# Making the most of VisualAge TeamConnection Version 2 user exits

Document Number TR 29.3032

Lee Perlov and Clifford Meyers

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

# ABSTRACT

This technical report provides guidance for family administrators developing VisualAge TeamConnection user exits in order to enhance their software configuration management (SCM) process. This document is organized by the types of problems that user exits solve, providing specific examples in a variety of programming languages.

## ITIRC KEYWORDS

- Software Configuration Management
- VisualAge
- TeamConnection
- User Exit

# 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. 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.

## CLIFFORD J. MEYERS

Mr. Meyers is an advisory programmer with the VisualAge TeamConnection development team. Previously, he was the technical team lead for IBM Global Services Distributed Configuration Management Services, one of the primary support organizations for CMVC and VisualAge TeamConnection users. He joined IBM in 1985 as support for AIX/370 and AIX/ESA before accepting his current assignment in 1993.

Cliff is currently a development engineer working on Unix platforms.

# CONTENTS

# INTRODUCTION

While VisualAge TeamConnection user exits are well documented in the **VisualAge TeamConnection Administrator's Guide**, and sample user exits are provided with the product in REXX, Korn Shell and C, there are no samples that address the typical problems that family administrators try to solve.

In order to make use of this document, it is necessary to have a working understanding of user exits. This information is provided in "Chapter 15: Providing user exits" and "Appendix C: Userexit Parameters" of the **VisualAge TeamConnection Administrator's Guide**.

This document focuses on areas that are not covered in the VisualAge TeamConnection documentation:

- Sample user exits that show how to solve real-world problems
- Sample user exits that access temporary files used in `teamc part` commands
- Accessing the database through the `teamc report` command
- Accessing the database through SQL queries
- Avoiding problems that commonly occur when issuing TeamConnection commands or integrating with other tools
- Debugging user exits
- A detailed comparison of user exits in VisualAge TeamConnection Version 2 and CMVC

# GENERAL SUGGESTIONS

There are many useful suggestions on user exits in the **VisualAge TeamConnection Administrator's Guide**. Here are more suggestions based on the experience of our enterprise family administrators and users:

- Use the environment file mechanism to make sure that you get the exact string that was entered. As stated in the Administrator's Guide, any value that contains a newline will truncate the parameter list on the command line for Windows NT and OS/2. Recent changes can cause the same problem in Unix. Using the environment file eliminates the risk of getting incorrect values. Many of the examples in this document use the environment file feature.
- If you are going to issue TeamConnection commands in your user exit, run extra daemons and use a mechanism like the code segment in “Techniques to avoid deadlock” on page 11.
- If there is significant processing to be done, use a high-level compiled language such as C. TeamConnection provides a sample user exit in C, `viewexit.c`. Also, compile using the optimizer option, for example `cc -O ...`
- Avoid using interpretive languages such as `perl`

Since most of our production families run on Unix, we tend to write most of our user exits in Korn shell. However, increasingly we are seeing families installed on multi-processor Intel-based workstations running Windows NT and OS/2. Therefore, many of the examples in this document are written in the most portable language currently available: C. Since VisualAge TeamConnection families can easily be moved to new servers, possibly including a change in operating systems, we recommend writing user exits with portability in mind.

# DEBUGGING TIPS

When a user exit isn't working properly, there are several techniques available to debug the problem:

- Set the TRACE environment variables before starting the TeamConnection daemon to start tracing. The trace will include the exact syntax of all calls to your user exit programs, along with all of the parameters passed to your user exit program. Complete documentation for using the TRACE facility is provided in the **VisualAge TeamConnection Administrator's Guide**.
- Use whatever trace facility is available in the language of your user exit program. For example:
  - In Korn shell, add `set -x` to turn on shell trace. There is an example of this in “Simple Unix user exit” on page 7.
  - In C, use the `perror` or `fprintf(stderr,...` to write information to standard error.
  - In DOS batch, omit the `@echo off` to allow echoing of all commands as they execute.
  - In REXX, add `TRACE ALL` to turn on command trace.
- If you have problems using an environment file, get a sample environment file and test your program independently of TeamConnection. You can use a simple user exit such as “Simple Windows NT and OS/2 user exit” or “Simple Unix user exit” on page 7 to copy an environment file to another location so that it will not be deleted when the user exit completes. Both of these examples have commented out lines that will copy the environment file.
- Try the user exit in a test family, like the "testfam" family, documented in the TeamConnection installation instructions. Once a user exit is know to the currently running TeamConnection daemon (i.e. in $HOME/config/userExit when the daemon starts), the program can be modified without restarting the daemon.

## SIMPLE WINDOWS NT AND OS/2 USER EXIT

The very simple user exit sample, `showParms.cmd`, displays each of the user exit parameters, including the environment file name. Here are the key points that make this sample useful for understanding and debugging user exit problems:

- Line 1 - Commenting (REM) will cause each line executed to be displayed in an information window when the user exit is run.
- Line 2 - Contains VisualAge TeamConnection keywords that indicate where the file came from (i.e. the context).
- Line 3 - Uncommenting will save a copy of the environment file so that you can test your user exit independent of VisualAge TeamConnection.

- Remaining lines display parameters and operate on the parameter list. Note that if one of the parameters is `Remarks`, which may contain a newline, the parameter list may be prematurely truncated.

```
@echo off
REM $KW=@(#); $FN=showParms.cmd; $ChkD=1998/03/21 07:44:33; $Ver=66:1; $EKW;
REM copy %2 envfile.%1
echo User exit parameters:
:again
if %1.==. goto done
echo Parm: %1
shift
goto again
:done
echo No more parameters.
```

## Output of showParms.cmd

Here is a sample output for the above sample user exit, run on a `teamc part -create` command.

```
User exit parameters:
Parm: "Create"
Parm: "C:\PROGRA~1\IBM\TEAMCO~1\testfam\tctmp\001640"
Parm: "readme.txt"
Parm: "C:\TEMP\2"
Parm: "NOTRACK"
Parm: "comp1"
Parm: "1"
Parm: "SpaceMountain"
Parm: "Initial Version of readme.txt"
Parm: "0600"
Parm: ""
Parm: ""
Parm: ""
Parm: "1"
Parm: ""
Parm: "TCPart"
Parm: "TCPart"
Parm: "no"
Parm: "minnie"
Parm: "0"
No more parameters.
```

