copy current parameter */
      strncpy(arEnvList[i], parmList, (int)(p - parmList));
      arEnvList[i][p-parmList] = NULL;
      TRACE(stderr," Adding %d to EnvList: %s.\n", i, &(arEnvList[i][0]));

      /* move pointer to next parameter in list */
      if (*p != '\0')
      {
         parmList = p + strlen(keySeparator);
      }
      else
      {
         parmList = p;
      }
   }

   return 0;
}

/*-----------------------------------------------------------------------------\
|  exitVerifyOrTest:                                                           |
|    This little tool allows verification and evaluation of conditions in one  |
|    little routine (handling return codes passed)                             |
|  - return nPassed for evaluation, reflecting if the condition passed         |
|  - return 0 RC for verificaton, regardless of whether the condition passed   |
\-----------------------------------------------------------------------------*/
int exitVerifyOrTest(int nPassed)
{

    if (fCheck)
    {
       return 0; /* always passed during verification */
    }
    else
    {
       return nPassed; /* if it passed, then return that */
    }
}

/*-----------------------------------------------------------------------------\
|  exitVerifyConditions:                                                       |
|  - Test user exit parameters for proper condition                            |
|  * When this routine is called, a defined user exit is pointed to in the     |
|    reference file.  If necessary, this routine can read the regular and      |
|    configurable parameters in order to verify the paramters used in the      |
|    conditional exppressions on any line in the multExit file that is         |
|    invoked for this action/exit combination.                                 |
|  - If equation requires checking value of named parameter, call              |
|    envRead                                                                   |
|Evaluate Condition:                                                           |
|- loop for each:                                                              |
|   envName > Number;envName...                                                |
|   RC      <                                                                  |
|           =                                                                  |
|           <>                                                                 |
|                                                                              |
|   envName == String                                                          |
|           != NULL                                                            |
|           () value,value,value                                               |
|- if envName = 'RC' use value from previous routines                          |
```

```
|- if operator '==' or '!=' or '()' and value = 'NULL'                          |
|  check fo strlen=0                                                            |
|- if operator = '()' then count occurances of ','                              |
\-------------------------------------------------------------------------------*/
int exitVerifyConditions(char *strCondition, int *nValueRC)
{
   int      nErrorRC = 0, nVerifyRC = 0, n = 0, nValue = 0, nComp = 0;
   char    *pStr, *pStr2, *pStrSet,
           *pSeparators="<>!=(),",
           *pRCKeyword = "RC",
            cCondSeparator = ';',
            cSetSeparator = ',',
           *pNULL = "NULL";
   char     str1stCond[nMaxUEParm], str2ndCond[nMaxUEParm], strCond[3];
   char     strValue[nMaxUEParm];

   TRACE(stderr," Verifying conditions: %s\n", strCondition);
   pStr = strCondition;
   while ((pStr != NULL) && (nErrorRC == 0))
   {
      if (*pStr == cCondSeparator)
      {
         pStr++;
      }
      stripBlanks(pStr);

      /* get first parameter (envName or RC) */
      /* - separators are ><=!(), */
      pStr2 = strpbrk(pStr, pSeparators);
      if (pStr2 == NULL)
      {
         fprintf(stderr," Error: Could not find a separator in condition.\n");
         nErrorRC = 1;
      }
      else
      {
         strncpy(str1stCond, pStr, (pStr2 - pStr));
         *(str1stCond+(pStr2-pStr)) = NULL;
         stripBlanks(str1stCond);

         if (strcmp(str1stCond, pRCKeyword)==0)
         {
            TRACE(stderr," First condition parameter is RC, defer setting.\n");
         }
         else
         {
            TRACE(stderr," First condition must be an envName: %s.\n",
               str1stCond);
            /* Does value of envFile need to be read? */
            if (fpEnvFile != (FILE *)NULL)
            {
               TRACE(stderr," Need to get Variable from envFile.\n");
               nErrorRC = envRead(strEnvPath, str1stCond, strValue);
            }
            else
            {
               TRACE(stderr," Verifying first condition is valid envParm.\n");
               strcpy(strValue,"");
               nErrorRC = envLookupParm(str1stCond);
            }
         }
```

```
   if (nErrorRC)
   {
      TRACE(stderr," Could not verify first condition parameter: %s.\n",
         str1stCond);
   }
   pStr = pStr+(pStr2-pStr);
}

/* Get operator */
if (nErrorRC == 0)
{
   /* Get all characters in condition */
   n = 0;
   pStr2=pStr;
   while (strpbrk(pStr2, pSeparators) == (pStr2))
   {
      pStr2++;
   }
   strncpy(strCond, pStr, (pStr2-pStr));
   *(strCond+(pStr2-pStr)) = NULL;
   stripBlanks(strCond);
   TRACE(stderr," Operator: %s.\n", strCond);
   pStr = pStr2;
}

/* Get second condition parameter */
if (nErrorRC == 0)
{
   /* Get 2nd parameter */
   pStr2 = strchr(pStr, cCondSeparator);
   if (pStr2 != NULL)
   {
      strncpy(str2ndCond, pStr, (pStr2-pStr));
      *(str2ndCond+(pStr2-pStr)) = NULL;
   }
   else
   {
      strcpy(str2ndCond, pStr);
   }
   stripBlanks(str2ndCond);
   TRACE(stderr," Second condition parameter: %s.\n", str2ndCond);
}

if (nErrorRC)
{
   fprintf(stderr, "Error: Invalid condition format: %s.\n", strCondition);
   return nErrorRC;
}

/* Analyze conditions                              */

/* Numeric checking:    */
/*    envName >  Number */
/*    RC      <    "    */
/*            =    "    */
/*            <>   "    */
if ((strcmp(strCond,">")==0) ||
    (strcmp(strCond,"<")==0) ||
    (strcmp(strCond,"=")==0) ||
    (strcmp(strCond,"<>")==0)
   )
{
```

```
         TRACE(stderr," Numeric compare: %s.\n", strCond);

         /* Check RC? */
         if (strcmp(str1stCond,pRCKeyword)==0)
         {
            nValue = *nValueRC;
            TRACE(stderr," Return code cumulative compare: %d.\n", nValue);
         }
         else
         {
            /* Use value from envFile */
            nValue = atoi(strValue);
            TRACE(stderr," Numeric value of 1st condition: %d.\n", nValue);
         }

         nComp = atoi(str2ndCond);
         TRACE(stderr," Numeric value of 2nd condition: %d.\n", nComp);


         /* Select proper compare */
         if (strlen(strCond)==1)
         {
            /* Compare first character of condition */
            switch (*strCond)
            {
               case '>': if (nValue <= nComp)
                         {
                            nErrorRC = exitVerifyOrTest(1);
                         }
                         break;

               case '<': if (nValue >= nComp)
                         {
                            nErrorRC = exitVerifyOrTest(1);
                         }
                         break;

               case '=': if (nValue != nComp)
                         {
                            nErrorRC = exitVerifyOrTest(1);
                         }
                         break;

              default:  fprintf(stderr,"\
Bad numeric condition %s; should not happen.\n", strCond);
                        nErrorRC = 1;
                        break;
            }
         }
         else /* not equal */
         {
            if (nValue == nComp)
               nErrorRC = exitVerifyOrTest(1);
         }

         /* Show results */
         if (nErrorRC)
            TRACE(stderr," Result Fail: %d %s %d.\n", nValue, strCond, nComp);
         else
            TRACE(stderr," Result Pass: %d %s %d.\n", nValue, strCond, nComp);
      }
      else /* Not Numeric - RC not allowed */
```

```
{
   if (strcmp(str1stCond,pRCKeyword)==0)
   {
      fprintf(stderr,"Error: Cannot use RC with non-numeric compare: %s.\n",
         strCond);
      nErrorRC = 1;
   }
}

if (nErrorRC == 0)
{
   /* Text checking        */
   /*    envName == String */
   /*            !=   "    */
   if ((strcmp(strCond,"==")==0) ||
       (strcmp(strCond,"!=")==0)
      )
   {
      TRACE(stderr," String compare: %s.\n", strCond);

      /* Check for NULL */
      if (strcmp(str2ndCond, pNULL) == 0)
      {
         TRACE(stderr," Checking for NULL.\n");
         if (
             ((*strValue == NULL) && (strcmp(strCond,"==")==0)) ||
             ((*strValue != NULL) && (strcmp(strCond,"!=")==0))
            )
         {
            nErrorRC = exitVerifyOrTest(0);
            TRACE(stderr," Result Pass, string is NULL.\n");
         }
         else
         {
            TRACE(stderr," Result Fail, string is %s.\n", strValue);
            nErrorRC = exitVerifyOrTest(1);
         }
      }
      else
      {
         TRACE(stderr," Checking for against string: %s.\n", str2ndCond);
         if (strcmp(strValue, str2ndCond) == 0)
         {
            TRACE(stderr," Result Pass, string match.\n");
            nErrorRC = exitVerifyOrTest(0);
         }
         else
         {
            TRACE(stderr," Result Fail, string %s does not match %s.\n",
               strValue, str2ndCond);
            nErrorRC = exitVerifyOrTest(1);
         }
      }
   }

   /* In set               */
   /*    envName () string,string... */
   if ((strcmp(strCond,"()")==0)
      )
   {
      TRACE(stderr," Set compare: %s.\n", str2ndCond);
```

```
            n = strlen(str2ndCond);
            pStrSet = str2ndCond;
            pStr2 = str2ndCond;
            while (pStrSet != NULL)
            {
               pStr2 = strchr(pStrSet, ',');
               if (pStr2 != NULL)
               {
                  *pStr2 = NULL;
               }
               TRACE(stderr," Current in set: %s.\n", pStrSet);
               if (strcmp(strValue,pStrSet)==0)
               {
                  TRACE(stderr," Result Pass, string in set.\n");
                  nVerifyRC = exitVerifyOrTest(0);
                  break;
               }
               if ((pStr2 != NULL) &&
                   (pStr2 < (str2ndCond + n))
                  )
               {
                  pStrSet = pStr2+1;
               }
               else
               {
                  pStrSet = NULL;
               }
            }
            if (pStrSet == NULL)
            {
               TRACE(stderr, "Result Fail, %s not in set.\n", strValue);
               nErrorRC = exitVerifyOrTest(1);
            }
         }
      }

      /* Get next condition */
      pStr = strchr(pStr, cCondSeparator);
   }

   return nErrorRC;
}

/*-------------------------------------------------------------------------\
|  refParseList:                                                           |
|  - Search the refence file for actionName and exitId                     |
|  - Gather values into an array                                           |
|  - Compare entries in arEnvList with entries from reference file         |
\-------------------------------------------------------------------------*/
int refParseList(char *strAction, int nExitId)
{
   char strLine[nMaxUEParm];
   char *pStr = NULL, *pStr2 = NULL;
   int  nRC = 0, n;
   char strTmp[nMaxUEParm];

   nRC = refInitFile();
   if (nRC)
      return nRC;

   /* Find start of configured exit section (ActionName) */
   /* - if Action is found, assume rest of file is correclty formatted */
```

```
   pStr = fgets(strLine, sizeof(strLine), fpRefFile);
   while (strstr(strLine, strAction) == NULL)
   {
      pStr = fgets(strLine, sizeof(strLine), fpRefFile);
   }

   if (feof(fpRefFile))
   {
      fprintf(stderr, "Error: Reference file not complete,\n\
likely error \"teamc report -userExitInfo\".\n");
      return 1;
   }

   /* Find User Exit strings */
   n = nExitId;
   if (n == 0)
      n = 3;
   sprintf(strTmp, "%d:", n);
   while (strstr(strLine, strTmp) == NULL)
   {
      fgets(strLine, sizeof(strLine), fpRefFile);
   }