## Registering showParms.cmd user exit

Here is the entry in the %TC_DBPATH%\config\userExit file that invokes showParms.cmd, including comments on each parameter:

```
#config/userExit file parameters:
#1. Action (e.g. PartAdd invoked for teamc part -create)
#2. Exit Number from 0 to 3 (e.g. 1 is to execute after initial checking)
#3. UserExit Command to be executed (e.g. showParms.cmd)
#4. UserExit Parameter (e.g. "Create" passed to user exit in first positional
#   parameter)
#   - tcadmin GUI always inserts UEParm.
#   - config/userExit does not require, but it is strongly recommended.
```

```
#   - Positional parameter list is shifted if UEParm is missing.
#5. Optional Environment Variable List (e.g. partpathName,
#   temporaryfileonserver,and filetype will be passed in an environment
#   file, C:\PROGRA~1\IBM\TEAMCO~1\testfam\tctmp\001640, to user exit
#   in second positional parameter
#6. Optional Comment (e.g. None used here, always starts with a #)
#
PartAdd     1 showParms.cmd "Create"  ENV=(partpathName,temporaryfileonserver,filetype)
```

## SIMPLE UNIX USER EXIT

This is the Korn shell equivalent to the DOS batch user exit in “Simple Windows NT and OS/2 user exit” on page 5. Uncomment the third line in order to get debug information, including information about values of commands and the results of test instructions. Uncommenting the fourth line will create a copy of the environment file so that you can test your user exit with an environment file, but without running TeamConnection commands.

```
#!/usr/bin/ksh
# $KW=@(#); $FN=showParms.ksh; $ChkD=1998/03/21 07:45:49; $Ver=66:1; $EKW;
# set -x
# cp $2 envfile.$1
print User Exit parameters:
while [[ "$1" != "" ]]
do
   print Parm: $1
   shift
done
print No more parameters.
```

### Output of showParms.ksh

Here is a sample output for the above sample user exit, run on a `teamc part -create` command.

```
User exit parameters:
Parm: ""
Parm: ""
Parm: "readme.txt"
Parm: "/tmp/xYor5H"
Parm: "NOTRACK"
Parm: "comp1"
Parm: "1"
Parm: "SpaceMountain"
Parm: "Initial Version of readme.txt"
Parm: "0600"
Parm: ""
Parm: ""
Parm: ""
Parm: "1"
Parm: ""
Parm: "TCPart"
Parm: "TCPart"
Parm: "no"
Parm: "minnie"
Parm: "0"
No more parameters.
```

### Registering showParms.ksh user exit

Here is the entry in the $HOME/config/userExit file that invokes showEnv.ksh, including comments on each parameter:

```
#config/userExit file parameters:
#1. Action (e.g. PartAdd invoked for teamc part -create)
#2. Exit Number from 0 to 3 (e.g. 1 is to execute after initial checking)
#3. UserExit Command to be executed (e.g. showEnv.ksh)
#4. UserExit Parameter (e.g. "" if no string is to be passed to user exit)
#   - tcadmin GUI always inserts UEParm.
#   - config/userExit does not require, but it is strongly recommended.
#   - Positional parameter list is shifted if UEParm is missing.
#5. Optional Environment Variable List (e.g. not used)
#6. Optional Comment (e.g. # Executed on Part create)
#
PartAdd     1 showEnv.ksh ""  # Executed on Part create
```

## SIMPLE UNIX USER EXIT ACCESSING ENVIRONMENT FILE

The following example contains a minor extension to the previous Korn shell script that does two useful things:

- Displays all of the values in the environment file
- Retrieves and displays a specific value (e.g. component)

In order to access the contents of the environment file, the sample program `teamcenv.c` should be compiled using instructions in the files comment block.

```
#!/usr/bin/ksh
# $KW=@(#); $FN=showEnv.ksh; $ChkD=1998/03/21 07:45:49; $Ver=66:1; $EKW;
# set -x
# cp $2 envfile.$1
print User Exit parameters:
while [[ "$1" != "" ]]
do
   print Parm: $1
   shift
done
#
#
if [[ $2 == "" ]]
then
   print No environment file, check $HOME/config/userExit
   exit 1
fi
print Contents of environment file:
teamcenv $2
#
set pathName=""
pathName= teamcenv $2 partpathName
if [[ $partName == "" ]]
then
   print Error: partpathName not set
```

```
   exit 1
else
   print partpathName value: $pathName
fi
exit 0
```

## Registering user exits

Here is the entry in the $HOME/config/userExit file that invokes showEnv.ksh, including comments on each parameter:

```
#config/userExit file parameters:
#1. Action (e.g. PartAdd invoked for teamc part -create)
#2. Exit Number from 0 to 3 (e.g. 1 is to execute after initial checking)
#3. UserExit Command to be executed (e.g. showEnv.ksh)
#4. UserExit Parameter (e.g. "" if no string is to be passed to user exit)
#   - tcadmin GUI always inserts UEParm.
#   - config/userExit does not require, but it is strongly recommended.
#   - Positional parameter list is shifted if UEParm is missing.
#5. Optional Environment Variable List (e.g. partpathName,
#   temporaryfileonserver,and filetype will be passed in an environment
#   file to user exit)
#6. Optional Comment (e.g. None used here, always starts with a #)
#
PartAdd     1 showEnv.ksh ""  ENV=(partpathName,temporaryfileonserver,filetype)
PartCheckIn 1 showEnv.ksh "" ENV=(partpathName,temporaryfileonserver,filetype)
```

# QUERYING AND UPDATING THE DATABASE

It is now possible, in many cases, to use the **teamc report** command to query the contents of the family from within a user exit. However, there are still some limitations:

- It is still possible to deadlock the database when trying to access data that is already locked
- Invoking a TeamConnection command will use a ***second*** daemon
- It may be necessary to schedule tasks to run outside the user exit, for performance or stability

Careful planning is necessary whenever TeamConnection commands are used inside a TeamConnection user exit. Therefore, teamc command calls should be tested.

## TECHNIQUES TO AVOID DEADLOCK

If a TeamConnection command deadlocks in a user exit everything stops! The only way to know whether an action will cause a deadlock is to timeout the command. Here is a technique that can allow a deadlock to timeout safely in Unix:

1. Start the TeamConnection command in background, redirecting the output to a temporary file.

   **Note:** Unix returns the process ID.

2. Run a wait loop, checking the status of the process periodically.
3. After a fixed time, if the command has not completed, kill it and end the user exit with a non-zero return code.
4. Upon successful completion of the command, continue processing.
5. In all cases, the output that was redirected to the file is displayed before removal of the temporary file

Here is a piece of Korn shell that will provide such a wait loop. It includes some sample processing to show you where you can insert your user exit code. The sample code is really there to help you understand how the routine manages the background task.

```
#!/bin/ksh
# Debugging, if needed
# set -x
#
# Input parameter processing
#
print Command line argument list: $*
if (( $# < 1 ))
then
   print For this example, 1 input parameter is required to Sleep Time,
```

```
   print for example 45, for 45 second sleep
   exit 1
else
   print Will sleep for $1 in this test
fi
#
# End input parameter processing
#
# TeamConnection command processing
#
# Set command to be executed in background here
command="sleep $1"
# Temporary filename for output of command
filename=/tmp/$$tmp.out
# Executing command, returning process ID
$command > $filename 2>&1 &
pid=$!
print Executing in background: $command
print Process ID: $pid
#
# loop 5 times, sleeping on each failed attempt
let i=0
let deadlock=1  # assume deadlock
# Allow about 1 minute for successful execution of command
while (( i < 5 )) && (( deadlock ))
do
   sleep 11        # sleep 11 seconds, then
   ps -fu $LOGNAME | cut -f3 -d' ' | grep $pid 1>/dev/null 2>&1
   # Check return code
   if (( $? ))
   then
      # Optional comment: print ...
      print Process $pid has terminated, exiting loop
      let deadlock=0  # no deadlock
   else
      # Optional comment: print ...
      print Process $pid still running, continuing to wait
      let i=i+1       # increment and try again
   fi
done
#
# Check for deadlock
if (( deadlock ))
then
   print Apparent deadlock on command: $command
   print Terminating user exit
   kill -9 $$
   cat $filename
   rm -f $filename
   exit 1
fi
#
# End of TeamConnection command processing
#
#
# Start other user exit processing here
#
print Completed processing
#
# End other user exit processing
#
# Start cleanup and exit
```

```
#
cat $filename
rm -f $filename
exit 0
```

## ASYNCHRONOUS EXECUTION OF TEAMCONNECTION COMMANDS

In Unix and on Windows NT, it is possible to schedule commands in a user exit to run independently of the user exit process. In other words, when the user exit is terminated, the scheduled command will run in its own shell.

Here is the Unix syntax for the `at` command:

```
print 'teamc report -view users -raw > /tmp/userlist' | at now
```

If the at job is going to be run from a script, the following syntax discards the output of the command

```
print 'teamc report -view users -raw>/tmp/userlist'|at now>/dev/null 2>&1
```

Here is the Windows NT syntax. Since the Windows NT `Schedule` service does not accept "now" as a parameter, we recommend using a C progam that can get the current time, then specify that time when executing the `at` command.

```
at \\%HOSTNAME% 00:00 "command string..."
```

**Notes:**

1. You need to set the `Schedule` service to start automatically.
2. When using `at` jobs, it is important to check the Windows NT Event Viewer frequently to make sure that the processes are running properly.

## TCSELECT QUERY TOOL

In addition to the `teamc report` command, `tcselect` allows for accessing data in the family using SQL syntax. The command is installed only on the server.

### tcselect syntax

```
tcselect ("QUERY_ARGUMENT")
```

where QUERY_ARGUMENT is a standard SQL select without the word "select", for example:

```
tcselect "* from users"
```

If no QUERY_ARGUMENT is provided, "tcselect" provides a prompt for entering a query, complete with "select", for example:

```
>tcselect
```

Enter a valid query string, for example:

```
select * from users
```

The query can include tables and fields not accessible by the `teamc report` command, such as:

```
select adName from DataAtlasPSB
```

The `tcselect` client command provides a generalized SQL query facility for TeamConnection. It is intended to be complementary to the `teamc report` command, in that it is possible to query on all data defined in TeamConnection tables and report all columns selected in the query. The `teamc report` command excludes some columns from reports and performs user authentication, only returning the records a user is authorized to see.

Tcselect command supports the same SQL rules as the -where clause in `teamc report`, including wildcards (% and _). Tcselect uses the environment variables TC_BECOME and TC_FAMILY for all commands. Further, tcselect uses the environment variable TC_RELEASE and TC_WORKAREA when a release or workarea context is required.

Upon successful completion, tcselect returns 0. Upon failure, tcselect returns 1 and outputs various diagnostic information to standard error.

**Notes:**

1. Family administrators may choose to change permissions on the tcselect executable to limit access to this command.
2. Quotes are required for QUERY_ARGUMENT only when wildcards are used that are interpreted by the command line processor (e.g. Unix ksh)
3. Tcselect is available only in English.

# ACCESSING PARTS FROM USER EXITS

TeamConnection commands that process file data, such as check-in and check-out, have access to temporary files on the server that contain the part data. Accessing these parts in a user exit running on the server allows the family administrator to perform a wide range of tasks helpful to the software configuration management process. Here are some examples:

- Copy files to a directory structure so that they can be accessed by file based tools
- Inspect the content of files in order to enforce coding standards
- Extend the process model by adding new processing based on the content of information in configurable fields.

**Note:** While it is possible to modify the contents of the temporary copy of a part in a user exit, we strongly discourage it. Changing part contents negatively impacts a user's impression that the library is a reliable means of saving EXACTLY what the programmer asked to be saved.

Here are some sample programs written in different languages that perform operations on the temporary files created during `teamc part -create` and `teamc part -checkin`.

## USER EXITS FOR PART SHADOWING

In this minimal set of user exits we demonstrate how to populate a directory structure with a **shadow** copy of the parts in a workarea. Shadowing allows administrators to create an instant backup of a workarea and users to execute tools that interrogate parts, such as `grep`.

This example only saves files that are updated in a workarea. The VisualAge TeamConnection development team uses these exits to verify change history while testing beta versions of TeamConnection. A complete shadowing solution would also include handling all of the `teamc part` commands, performing `teamc driver -extract` each time a driver is updated (i.e. adding/removing a driver member, or performing a part build) and performing a `teamc release -extract` each time a driver is committed.

This is a Unix implementation, using Korn Shell. For this set of user exits to work, the ${DIRNAME} variable must be set.

## TEAMC PART -ADD: PARTADD2

```
#!/usr/bin/ksh

##############################################################################
# Author  : Clifford J. Meyers
# Function: Shadow PartAdd action to server
##############################################################################
```

```
################################################################################
# NOTES: stdout prints take the form print "<<STATUS>> ..."
#        stderr prints take the form print -u2 "<<ERROR>> ..."
################################################################################

################################################################################
function initialize
{
      # program return code
      RC=0
      DIRNAME=/shadow/${FAMILY}/${RELEASE}/${WORKAREA}/ dirname ${PART}
      BASENAME= basename ${PART}
      BADTEMPFILE=/tmp/badtempfile.$$
}
################################################################################
function process
{
      mkdir -p ${DIRNAME} >/dev/null 2>&1
      cp ${TEMPFILE} ${DIRNAME}/${BASENAME} >/dev/null 2>&1
      if ( $? -ne 0 )
        then     cp ${TEMPFILE} ${BADTEMPFILE} >/dev/null 2>&1
            print "cp ${BADTEMPFILE} ${DIRNAME}/${BASENAME}" | mail        \
             -s "${HOSTNAME} ${FAMILY} ${PROG} failure" ${FA_ADMIN}
        else     print "<<STATUS>>  Part ${BASENAME} has been shadowed\c"
            print " to directory ${DIRNAME}"
      fi
}
################################################################################
# main
PROG= basename ${0}
PART=${2}
TEMPFILE=${3}
RELEASE=${4}
FILETYPE=${6}
WORKAREA=${7}
initialize
if ( -n "${TEMPFILE}" )
  then     process
fi
exit ${RC}
```