   /* Populate arRefList */
   /* - search until Parameter List line does NOT end with a comma, */
   /* - then search configurable field for same thing */
   /* - Skip past "Parameter List:" on first line */
   /* - can tell if there are configurable fields by presence, or */
   /*   lack of presence of "onfigurable" */
   /* - parse each line by getting string */
   /*   - if last char is a comma, replace with NULL */
   /*   - if last character is NULL, copy then get next line */

   pStr = fgets(strLine, sizeof(strLine), fpRefFile);
   stripBlanks(pStr);
   pStr = strchr(pStr,':');
   pStr++;
   stripBlanks(pStr);
   n = 0;
   /* Get each parameter (comma separated) from regular parameter list */
   /* - End when encountering a line with a colon or period */
   while (strpbrk(pStr,":.") == NULL)
   {
      while (*pStr != NULL)
      {
         /* copy */
         pStr2 = strchr(pStr, ',');
         if (pStr2 != NULL)
            *pStr2 = NULL;
         else
            pStr2 = pStr + strlen(pStr);

         strcpy(&(arRefList[n][0]), pStr);
         TRACE(stderr," Adding %d to Ref table: %s.\n", n, pStr);
         n++;

         pStr = pStr2+2; /* skip null and space */
      }
      pStr = fgets(strLine, sizeof(strLine), fpRefFile);
      stripBlanks(pStr);
   }
```

```
   /* Find "configurable string message */
   while (strstr(strLine, "onfigurable") == NULL)
   {
      pStr = fgets(strLine, sizeof(strLine), fpRefFile);
   }
   if (strstr(strLine, "No configurable") != NULL)
   {
       TRACE(stderr," No configurable fields to add for %s\n.", strAction);
       return 0;
   }

   /* Get each parameter (comma separated) from configurable list */
   stripBlanks(pStr);
   pStr = strchr(pStr,':');
   pStr++;
   stripBlanks(pStr);
   /* Get each parameter (comma separated) from regular parameter list */
   /* - End when encountering a line with a colon or period */
   while (!feof(fpRefFile) && (strlen(pStr) > 0))
   {
      while (*pStr != NULL)
      {
         /* copy */
         pStr2 = strchr(pStr, ',');
         if (pStr2 != NULL)
            *pStr2 = NULL;
         else
            pStr2 = pStr + strlen(pStr);

         strcpy(&(arRefList[n][0]), pStr);
         TRACE(stderr," Adding config %d to Ref table: %s.\n", n, pStr);
         n++;

         pStr = pStr2+2; /* skip null and space */
      }
      pStr = fgets(strLine, sizeof(strLine), fpRefFile);
      stripBlanks(pStr);
   }

   return nRC;
}


/*-------------------------------------------------------------------------\
|  refVerifyEntry:                                                         |
|  - Compare entries in env list from ENV to those in reference file       |
\-------------------------------------------------------------------------*/
int refVerifyEntry(void)
{
   int  nRC = 0, nEnv = 0, nRef = 0;

   for (nEnv=0;
        ((nEnv < maxArgs) && (arEnvList[nEnv][0] != NULL));
        nEnv++
       )
   {
      for (nRef=0;
           ((nRef < maxArgs) && (arRefList[nRef][0] != NULL) &&
            (strcmp(&(arEnvList[nEnv][0]), &(arRefList[nRef][0]))!=0)
           );
           nRef++
          )
```

```
      {
         if ((nRef == maxArgs) || (arRefList[nRef][0] == NULL))
         {
           fprintf(stderr, "Error: ENV list entry %s in %s is not valid\n",
              &(arEnvList[nEnv][0]), strUEPath);
           nRC++;
         }
      }
   }
   return nRC;
}

/*-------------------------------------------------------------------------\
|  exitParseAction:                                                        |
|  - Parse just ActionName and ExitID                                      |
|    where:                                                                |
|    - ActionName: TeamConnection Action                                   |
|    - ExitID:   verified in range 0..3 and from contents of mue.ref       |
|  * Some code is redundant with exitParseCall                             |
\-------------------------------------------------------------------------*/
int exitParseAction(char *strActionExit, char *strAction, int *nID)
{
   char *pStr = NULL;
   int  nRC = 0;

   pStr=strActionExit;

   /* Get Action Name */
   stripBlanks(pStr);
   strcpy(strAction, "");
   sscanf(pStr, "%s", strAction);
   if (pStr == NULL)
   {
      fprintf(stderr," Error: Could not find Action Name.\n");
      nRC = 1;
   }
   else
      pStr = strchr(pStr+1, ' ');

   /* Get Exit ID */
   if ((nRC == 0) && (pStr != NULL))
   {
      stripBlanks(pStr);
      sscanf(pStr, "%i", nID);
      if (*nID < 0 || *nID > 3)
      {
         fprintf(stderr," Error: Exit ID not in range.\n");
         nRC = 2;
      }
      else
         pStr = strchr(pStr+1, ' ');
   }

   return nRC;
}

/*-------------------------------------------------------------------------\
|  exitParseCall:                                                          |
|  - Parse UserExitParm into the following: ActionName ExitID ExitName Other |
|    where:                                                                |
|    - ActionName: TeamConnection Action                                   |
|    - ExitID:   verified in range 0..3 and from contents of mue.ref       |
```

```
|    - ExitName: verified from contents of mue.ref (in the registered section) |
|    - UEPstring: whatever is left                                             |
|    * Once verified all fields are stored in global variables                 |
\------------------------------------------------------------------------------*/
int exitParseCall(char *strUEParm, char *strAction, int *nID, char *strExit,
    char *strOther)
{
   char *pStr, *pStr2;
   int  nRC = 0;

   pStr=strUEParm;

   /* Get Action Name */
   stripBlanks(pStr);
   strcpy(strAction, "");
   sscanf(pStr, "%s", strAction);
   if (pStr == NULL)
   {
      fprintf(stderr," Error: Could not find Action Name.\n");
      nRC = 1;
   }
   else
      pStr = strchr(pStr+1, ' ');

   /* Get Exit ID */
   if ((nRC == 0) && (pStr != NULL))
   {
      stripBlanks(pStr);
      sscanf(pStr, "%i", nID);
      if (*nID < 0 || *nID > 3)
      {
         fprintf(stderr," Error: Exit ID not in range.\n");
         nRC = 2;
      }
      else
         pStr = strchr(pStr+1, ' ');
   }

   /* Get Exit Name */
   if ((nRC == 0) && (pStr != NULL))
   {
      stripBlanks(pStr);
      if ((pStr == NULL) || (strlen(pStr)==0))
      {
         fprintf(stderr," Error: Could not find user exit name.\n");
         nRC = 3;
      }
      else
      {
         sscanf(pStr, "%s", strExit);
         pStr = strchr(pStr+1, ' ');
      }
   }