## teamc part -checkin: PartCheckIn2

```
#!/usr/bin/ksh
################################################################################
# Author  : Clifford J. Meyers
# Function: Shadow PartCheckIn action to server
################################################################################

################################################################################
# NOTES: stdout prints take the form print "<<STATUS>> ..."
#        stderr prints take the form print -u2 "<<ERROR>> ..."
################################################################################

################################################################################
function initialize
{
      # program return code
      RC=0
      DIRNAME=/shadow/${FAMILY}/${RELEASE}/${WORKAREA}/ dirname ${PART}
      BASENAME= basename ${PART}
```

```
      BADTEMPFILE=/tmp/badtempfile.$$
}
################################################################################
function process
{
      mkdir -p ${DIRNAME} >/dev/null 2>&1
      cp ${TEMPFILE} ${DIRNAME}/${BASENAME} >/dev/null 2>&1
      if ( $? -ne 0 )
        then     cp ${TEMPFILE} ${BADTEMPFILE}
            print "cp ${BADTEMPFILE} ${DIRNAME}/${BASENAME}" | mail        \
             -s "${HOSTNAME} ${FAMILY} ${PROG} failure" ${FA_ADMIN}
        else     print "<<STATUS>>  Part ${BASENAME} has been shadowed to \c"
            print "directory ${DIRNAME}"
      fi
}
################################################################################
# main
PROG= basename ${0}
PART=${2}
TEMPFILE=${3}
RELEASE=${4}
WORKAREA=${6}
initialize
if ( -n "${TEMPFILE}" )
  then     process
fi
exit ${RC}
```

## Registering user exits

In this example, there is no `ueParm` specified. This is allowed by VisualAge TeamConnection user exits, but not encouraged. It is not certain that this will be possible in the future. The impact of not including ueParm is that the argument list is shifted one (e.g $3 becomes $2, etc.), as you can see from this excerpt of the help text delivered in the sample userExit file in /usr/teamc/nls/cfg/En_US:

```
# ------------------------------------------------------------------------
#
# The following tables show the parameters passed to the user exit
# programs defined for the PartAdd TeamConnection action.  This information
# is provided so that you can make use of the parameters being passed to your
# user exit programs by the TeamConnection Server software.
#
# At the end of this file is an example of a user exit definition.  Add
# your own definitions to the end of this file.    For more information on
# user exits, refer to the IBM TeamConnection Server Administration and Installation
# guide.
#
#
# ===================
# PartAdd, ExitID = 0
# ===================
# ------------------------------------------------------------------------
# Argument Number     Argument Number       Argument Value
# in an executable    in a shell program
# source (e.g. C)
# ------------------------------------------------------------------------
#     arg[0]                  $0        User exit program name
#     arg[1]                  $1        User defined parameter - should be in quotes
```

```
#     arg[2]                $2       Environment file name - null if not used
#     arg[3]                $3       File path name
#     arg[4]                $4       '0' if the file will not be
#                                 transmitted; otherwise the
#                                 file will be transmitted.
#     arg[5]                $5       Client location
#     arg[6]                $6       Temp file on server
#
#     arg[7]                $7       Release name
#     arg[8]                $8       Component name
...
```

userExit file:

```
PartCheckIn   2 PartCheckIn2  # Part Checkin, exit 2
PartAdd          2 PartAdd2      # Part Add, exit 2
```

## VERIFY KEYWORDS IN SOURCE CODE AND TEXT FILES

We recommend using VisualAge TeamConnection keywords in all source code, including text documents, in order to insert version information into each file you manage. This allows you to check the version of a file, even after files are extracted and delivered to end users.

The `uekey` user exit interrogates the temporary file and verifies that the keywords defined in a coding standards document are properly included in each text file with an appropriate file extension in the file name (e.g. a.c, readme.txt, go.cmd).

This example is written in C, so it should be portable to all operating systems where the VisualAge TeamConnection daemon runs. However, it has only been tested on Windows NT.

### teamc part -create/-checkin: uekey

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

SAMPLE NAME:  uekey.c

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

COMPILATION:  cc -o uekey uekey.c

ENVIRONMENT VARIABLES:
                      none

DESCRIPTION:  This sample user exit inspects the contents of any file
              with the name matching:
                 *.h, *.c, *.cmd, *.bat, *.txt
              During checkin or create insure that appropriate
              TeamConnection keywords are included in the file,
              between $KW=@(#); and $EKW;, on one line.

              If they are not, or are in the wrong order, an
              explanation of how they should be used is returned,
              with the error message.
              * Handle different parameter lists for PartAdd and
```

```
              PartCheckIn
             * Handle different order of parameters by using
               envFile
             NOTE: When changing file type on CheckIn, user exit
                   reports current type, not new type!

*********************************************************************
*    (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 <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>

extern int errno;

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

/* Name of program, for print statements */
#define PROG_NAME "uekey"

#define TRUE 1
#define FALSE 0

/* Part Action file types */
/* - On part create text=1, binary=2, none=0 */
/* - On part checkin text=text, binary=binary ... */
#define CR_PART_BINARY "2"
#define CH_PART_BINARY "binary"

/* Global variables, created for performance */
#define LEN_PARM_VALUE 16001
char parameterValue[LEN_PARM_VALUE]; /* allow for max in TeamC 16000 for remarks + NULL */

/* Function Declarations */
void usageError(void);
void userHelp(void);
```

```
char *getEnvVal(char *inValue);
FILE *initFile(char *envFileName, char *parms);
int envGetFromEnvFile(FILE *envFile, char *parameterName, char *outValue);

/*-----------------------------------------------------------------------------\
|  usageError                                                                  |
\-----------------------------------------------------------------------------*/
void usageError(void)
{
   fprintf(stderr, "%s: Use of keywords in source files:\n\
   The following keywords are required:\n\
      $%s; $FN; $ChkD; $Ver; $EKW;\n\
   The string must contain these keywords in order, on one line.\n\
   Other TeamConnection keywords are optional.  It is recommended\n\
   that a keyword string be compiled into the code.  This\n\
   imbeds the value of the keywords so that they can be seen by\n\
   using the \"what\" (Unix) or \"tcwhat\" (Intel) command against\n\
   the executable.  This is a sample in C syntax:\n\
      char fnStr[]=\"$%s; $FN; $ChkD; $Ver; $EKW;\";\n\
   The values are expanded on extract, including \"teamc part -build\".\n\
   These are the keyword values for this file:\n\
      $KW=@(#); $FN=uekey.c; $ChkD=1998/03/21 07:47:18; $Ver=66:1; $EKW;\n\
   Please update this file and repeat part action.\n\n",
      PROG_NAME, "KW", "KW");
   return;
}

/*-----------------------------------------------------------------------------\
|  userHelp:                                                                   |
\-----------------------------------------------------------------------------*/
void userHelp(void)
{
   fprintf(stderr,"%s usage:\n", PROG_NAME);
   fprintf(stderr,"   %s user exit parameters ...\n", PROG_NAME);
   fprintf(stderr,"   Note: The UEParm must equal \"Create\" or \"CheckIn\"\n\n");

   usageError();
   return;
}

/*-----------------------------------------------------------------------------\
|  getEnvVal:                                                                  |
\-----------------------------------------------------------------------------*/
char *getEnvVal(char *inValue)
{
   char *outValue;

   outValue = getenv(inValue);
   if (outValue == NULL)
   {
      fprintf(stderr, "%s: Error, %s environment variable must be set\n",
         PROG_NAME, inValue);
      return NULL;
   }
   return outValue;
}

/*-----------------------------------------------------------------------------\
|  initFile:                                                                   |
|  - Return File Handle                                                        |
|  - temporary file automatically deleted when daemons killed                  |
\-----------------------------------------------------------------------------*/
```

```
FILE *initFile(char *envFileName, char *parms)
{
   FILE *envFile;
   /* Open temporary file */
   envFile = fopen(envFileName, parms);
   if (envFile == (FILE *)0)
   {
       fprintf(stderr,"%s: Error, could not open file \"%s\"\n",
          PROG_NAME, envFileName);
       return (FILE *)0;
   }

   return (envFile);
}

/*-------------------------------------------------------------------------\
|  envGetFromEnvFile:                                                      |
|  - Write an entry to environment file                                    |
|  - Write binary:                                                         |
|    size of parameter, parameter string, size of value, value string      |
\-------------------------------------------------------------------------*/
int envGetFromEnvFile(FILE *envFile, char *inName, char *outValue)
{
   int nNameLength;
   int nValueLength;
   char parameterName[MAX_PARM_NAME+1]; /* allow for maximum in TeamC (15 + NULL) */
   /* Defining parameterValue as global for performance */

   /* Search for parameter identified by inName */
   if (*inName == '\0')
   {
      fprintf(stderr,"%s: Error, require parameter name\n", PROG_NAME);
      return 1;
   }
   else /* Searching for one entry */
   {
      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 ((strcmp(inName, parameterName) != 0) && (!feof(envFile)))
      {
         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';
      }
      if (!feof(envFile))
      {
         /* Return the value of the parameter */
         strcpy(outValue, parameterValue);
      }
      else
      {
         strcpy(outValue, "");
         fprintf(stderr, "%s: Error, parameter \"%s\" not found\n",
            PROG_NAME, inName);
```

```
         return 1;
      }
   }

   return 0;
}

/*-----------------------------------------------------------------------------\
|  main:                                                                       |
|  - Get parameters from envFile for PartCheckIn and PartAdd User Exit 1       |
|    action                                                                    |
|  - Check keywords and provide guidance                                       |
\-----------------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
   FILE *envFile, *tempFile;
/*   char *family; */
   char envPathName[195], envTempFile[256], envFileType[7];
   char envPathNameLC[195];
   char aKeywords[][7]={"$KW","$FN","$ChkD","$Ver","$EKW",""};
   char *p;
   int  fFound, i, n, nParm = 0, rc = 0;

   /* If no parameters, print help text */
   /* Parameter 1, UEParameters string must equal "Create" or "CheckIn" */
   if ((argc < 3) ||
       ((strcmp(argv[1],"Create")!=0)&&(strcmp(argv[1],"CheckIn")!=0))
      )
   {
      fprintf(stderr, "%s: Error, Not enough parameters or the UEParm is not set properly.\n", PROG_NAME);
      userHelp();
      exit (1);
   }

   /* no environment variables required */
#ifdef NOTTODAY
   family = getEnvVal("TC_FAMILY");
   if (family == NULL)
   {
      fprintf(stderr, "%s: Error, Could not get one or more environment variable values\n",
         PROG_NAME);
      exit (1);
   }
#endif

   /* Parameter 2, envFile, must be set */
   /* Access parameter env file. */
   if (strlen(argv[2]) == 0)
   {
      fprintf(stderr, "%s: Error, ENV=(partpathName,temporaryfileonserver,filetype)\n\
   not defined in config/UserExit for this user exit.\n", PROG_NAME);
      userHelp();
      exit (1);
   }

   envFile = initFile(argv[2], "rb");
   if (envFile == (FILE *)0)
   {
      exit (1);
   }

   /* Get parameters needed for message from envFile */
```

```
/* NOTE: Using envFile is independent of keyword position */
rc =  envGetFromEnvFile(envFile, "partpathName", envPathName);
rc += envGetFromEnvFile(envFile, "temporaryfileonserver", envTempFile);
rc += envGetFromEnvFile(envFile, "filetype", envFileType);

if (rc > 0)
{
   fprintf(stderr, "%s: Error, envFile must be specified in config/UserExit\n",
      PROG_NAME);
   usageError();
   exit (1);
}

/* File must be text */
if (strcmp(argv[1],"Create")==0)
{
   if (strcmp(envFileType, CR_PART_BINARY)==0)
   {
      /* do not process for binary files */
      exit (0);
   }
}
else /* CheckIn */
{
   if (strcmp(envFileType, CH_PART_BINARY)==0)
   {
      /* do not process for binary files */
      exit (0);
   }
}

/* FileName must end with *.h, *.c, *.cmd, *.bat, *.txt */
/* - LowerCase the file name to match */
n = strlen(envPathName);
for (i=0; (i < n); i++)
{
   envPathNameLC[i] = tolower(envPathName[i]);
}
p = envPathNameLC+n;
*p = '\0';
if ((strcmp(envPathNameLC+n-strlen(".h"),".h")==0) ||
    (strcmp(envPathNameLC+n-strlen(".c"),".c")==0) ||
    (strcmp(envPathNameLC+n-strlen(".bat"),".bat")==0) ||
    (strcmp(envPathNameLC+n-strlen(".cmd"),".cmd")==0) ||
    (strcmp(envPathNameLC+n-strlen(".txt"),".txt")==0)
   )
{
   /* valid file type */
   rc = 0;
   fFound = FALSE;
   tempFile = initFile(envTempFile, "rb");
   if (tempFile == (FILE *)0)
   {
      fprintf(stderr, "%s: Error, temporary file on server, %s, could not be opened\n",
         PROG_NAME, envTempFile);
      usageError();
      exit (1);
   }
   /* Loop through each line, until a line has all of the keywords, or eof */
   /* - start by setting Found Flag to TRUE, then set FALSE on first failure */
   /* - if a line is parsed and the flag is still TRUE, exit */
   /* - if any line is only partial, set rc to first missing parm */
```