   /* Get User Defined Paramter */
   if ((nRC == 0) && (pStr != NULL))
   {
      stripBlanks(pStr);
      if ((pStr == NULL) || (strlen(pStr) < 2))
      {
         fprintf(stderr," Error: Could not find conditional expression.\n");
         nRC = 4;
```

```
      }
      else
      {
         strcpy(strOther, "");
         if (*pStr == '"')
         {
            pStr++;
            pStr2 = strchr(pStr, '"');
            if ((pStr2 != NULL) && (*pStr2 == '"'))
            {
               strncpy(strOther, pStr, (pStr2-pStr));
               *(strOther+(pStr2-pStr)) = NULL;
            }
         }

         if ((strOther == NULL))
         {
            fprintf(stderr," Error: Could not validate conditional expression.\n");
            nRC = 5;
         }
      }
   }

   /* Were all required parameters found? */
   if (pStr == NULL)
   {
      fprintf(stderr," Error: All required parameters not found.\n");
      nRC = 6;
   }

   if (nRC)
   {
       pStr = strrchr(strUEParm,'\n');
       if (pStr != NULL)
          *pStr = NULL;

       fprintf(stderr,"Error: User Exit Line (rc=%d): %s\n\
 did not have the required parameters for this program:\n\
 ActionName UserExitID UserExitName \"User Defined Parameter\".\n",
          nRC, strUEParm);
   }

   return nRC;
}

/*-----------------------------------------------------------------------------\
|  refVerifyFile:                                                              |
|  - For each entry in the UserExit file verify the entries using the output   |                          |
|    of the "teamc report -userExitInfo command                                |
\-----------------------------------------------------------------------------*/
int refVerifyFile(char *strMUECall)
{
   char strActionName[nMaxUEParm];
   char strExitName[nMaxUEParm];
   char strCondition[nMaxUEParm];
   int  nExitID;                        /* Current user exit id (0-3) */
   char strUEParmAction[nMaxUEParm];
   int  nUEParmID;
   char strLine[nMaxUEParm];
   char *pLine;
   int  nRC = 0, nTotalErrors = 0;
```

```
  nRC = ueInitFile();
  if (nRC)
  {
     return nRC;
  }

  /* Get A line and verify it */
  pLine = fgets(strLine, sizeof(strLine), fpUEFile);

  while (!feof(fpUEFile))
  {
     /* Initialize each loop */
     nRC = 0;
     stripBlanks(strLine);
     fprintf(stderr," Verifying ENV from UserExit: %s.\n", strLine);

     /* Make sure it is not a comment */
     /* If exit name found and not comment */
     if (*pLine != '#')
     {
        /* Get line out of userexit file */
        /* - False, do not check for condition in User defined parm */
        nRC = exitParseCall(strLine, strActionName, &nExitID,
              strExitName, strCondition);
        if (nRC)
        {
           fprintf(stderr, "Warning: Line in UserExit file: %s\n\
ignored: %s\n", strUEPath, strLine);
        }

        /* Populate environment list array for this call */
        if (nRC == 0)
        {
           nRC = envParseList(strLine);
        }

        /* Populate environment list array from reference report file */
        if ((nRC == 0) && (arEnvList[0][0] != NULL))
        {
           nRC = refParseList(strActionName, nExitID);

           if (nRC == 0)
           {
              /* Verify against teamc report -userExitInfo */
              /* - Both tablesw will be global */
              nRC = refVerifyEntry();
           }
        }
        else
           fprintf(stderr," No ENV parameters to verify.\n");

        /* Count MUE related errors */
        if (nRC != 0)
        {
              nTotalErrors++;
        }
     }
     else
     {
        TRACE(stderr," Comment in %s: %s\n", strMUEPath, strLine);
     }
     pLine = fgets(strLine, sizeof(strLine), fpUEFile);
```

```
   }

   /* Report summary of results */
   if (nTotalErrors > 0)
   {
      fprintf(stderr, "\
%s: %d error(s) found in %s, comparing to \"teamc report -userExitInfo\".\n",
         strMUECall, nTotalErrors, strUEPath);
      return 1;
   }
   else
   {
      fprintf(stderr, "%s successfully verified.\n", strUEPath);
   }

   fclose(fpMUEFile);
   fclose(fpUEFile);
   fclose(fpRefFile);
   return 0;
}

/*-----------------------------------------------------------------------------\
|  mueRun:                                                                     |
|  - Execute a single user exit from multexit file                             |
|  - Return output in a buffer                                                 |
|  - Pass back RC from system()                                                |
\-----------------------------------------------------------------------------*/
int mueRun(char *strNewCmd)
{
        int             rc = 0;
        int             parmIndex = 0, confIndex = 0, i = 0;
        short int       exec_rc;
        char            cmdToBeExec[OSmaxCommandLen] = "";
        struct          stat statBuff;
        int             maxReadLen = 0;
        int             oldStdOut = -1;
        int             oldStdErr = -1;
        int             stdOut = STDOUT;
        int             stdErr = STDERR;
        int             tmpFileHandle = -1;
        char            tmpFileName[maxPathLen] = "";
        int             tid;
        char            *p = NULL, *sp = NULL, *pSp = NULL, *pCmd = NULL,
                        *pBeginArg = NULL;
        int             maxCommandLen  = OSmaxCommandLen;
        int             maxArgumentLen = OSmaxCommandLen;
        int             nSp = 0;

        /*-------------------------------------------------------------+
        | Get starting information for generating a command and        |
        | Set up temporary file for environment variables              |
        +-------------------------------------------------------------*/
        /* Add name of environment variable file to output string */
        /* - Add in a safe spot, after command to be executed */
        sprintf(cmdToBeExec, "%s \"%s\" %s", strNewCmd, ArgV[1], ArgV[2]);

        /*-------------------------------------------------------------+
        | Feed in the action defined parameters to the "args" array.   |
        |                                                              |
        | Rules for copying from args array to output command string:  |
        | - each argument is enclosed in quotes so that they become    |
        |   separate arguments in the parameter list of the user's     |
```

```
|   user exit program                                            |
| - replace all quotes in arguments with backslash-quote         |
| - OS/2 limits:                                                 |
|   - cmd.exe has a line limit of 1024 characters.               |
|   - setting artificial max of 400 for each parameter           |
|     (only 2 fields can exceed 400, allowing 224 for rest)      |
| - Unix limits:                                                 |
|   - ksh will use an artificial limit of 32,000 chars.          |
|   - The limit for each parm is 16,000 (size of remarks)        |
| * Ellipse processing has already occurred in teamcd; stripped.|
+----------------------------------------------------------------*/

pCmd = cmdToBeExec + strlen(cmdToBeExec);
for (parmIndex = 3;
     (parmIndex <= ArgC) &&
     ((sp = ArgV[parmIndex]) != NULL);
     parmIndex++
    )
{
   /* first part of separator inserted before parameters in list */
   strcpy(pCmd, " \"");
   pCmd++; pCmd++;

   /* Remember where argument began for limit calculation */
   pBeginArg = pCmd;
   /* If space remains for argument */
   if (maxArgumentLen > 0)
   {
      /* process quotes and check length limits */
      if (strstr(sp, "\"") != 0)
      {
         /* Copy and insert back-quotes */
         for (pSp = sp;
              (pCmd < (pBeginArg + maxArgumentLen)) &&
              (*pSp != NULL);
              pSp++, pCmd++)
         {
            if (*pSp == '\"')
            {
               *pCmd = '\\';
               pCmd++;
               *pCmd = '\"';
            }
            else
               *pCmd = *pSp;
         }
         *pCmd = NULL;
      }
      else /* No quotes in argument */
      {
         nSp = (strlen(sp));
         if (nSp > maxArgumentLen)
         {
            strncpy(pCmd, sp, maxArgumentLen);
            *(pCmd+maxArgumentLen) = NULL;
         }
         else
         {
            strcpy(pCmd, sp);
            pCmd = pCmd + nSp;
         }
      }
```

```
   }
   /* second part of separator inserted after parameters in list */
   strcpy (pCmd, "\"");
   pCmd++;
} /* end for loop */

/*----------------------------------------+
| dup the stdout and std err file handles  |
| - save the current file handles to be    |
|   restored after the "system" call       |
+----------------------------------------*/
oldStdOut = dup(stdOut);
if (oldStdOut == -1)
{
   fprintf(stderr,"Error occurred during \"dup\", with errno=%d\n",
      errno);
   return 1;
}

oldStdErr = dup(stdErr);
if (oldStdErr == -1)
{
   fprintf(stderr,"Error occurred during \"dup\", with errno=%d\n",
      errno);
   close(oldStdOut);
   return 1;
}

/*----------------------------------------+
| Open a temporary file to save the output  |
| from the system() function                |
| - Using tmpnam to generate filename.      |
| - Open file so that handle can be dup'd   |
| - Open file so that handle can be dup'd   |
+----------------------------------------*/
strcpy(tmpFileName, "");
p = tmpnam(tmpFileName);
if (p == NULL)
{
   fprintf(stderr,"Error occurred during \"open\", with errno=%d\n",
      errno);
   close(oldStdOut);
   close(oldStdErr);
   return 1;
}

tmpFileHandle = open(tmpFileName, O_CREAT|O_TRUNC|O_RDWR,
   S_IREAD|S_IWRITE);
if (tmpFileHandle == -1)
{
   fprintf(stderr,"Error occurred during \"open\", with errno=%d\n",
      errno);
   close(oldStdOut);
   close(oldStdErr);
   return 1;
}

/*----------------------------------------+
| dup the pipe write handle to stdout and  |
| stderr.                                  |
+----------------------------------------*/
rc = dup2(tmpFileHandle, stdOut);
```

```
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"dup\", with errno=%d\n",
               errno);
            close(oldStdOut);
            close(oldStdErr);
            close(tmpFileHandle);
            return 1;
         }

         rc = dup2(tmpFileHandle, stdErr);
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"dup\", with errno=%d\n",
               errno);
            close(oldStdOut);
            close(oldStdErr);
            close(stdOut);
            close(tmpFileHandle);
            return 1;
         }

         /*-------------------------------------------+
         | Calling user exit program:                 |
         | - Error checking operating system specific |
         | - Return code is only 16 bits in OS/2, have|
         |   defined exec_rc as "short int"           |
         | - Errno is set to non-zero on failure of   |
         |   system(), at which you can test exec_rc  |
         |   for the reason.                          |
         | - If system succeeds, exec_rc is set to    |
         |   value returned from the user exit.       |
         | - The return code from the user exit needs |
         |   to be shifted right 8 bits.              |
         +-------------------------------------------*/
         errno=0;
         exec_rc= system(cmdToBeExec);
         if (errno != 0)
         {
            fprintf(stderr,"Error occurred during \"system\", with errno=%d\n",
               errno);
            strtok(cmdToBeExec, " ");

            /* Ran out of memory */
            if (errno == ENOMEM)
            {
               fprintf(stderr,"Error, not enough memory attempting command:\n%s\n",
                  cmdToBeExec);
            }

#ifdef __UNIX__
            /* Could not spawn process */
            if (errno == EAGAIN)
            {
               fprintf(stderr,"Error, could not start process for command:\n%s\n",
                  cmdToBeExec);
            }
#endif
            dup2(oldStdOut, stdOut);
            dup2(oldStdErr, stdErr);
            close(oldStdOut);
            close(oldStdErr);
```

```
            close(tmpFileHandle);
            return 1;
         }

#ifdef __OS2__
         /*----------------------------------------+
         | OS/2 returns the error SYS1041 (program  |
         | not found) in the return, not in errno.  |
         +----------------------------------------*/
         /* Could not find user exit */
         /* In OS/2 return code 1041 */
         if (exec_rc == 1041)
         {
            /* Insert NULL after command */
            strtok(cmdToBeExec, " ");
            fprintf(stderr,"Error (OS/2), could not find command:\n%s\n",
               cmdToBeExec);
         }
#endif