```
      /*   on last line with a partial match */
      p = fgets(parameterValue, LEN_PARM_VALUE, tempFile);
      while ((!feof(tempFile)) && (!fFound))
      {
         /* Parse file for "$KW $FN $ChkD $Ver $EKW", in order */
         /* - Other characters may exist in between words */
         /* - Process for each keyword in array */
         fFound = TRUE;
         for (i=0; ((strlen(aKeywords[i]) > 0) && (fFound)); i++)
         {
            p = strstr(p, aKeywords[i]);
            if (p == NULL)
            {
               nParm = i; /* if first keyword not found, rc = 0 */
               fFound = FALSE;
            }
         }
         /* Save any non-zero parm index for error message */
         if (nParm > 0)
         {
            rc = nParm;
         }
         /* Read next line */
         p = fgets(parameterValue, LEN_PARM_VALUE, tempFile);
      }
   }

   /* Determine if there was a failure when parsing file */
   if ((rc > 0) || (!fFound))
   {
      if (!fFound)
      {
         fprintf(stderr, "%s: Error, no line containing keyword string was found.\n",
            PROG_NAME);
      }
      if (rc > 0)
      {
         fprintf(stderr, "%s: Error, at least one like contained some keywords, but\n\
   keyword \"%s\" was missing or out of order.\n", PROG_NAME, aKeywords[rc]);
      }
      usageError();
      exit (1);
   }

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

## Registering user exits

In this example, the UEParm is different for the two user exits registered. Since this value is passed to the user exit program, it allows the user exit to perform different functions, handle differences in parameter lists for each action, etc. by checking the value of the UEParm.

```
PartAdd     1 ueKey "Create"  ENV=(partpathName,temporaryfileonserver,filetype)
PartCheckIn 1 ueKey "CheckIn" ENV=(partpathName,temporaryfileonserver,filetype)
```

## help message

```
uekey usage:
   uekey user exit parameters ...
   Note: The UEParm must equal "Create" or "CheckIn"

uekey: Use of keywords in source files:
   The following keywords are required:
      $KW; $FN; $ChkD; $Ver; $EKW;
   The string must contain these keywords in order, on one line.
   Other TeamConnection keywords are optional.  It is recommended
   that a keyword string be compiled into the code.  This
   imbeds the value of the keywords so that they can be seen by
   using the "what" (Unix) or "tcwhat" (Intel) command against
   the executable.  This is a sample in C syntax:
      char fnStr[]="$KW; $FN; $ChkD; $Ver; $EKW;";
   The values are expanded on extract, including "teamc part -build".
   These are the keyword values for this file:
      $KW=@(#); $FN=uekey.c; $ChkD=1998/03/21 07:47:18; $Ver=66:1; $EKW;
   Please update this file and repeat part action.
```

# INTEGRATING WITH OTHER TOOLS

VisualAge TeamConnection user exits can be used to integrate with other tools in your development environment. As discussed in “Asynchronous execution of TeamConnection commands” on page 13 and “Techniques to avoid deadlock” on page 11, it is recommended that any calls to other tools that will take a significant amount of time be run in background or scheduled to run later.

## NEW USER INFORMATION USER EXIT

Here are two examples:

- A C program that runs on Windows NT and sends each new user a note using Lotus Notes. For complete information on this user exit see the technical report **Evolution of a new TeamConnection Family: Taking advantage of automation (second in a series)**
- A Korn shell script that runs on AIX and sends a note using sendmail.

Both programs send a note that provides new user information about the family, so that they can be effective quickly.

**Notes:**

1. Use environment file feature
2. Use interface to Lotus Notes on Windows NT
3. Written in C, copying code from samples `viewexit.c` and `envfile.c`

### C program: newuser.c

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

SAMPLE NAME:  newuser.c

USAGE:        newuser

COMPILATION:  icc -O -o newuser newuser.c

ENVIRONMENT VARIABLES:
                       TC_FAMILY
                       TC_DBPATH

DESCRIPTION:  This user exit is designed to be registered for UserCreate, exit 2.
              Here is the recommended entry in the config/UserExit file
                 UserCreate 2 newusers "" ENV=(component,release) # New User Mail Message
              This program extracts the current contents of %TC_DBPATH%/motd, adds a
              standardized subject line and some framing, then invokes tcnotes.exe to
              mail the document using Lotus Notes.
```

```
NOTES:        1. The tcnotes.exe utility is required as the mail exit, and notifyd must
                 be running.
              2. The functions envInitReadEnvFile and envGetFromEnvFile are derived
                 from the TEAMCENV.C sample provided with TeamConnection.
              3. The function userHelp is derived from the help function in VIEWEXIT.C

*********************************************************************
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

/* Function Declarations */
void userHelp(void);
char *getEnvVal(char *inValue);
FILE *initFile(char *envFileName, char *parms);
int envGetFromEnvFile(FILE *envFile, char *parameterName, char *outValue);

/*-------------------------------------------------------------------------\
|  userHelp:                                                               |
\-------------------------------------------------------------------------*/
void userHelp(void)
{
   fprintf(stderr,"newuser usage:\n");
   fprintf(stderr,"\tnewuser                                -  This help message\n");
   fprintf(stderr,"\tnewuser UserCreate_ParameterList ...  -  Send motd to new user\n");
   return;
}

/*-------------------------------------------------------------------------\
|  getEnvVal:                                                              |
\-------------------------------------------------------------------------*/
char *getEnvVal(char *inValue)
{
   char *outValue;

   outValue = getenv(inValue);
   if (outValue == NULL)
   {
      fprintf(stderr, "newuser: Error, %s environment variable must be set\n",
         inValue);
      return NULL;
   }
   return outValue;
}

/*-------------------------------------------------------------------------\
|  initFile:                                                               |
|  - Return File Handle                                                    |
|  - temporary file automatically deleted when daemons killed              |
\-------------------------------------------------------------------------*/
FILE *initFile(char *envFileName, char *parms)
{
   FILE *envFile;
   /* Open temporary file */
   envFile = fopen(envFileName, parms);
   if (envFile == (FILE *)0)
   {
       fprintf(stderr,"newuser: Error, could not open file \"%s\"\n",
```

```
          envFileName);
       return (FILE *)0;
   }

   return (envFile);
}

/*-------------------------------------------------------------------------\
|  envGetFromEnvFile:                                                      |
|  - Write an entry to environment file                                    |
|  - Write binary:                                                         |
|    size of parameter, parameter string, size of value, value string      |
\-------------------------------------------------------------------------*/
int envGetFromEnvFile(FILE *envFile, char *inName, char *outValue)
{
   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 */