         /*----------------------------------------+
         | Return our copies of stdout and stderr   |
         | back to what they were when we started   |
         | this routine.                            |
         +----------------------------------------*/
         rc = dup2(oldStdOut, stdOut);
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"dup2\", with errno=%d\n",
               errno);
            close(oldStdOut);
            close(oldStdErr);
            close(tmpFileHandle);
            return 1;
         }

         rc = dup2(oldStdErr, stdErr);
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"dup2\", with errno=%d\n",
               errno);
            close(oldStdOut);
            close(oldStdErr);
            close(tmpFileHandle);
            return 1;
         }

         /*----------------------------------------+
         | We're done with the pipe so close the    |
         | read handle that we have to it.          |
         +----------------------------------------*/
         rc = close(tmpFileHandle);
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"close\", with errno=%d\n",
               errno);
            close(oldStdOut);
            close(oldStdErr);
            return 1;
         }
```

```
/*-----------------------------------------+
| Close old file handles                   |
+-----------------------------------------*/
close(oldStdOut);
if (rc == -1)
{
   fprintf(stderr,"Error occurred during \"close\", with errno=%d\n",
      errno);
   close(oldStdErr);
   return 1;
}

close(oldStdErr);
if(rc == -1)
{
   fprintf(stderr,"Error occurred during \"close\", with errno=%d\n",
      errno);
   return 1;
}

/*-----------------------------------------+
| Check for output from user exit program  |
| - If file contains any output, put in    |
|   message buffer.                        |
| - Make sure not to overflow the buffer   |
+-----------------------------------------*/
rc = stat(tmpFileName, &statBuff);
if (rc == -1)
{
   fprintf(stderr,"Error occurred during \"stat\", with errno=%d\n",
      errno);
   return 1;
}

if (statBuff.st_size > 0 )
{
   tmpFileHandle = open(tmpFileName, O_RDONLY, S_IREAD);
   if(tmpFileHandle == -1)
   {
      fprintf(stderr,"Error occurred during \"open\", with errno=%d\n",
         errno);
      return 1;
   }

   /*-----------------------------------------+
   | Compute maximum size of read:            |
   | - Size of buffer                         |
   |   * using msgLen as size of buffer, if it|
   |     changes, need to change value here   |
   | - Less amount of data already in message |
   |   buffer (hopefully 0)                   |
   | - Less 1 for NULL                        |
   +-----------------------------------------*/
   maxReadLen = sizeof(msg) - strlen(msg) - 1;
   maxReadLen = statBuff.st_size; /* read size of file */

   sp = msg + strlen(msg);
   rc = read(tmpFileHandle, sp, maxReadLen);
   if (rc == -1)
   {
      fprintf(stderr,"Error occurred during \"read\", with errno=%d\n",
         errno);
```

```
            return 1;
         }

         /* Append a Null */
         strcpy(sp+rc, "\0");

         close(tmpFileHandle);
         if (rc == -1)
         {
            fprintf(stderr,"Error occurred during \"close\", with errno=%d\n",
               errno);
            return 1;
         }
      }

      /*---------------------------------------+
      | If user exit fails, return error code  |
      | - return code does not require change  |
      |   for OS/2 and NT                      |
      +---------------------------------------*/
#ifdef __UNIX__
      exec_rc=exec_rc>>8;
#endif
      if(exec_rc)
      {
        fprintf(stderr,"Command Failed with return code: %d\n", exec_rc);
        return exec_rc;
      }
      return 0;
}

/*-------------------------------------------------------------------------\
|  mueVerifyFile:                                                          |
|  - Verify that at least one user exit is defined in mue file, and that   |
|    conditions are defined correctly for each exit in mue file.           |
|  - If a condition is defined, call exitVerifyConditions to verify conditions |
\-------------------------------------------------------------------------*/
int mueVerifyFile(char *strActionName, int nExitId, char *strExitName)
{
   char    strLine[nMaxUEParm] = "";
   char   *pLine = NULL, *pStr = NULL;
   int     nRC = 0, nTotalErrors = 0, nValueRC = 0;
   char    strMUEActionName[nMaxUEParm] = "";
   int     nMUEExitId = 0;
   char    strMUEExitName[nMaxUEParm] = "";
   char    strMUECondition[nMaxUEParm] = "";
   boolean fFound = False;

   nRC = mueInitFile();
   if (nRC)
      return nRC;

   /* Failed to open or rewind mue */
   if (feof(fpMUEFile))
   {
      fprintf(stderr," Error: With multexit file in mueVerifyFile: %s.\n",
         strMUEPath);
      return 1;
   }

   /* Loop through entries in mue file, looking for this one */
   /* - Find a user exit line, then compare to input */
```

```
   pLine = fgets(strLine, sizeof(strLine), fpMUEFile);
   while (!feof(fpMUEFile) && pLine != NULL)
   {
      /* Initialize Loop */
      nRC = 0;
      stripBlanks(strLine);
      fprintf(stderr,"  Comparing with multexit line: %s.\n", strLine);

      /* If not a comment */
      if (*pLine != '#')
      {
         /* parse to get action name and exitID */
         nRC = exitParseCall(strLine, strMUEActionName, &nMUEExitId,
               strMUEExitName, strMUECondition);

         if ((nRC == 0) &&
             (strcmp(strActionName, strMUEActionName)==0) &&
             (nExitId == nMUEExitId)
            )
         {
            if (strlen(strMUECondition) > 0)
            {
               /* Verify conditions for running a user exit, and if necessary */
               /* get environment variable */
               nRC = exitVerifyConditions(strMUECondition, &nValueRC);

               /* Check results of Conditions */
               /* - ignore nValueRC during verificaton */
               if (nRC == 0)
                  TRACE(stderr," Exit Conditions Verified.\n");
               else
                  fprintf(stderr," Error: Could not parse user exit conditions.\n");
            }
            else
            {
               fprintf(stderr,"  Info: No condition to verify; always run.\n");
            }
            if (nRC == 0)
               fFound = True;
         }

         if (nRC)
            nTotalErrors++;
      }
      pLine = fgets(strLine, sizeof(strLine), fpMUEFile);
   }

   /* Either teamc report failed, or no user exits configured */
   if (!fFound)
   {
      fprintf(stderr, "\
Error: Could not find valid ActionName/ExitID in %s\n\
         for Action %s, ExitID %d, ExitName %s\n",
         strMUEPath, strActionName, nExitId, strExitName);
      return nRC;
   }
   else
   {
      fprintf(stderr, "\
Completed verification of UE Parm: for...\n\
 Action=%s, ExitID=%d, ExitName=%s\n",
         strActionName, nExitId, strExitName);
```

```
      if (nTotalErrors)
      {
         fprintf(stderr, " %d error(s) found in %s.\n", nTotalErrors,
            strMUEPath);
      }
   }
   return 0;
}

/*-------------------------------------------------------------------------\
|  ueVerifyFile:                                                           |
|  - For each mue entry in UserExit file (look for calls to argv[0]        |
|    - Read contents of parameter file one line at a time and verify it    |
|    - call exitParseCall to verify calls in multExit file                 |
|  * This code assumes that the multExit file is in the same directory as the  |
|    userExit file.                                                        |
\-------------------------------------------------------------------------*/
int ueVerifyFile(char *strMUECall)
{
   char strActionName[nMaxUEParm];
   char strExitName[nMaxUEParm];
   char strCondition[nMaxUEParm];
   int  nExitID;                        /* Current user exit id (0-3) */
   char strUEParmAction[nMaxUEParm];
   int  nUEParmID;
   char strLine[nMaxUEParm];
   char *pLine;
   int  nRC = 0, nTotalErrors = 0;

   nRC = ueInitFile();
   if (nRC)
   {
      return nRC;
   }

   /* Get A line and verify it */
   fgets(strLine, sizeof(strLine), fpUEFile);

   while (!feof(fpUEFile))
   {
      /* Initialize each loop */
      nRC = 0;
      stripBlanks(strLine);
      fprintf(stderr, "Verifying from UserExit: %s.\n", strLine);

      /* Make sure it is not a comment, then check */
      /* Check for exit name */
      pLine = strstr(strLine, strMUECall);
      /* If exit name found and not comment */
      if ((pLine != NULL) && (*pLine != '#'))
      {
         /* Get line out of userexit file */
         /* - False, do not check for condition in User defined parm */
         nRC = exitParseCall(strLine, strActionName, &nExitID,
               strExitName, strCondition);
         if (nRC)
         {
            fprintf(stderr, "Warning: Line in UserExit file: %s\n\
 ignored: %s\n", strUEPath, strLine);
         }
         else if (strcmp(strMUECall, strExitName)==0)
         {
```

```
            nRC = exitParseAction(strCondition, strUEParmAction, &nUEParmID);
            if (nRC ||
                (strcmp(strActionName,strUEParmAction)!=0) ||
                (nExitID != nUEParmID)
               )
            {
               fprintf(stderr, "\
Error: ActionName and ExitID in User Defined Parameter must match\n\
       ActionName and ExitID parameters.\n");
            }

            /* Populate environment list array for this call */
            if (nRC == 0)
            {
               nRC = envParseList(strLine);
            }

            if (nRC == 0)
            {
               /* Verify contents of line: */
               /* - Are required parameters correct? */
               /* - If Keywords are specified, are they formatted correctly ? */
               /* - If Keywords are specified (for EnvFile) are they in UserExit? */
               /* - If Conditions are specified, are they valid */
               /* - Do not verify environment variables in userExit file */
               /*   since this is done by teamcd */
               nRC = mueVerifyFile(strActionName, nExitID, strExitName);
            }

            /* Count MUE related errors */
            if (nRC != 0)
            {
               nTotalErrors++;
            }
         }
         else
         {
            TRACE(stderr," Valid but different exit: %s\n", pLine);
         }
      }
      else
      {
         TRACE(stderr," Comment in %s: %s\n", strMUEPath, strLine);
      }
      pLine = fgets(strLine, sizeof(strLine), fpUEFile);
   }

   /* Report summary of results */
   if (nTotalErrors > 0)
   {
      fprintf(stderr, "%s: %d error(s) found in %s.\n",
         strMUECall, nTotalErrors, strUEPath);
      return 1;
   }
   else
   {
      fprintf(stderr, "%s successfully verified.\n", strUEPath);
   }

   fclose(fpMUEFile);
   fclose(fpUEFile);
   fclose(fpRefFile);
```

```
   return 0;
}

/*-------------------------------------------------------------------------\
|  teamcRun:                                                               |
|  - Read multExit file                                                    |
|  - Evaluate Other Parameters (provide sample conditional processing)     |
|  - Execute user exits                                                    |
\-------------------------------------------------------------------------*/
int teamcRun(void)
{
   char strActionName[nMaxUEParm];
   char strExitName[nMaxUEParm];
   char strCondition[nMaxUEParm];
   int  nExitID;
   char strMUEActionName[nMaxUEParm];
   int  nMUEExitId;
   char strMUEExitName[nMaxUEParm];
   char strMUECondition[nMaxUEParm];
   char strLine[nMaxUEParm], strLine2[nMaxUEParm];
   char *pLine;
   int  nRC = 0, nValueRC = 0, nTempRC = 0;

   /* Open MUE file */
   nRC = mueInitFile();
   if (nRC)
      return nRC;

   /* What user exit need to be called */
   strcpy(strLine, ArgV[1]);
   nRC = exitParseAction(strLine, strActionName, &nExitID);
   if (nRC)
   {
      fprintf(stderr, "Error: Could not parse input from userexit file.\n");
      return nRC;
   }

   /* Get Name of environment file */
   strcpy(strEnvPath, ArgV[2]);

   /* For Action/Exit passed in find all entries in multiExit file */
   pLine = fgets(strLine, sizeof(strLine), fpMUEFile);
   while (!feof(fpMUEFile) && (nRC == 0))
   {
      /* Initialize Loop */
      stripBlanks(strLine);
      TRACE(stderr," Comparing with multexit line: %s.\n", strLine);

      /* If not a comment */
      if (*pLine != '#')
      {
         /* parse to get action name and exitID */
         nRC = exitParseCall(strLine, strMUEActionName, &nMUEExitId,
               strMUEExitName, strMUECondition);
         if (nRC)
            return nRC;

         if ((strcmp(strActionName, strMUEActionName)==0) &&
             (nExitID == nMUEExitId))
         {

            if (strlen(strMUECondition) > 0)
```

```
            {
               /* Open environment file */
               nRC = envInitFile(strEnvPath);
               if (nRC)
                  return nRC;