   /* Search for parameter identified by inName */
   if (*inName == '\0')
   {
      fprintf(stderr,"newuser: Error, require parameter name\n");
      return 1;
   }
   else /* Searching for one entry */
   {
      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 ((strcmp(inName, parameterName) != 0) && (!feof(envFile)))
      {
         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';
      }
      if (!feof(envFile))
      {
         /* Return the value of the parameter */
         strcpy(outValue, parameterValue);
      }
      else
      {
         strcpy(outValue, "");
         fprintf(stderr, "newuser: Error, parameter \"%s\" not found\n", inName);
         return 1;
      }
   }

   return 0;
}

/*-------------------------------------------------------------------------\
|  main:                                                                   |
```

```
|  - Get parameters from envFile for UserCreate User Exit 2 action             |
|  - Read %TC_DBPATH%/motd and create a new user mail message                  |
|  - Use Lotus Notes mail exit to mail new user message                        |
\------------------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
   FILE *envFile, *userFile;
   char *family, *dbpath, *userFileName;
   char envLogin[31], envFullName[63], envMailAddr[159];
   char command[1000];
   int  rc = 0;

   /* If no parameters, print help text */
   if (argc < 3)
   {
      userHelp();
      exit (1);
   }

   /* Check environment variables */
   family       = getEnvVal("TC_FAMILY");
   dbpath       = getEnvVal("TC_DBPATH");
   if (family == NULL || dbpath == NULL)
   {
      fprintf(stderr, "newuser: Error, Could not get one or more environment variable values\n");
      exit (1);
   }
   /* Parameter 1, UEParameters string, is not currently used                */

   /* Parameter 2, envFile, must be set                                      */
   /* Access parameter env file.                                             */
   if (strlen(argv[2]) == 0)
   {
      fprintf(stderr, "newuser: Error, envFile must be specified in config/UserExit\n");
      exit (1);
   }
   envFile = initFile(argv[2], "rb");
   if (envFile == (FILE *)0)
   {
      exit (1);
   }
   /* Get parameters needed for message from envFile */
   rc =  envGetFromEnvFile(envFile, "login", envLogin); /* Login ID */
   rc += envGetFromEnvFile(envFile, "usersfullname", envFullName); /* Full Name */
   rc += envGetFromEnvFile(envFile, "sendmailaddress", envMailAddr); /* mail address */
   if (rc > 0)
   {
      fprintf(stderr, "newuser: Error, Could not get one or more parameters from \"%s\"\n",
         argv[2]);
      exit (1);
   }

   /* Construct mail message */
   /* - Create temporary file */
   /* - Write subject line */
   /* - append motd */
   userFileName = tmpnam(NULL);
   userFile = initFile(userFileName, "wb");
   if (userFile == (FILE *)0)
   {
      exit (1);
   }
```

```
   fprintf(userFile, "Subject:  Welcome New TeamConnection User to %s\n", family);
   fprintf(userFile, "Welcome %s, your new userid is %s\n", envFullName, envLogin);
   fprintf(userFile, "--------------------------------------------------\n");
   fclose(userFile);

   sprintf (command, "type %s\\motd >> %s", dbpath, userFileName);
   rc = system(command);
   if (rc > 0)
   {
      fprintf(stderr, "newuser: Error, Could not execute command: %s\n", command);
      exit (1);
   }

   /* Send file to user */
   sprintf(command, "tcnotes.exe %s %s %s", userFileName, family, envMailAddr);
   rc = system(command);
   remove (userFileName);
   if (rc > 0)
   {
      fprintf(stderr, "newuser: Error, Could not send new user note\n\
command: tcnotes %s %s %s\n", userFileName, family, envMailAddr);
      exit (1);
   }

   return (0);
}

/* End of File */
```

## Korn shell script: UserCreate2

**Note:** This shell script is also a port from CMVC. As such, it does not use a UEParameter. As a result, the position of the parameters is shifted. Also, there are no keywords for tracking purposes.

```
#!/bin/ksh

function initialize
{
      IPADDRESS=$(host  hostname  | awk '{ print $3 }')
      IPNAME= hostname
      NOTETOSEND=/tmp/user.note.$$
      cp /usr/lpp/teamc/scm/userExits/usercreate.note ${NOTETOSEND}
}

function update_junk_mail
{
      awk 'BEGIN{print "%s/'TCLOGIN'/'${TCLOGIN}'/g";print "wq"}'            \
       /dev/null | ex ${NOTETOSEND} > /dev/null 2>&1
      awk 'BEGIN{print "%s/'FAMILYNAME'/'${FAMILY}'/g";print "wq"}'          \
       /dev/null | ex ${NOTETOSEND} > /dev/null 2>&1
      awk 'BEGIN{print "%s/'PORT'/'${TC_PORT}'/g";print "wq"}' /dev/null |   \
       ex ${NOTETOSEND} > /dev/null 2>&1
      awk 'BEGIN{print "%s/'IPADDRESS'/'${IPADDRESS}'/g";print "wq"}'        \
       /dev/null | ex  ${NOTETOSEND} > /dev/null 2>&1
      awk 'BEGIN{print "%s/'IPNAME'/'${IPNAME}'/g";print "wq"}' /dev/null |  \
       ex ${NOTETOSEND} > /dev/null 2>&1
}
```

```
# main processing routine
TCLOGIN=${1}
ADDRESS=${2}
initialize
update_junk_mail
mail -s "Welcome to TeamConnection" ${ADDRESS} < ${NOTETOSEND}
rm -f ${NOTETOSEND}
```

# VERY LARGE USER EXITS

It is possible for user exits to be very complicated. This section briefly discusses the `mue` user exit. Since the source file, `mue.c`, is over 2000 lines (including lots of comments and white space), it is a very large user exit sample. This user exit is fully documented in the technical report **29.2307 Data Driven TeamConnection User Exits**.

## INTRODUCTION TO MUE

The **mue** (multiple user exits) user exit 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.

## HOW TO USE MUE

The complete details of using `mue` are in the technical report **29.2307 Data Driven TeamConnection User Exits**. However, here is a short explanation of how to use 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.

The usage message for the mue program:

```
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.
```

An example of configuring a family to use mue:

This config/userExit file calls mue as the user exit known to TeamConnection.

```
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
```

Here is a config/multExit file. When called from the TeamConnection user exit interface, it evaluates the contents of variables passed through the environment file, then calls the appropriate user exit programs.

```
# 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 ?
```

# COMPARING VISUALAGE TEAMCONNECTION AND CMVC

You have seen how to use the new features available for VisualAge TeamConnection Version 2. Here is a quick summary of the differences between the current user exit interface and its predecessors.

## NEW TO TEAMCONNECTION

- Reliability and error handling have been enhanced, through the following changes:
  - Each parameter is quoted using apostrophes(') to prevent interpretation of the contents. Apostrophes(') within a parameter are converted to quotes(") in the positional parameters. The values are unchanged in the environment file.
  - In Windows NT and OS/2 imbedded quotes are backquoted.
  - Since each string is quoted, null strings are denoted by a pair of quotes(""). As a result, null strings do not change the position of parameters.
  - The administration tool that helps set up user exits quotes a single user exit parameter instead of writing each word separately. As a result, the number of parameters passed to the user exit is consistent.
- Flexibility:
  - The environment file is easier to use reliably than positional parameters on the command line. Hopefully, the examples in this technical report have convinced you of that.
  - User exit files must be on the path, but no longer need to be in the $HOME/bin directory.
- Ease of configuration:
  - Administration GUI simplifies the setup of user exits
  - The new report option, `teamc report -userExitInfo`, reports available user exits and their parameters. This includes available configurable fields.
  - The new report option, `teamc report -userExitInfo -long`, also reports registered user exits, including any use of environment files and UEParm values.
- Security
  - User exits run as family account. There is no more setuid to deal with, making invocation of user exits more predictable and controllable.
- Other differences
  - The order and list of parameters for each action have changed.
  - User Exit 2 is run after the database transaction has been committed. Therefore, returning a non-zero value in a user exit will NOT rollback a transaction.

– Comments in the sample userExit file are stripped from the configured $HOME/config/userExit file by `tcadmin` to improve performance. The commented file is still installed with VisualAge TeamConnection, but it is in the $HOME/nls/cfg/enu (for iso English) directory.

## MIGRATING FROM CMVC TO VISUALAGE TEAMCONNECTION V2

Other than changes to the order of parameters, CMVC user exits can run in TeamConnection. However, we recommend several changes to improve the maintainability of the code and the likelihood that the user exit will continue to perform as expected. You can test after each step to insure the code still behaves as expected.

1. Put quotes around all User Exit Parameters in the config/userExit file, so that they become a single UEParameter.
2. Convert existing programs to expect only 1 UEParameter.
3. Convert existing programs to set variables from the input parameter list instead of referencing positional parameters by their numbers. In other words, replace the use of positional parameters like $1 and $2 with assignments (e.g. let defectName = $2), then use those variables.
4. Copy code from the appropriate version of `viewexit` to access the positional parameters.
5. Replace the use of positional parameters with environment file values. You can use sample code in “Simple Unix user exit accessing environment file” on page 8, the program in the samples directory, `viewexit`, or compile the `teamcenv.c` program in the samples in order to extract values from the environment file.
6. Use either “Simple Windows NT and OS/2 user exit” on page 5 or “Simple Unix user exit” on page 7 to create an environment file to use during unit testing (i.e. without running `teamcd`), then
7. Configure $HOME/config/userExit to call your new user exit.
8. Convert keywords:

   A user exit that converts SCCS keywords used by CMVC into TeamConnection keywords is documented in the **VisualAge TeamConnection Administrators Guide**, in the chapter on migration from CMVC to TeamConnection.

## COMPARING TEAMCONNECTION V1 TO TEAMCONNECTION V2

The user exit facility in TeamConnection Version 1 was a unique implementation, unlike that of CMVC or VisualAge TeamConnection Version 2. If you have written any user exits for Version 1, you need to replace the way arguments are parsed. Otherwise, the body of your user exit should remain unchanged.

In TeamConnection V1, each parameter is separated by a pipe(|). While this works well for parsing in REXX, it does not allow programs written in other languages, such as C or Korn shell, to interpret the parameter list and assign the positional parameters. As a result, all of the values in the parameter list will be assigned to $1, unless there are spaces in the values of some of the parameters.

If you are parsing for a pipe in REXX, look at the sample program `viewexit.cmd` to see how REXX can parse the argument list in VisualAge TeamConnection V2.

If you are porting your TeamConnection V1 family server to another operating system, it is recommended that you use the sample programs `viewexit.c` or `viewexit.ksh` as guides to rewriting your user exit programs.

## COMPARING VISUALAGE TEAMCONNECTION V2 TO VISUALAGE TEAMCONNECTION V3

Since the next version of VisualAge TeamConnection has recently been announced, it is appropriate to highlight some of the expected impacts to user exits for current users of VisualAge TeamConnection Version 2:

- Replacement of the ObjectStore database with DB2

  As a result of replacing the current object-oriented database with a relational database, the `tcselect` command will no longer be needed for user exits. The database can be queried directly, without the overhead of executing a TeamConnection client command.

- Shadowing implemented as part of TeamConnection server

  The shadowing user exit example is an excellent teaching tool. However, a fully integrated shadowing facility will be available in VisualAge TeamConnection Version 3.

- More parameters have been added to many of the actions

  Many of the currently available actions, such as PartAdd, will have new parameters inserted near the end of the list. This will impact user exits that use positional parameters. For those using the environment file, there should be no impact.

- Fewer deadlocks for TeamConnection commands in user exit 2

  As a result of changing databases, as well as minor changes to the underlying transaction handling, user exit 2 is outside of the database transaction. This will reduce the occurrence of deadlocks when running TeamConnection commands in a user exit.

- New actions

  There are some new actions, and new configurable fields for releases and workareas. This provides new opportunities for user exits.

After VisualAge TeamConnection Version 3 becomes generally available, and our customers have a chance to provide feedback on any behavior changes in the user exit interface, we hope to provide an updated technical report.

# APPENDIX A. BIBLIOGRAPHY

## VISUALAGE TEAMCONNECTION PUBLICATIONS

For more information on how to use VisualAge TeamConnection, you can consult the following manuals:

SC34-4551 TeamConnection, Administrator's Guide
SC34-4552 Getting Started with the TeamConnection Clients
SC34-4499 TeamConnection, User's Guide
SC34-4501 TeamConnection, Commands Reference
SC34-4500 TeamConnection, Quick Commands Reference

## RELATED REDBOOKS

The following IBM redbooks provide practical advice about VisualAge TeamConnection from software specialists:

SG24-4648 Introduction to the IBM Application Development Team Suite
SG26-2008 TeamConnection Family and Application Development
SG24-4610 TeamConnection Workframe Integration Survival Guide

## RELATED TECHNICAL REPORTS

The following technical reports provide hints for using VisualAge TeamConnection:

29.2357 Evolution of a new TeamConnection Family: Taking advantage of automation (second in a series)
29.2267 TeamConnection frequently asked questions: How to do routine operating system tasks
29.2321 Comparison between TeamConnection 2 and CMVC 2.3
29.2307 Data Driven TeamConnection User Exits

# APPENDIX B. 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, CMVC, VisualAge TeamConnection, | IBM Corporation |
| ObjectStore | Object Design, Inc. |
| UNIX, USL | UNIX System Laboratories, Inc. |
| Acrobat, PostScript | Adobe Systems Incorporated |
| HP, HP-UX, SoftBench | Hewlett-Packard Company |
| Microsoft, Windows, Windows NT | Microsoft Corporation |
| X Window System | Massachusetts Institute of Technology |

**END OF DOCUMENT**