               /* check condition to determine whether to run exit */
               if (exitVerifyConditions(strMUECondition, &nValueRC)==0)
               {
                  TRACE(stderr," Conditions passed, running.\n");
                  /* Run user exit and set nValueRC */
                  /* - checking nValueRC is done in exitVerifyConditions */
                  fclose(fpEnvFile);
                  nTempRC = mueRun(strMUEExitName);
               }
               fclose(fpEnvFile);
               fpEnvFile == (FILE *)NULL;
            }
            else
            {
               TRACE(stderr," No conditions, always run.\n");
               nTempRC = mueRun(strMUEExitName);
            }
            /* Update return code from mueRun */
            if (nTempRC > nValueRC)
            {
               nValueRC = nTempRC;
            }
         }
      }
      pLine = fgets(strLine, sizeof(strLine), fpMUEFile);
   }

   return 0;
}

/*-----------------------------------------------------------------------------\
|  main:                                                                       |
|  - If called from the teamcd process, validate call and run user exits       |
|  - If called with -C run verification routines for files                     |
|  - If called with -? or no parameters, display help                          |
\-----------------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
   int i, n, nRC;
   char *p, *pp;
   char strTmp[nMaxUEParm];
   char strUEShort[nMaxUEParm];

   ArgC = argc;
   ArgV = (char **)argv;

   /* Check environment variables */
   nRC = 0;
   if (getenv("TC_FAMILY") == NULL)
   {
      fprintf(stderr, "Error: Environment variable TC_FAMILY must be set.\n");
      nRC = 1;
   }
   if (getenv("TC_BECOME") == NULL)
   {
      fprintf(stderr, "Error: Environment variable TC_BECOME must be set.\n");
```

```
      nRC = 1;
   }
   strTCHome = getenv("TC_DBPATH");
   if (strTCHome == NULL)
   {
      fprintf(stderr, "Error: Environment variable TC_DBPATH must be set.\n");
      nRC = 1;
   }
   strTCTmp = getenv("TC_TMP");
   if (strTCTmp == NULL)
   {
      fprintf(stderr, "Error: Environment variable TC_TMP must be set.\n");
      nRC = 1;
   }

   /* Parse command line */

   strcpy(strTmp, argv[0]);
#ifdef __UNIX__
   p = strrchr(strTmp, '/');
#else
   p = strrchr(strTmp, '\\');
#endif
   if (p != NULL)
   {
      p++;
      pp = strchr(p, '.');
   }
   else
   {
      pp = strchr(strTmp, '.');
   }

   if (pp != NULL)
   {
      *pp = NULL;
   }

   if (p != NULL)
   {
      strcpy(strUEShort, p);
   }
   else
   {
      strcpy(strUEShort, strTmp);
   }

   /* Lower case name */
   for (p=strUEShort; *p!=NULL; p++)
   {
     *p = tolower(*p);
   }

   /* Display usage */
   if (
       (argc == 1) ||
       ((argc == 2) && (strcmp(argv[1],"-?")==0))
      )
   {
      usage(True);
   }
   else if (nRC > 0)
```

```
   {
      usage(False);
   }
   else if ((strcmp(argv[1],"-C") == 0) && (argc == 3) &&
            (argv[2][0]=='a' || argv[2][0]=='m' || argv[2][0]=='u')
           )
   {
      fCheck = True;
      switch (argv[2][0])
      {
         case 'a':
                   fprintf(stderr, "\
Checking integrity of userexit against daemon and multexit against userexit:\n");
                   nRC = refVerifyFile(strUEShort);
                   fprintf(stderr, "\
Checking integrity of mue configuration:\n");
                   n = ueVerifyFile(strUEShort);
                   nRC = nRC | n;
                   break;
         case 'm':
                   fprintf(stderr, "\
Checking integrity of mue configuration:\n");
                   nRC = ueVerifyFile(strUEShort);
                   break;
         case 'u':
                   fprintf(stderr, "\
Checking integrity of userexit against daemon:\n");
                   nRC = refVerifyFile(strUEShort);
                   break;
      }
      fclose(fpEnvFile);
      fclose(fpMUEFile);
      fclose(fpEnvFile);
      fclose(fpRefFile);
      remove(strRefPath);
      return nRC;
   }
   else if (argc < 4)
   {
      fprintf(stderr, "Error: Incorrect number of parameters, or.\n\
 incorrect parameter options. Contact your system administrator to \n\
 insure this program is being called by teamcd from a user exit definition.\n");
      usage (False);
   }
   else
   {
      TRACE(stderr," Normal user exit processing:\n");
   }

   /* Execute multiple user exits */
   nRC = teamcRun();
   /* Display output of user exits */
   printf("%s\n", msg);

   if (nRC)
   {
      TRACE(stderr, " Exiting.\n");
      return 1;
   }

   fclose(fpEnvFile);
   fclose(fpMUEFile);
```

```
   fclose(fpEnvFile);
   return (0);
}
/* End of File */
```

APPENDIX C. GETTING UPDATES

HOW TO GET ELECTRONIC COPIES OF MANUALS AND TRS

Many of the manuals and technical reports mentioned in this document 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://tc-cmvc.raleigh.ibm.com/cmvc`

From the index at the top of the page, select:

- Technical Reports and related tools
- Copies of ALL the archived versions for the forums CMVC and CMVC6000
- Documentation in PostScript files
- Documentation in ASCII text files

FTP

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

1. ftp tc-cmvc.raleigh.ibm.com
2. login as 'anonymous' and for password give your email address.
3. cd e:
4. cd cmvc/doc
5. binary
6. get fileName
7. quit

Internet

Web Home Page

Not available.

FTP

Downloading the code from an external FTP site, can be arranged by contacting the TeamConnection organization.

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
AIX, OS/2, IBM, DB2/6000, DB2, VisualGen, CMVC TeamConnection	IBM Corporation
UNIX, USL	UNIX System Laboratories, Inc.
Microsoft, Windows	Microsoft Corporation

END OF DOCUMENT °