Revelation
Character to OpenInsight
Reference Guide

CHARACTER TO OPENINSIGHT™

REVELATION
S O F T W A R E
A Division of Revelation Technologies, Inc.

Part No. 125-826

Printed in the United States of America.

Table of Contents

Introduction

The new Character-To-OpenInsight (CTO) interface allows users of traditional multivalue applications to migrate to the graphical world of OpenInsight at their own pace. The CTO is designed to provide an intermediary platform, and as such emulates to a great extent the traditional multivalue environment. However, as this is an emulation rather than a full implementation, there are some differences that the multivalue user should be aware of. These will be discussed throughout this text where appropriate.

A note on terminology for traditional multivalue users: in OpenInsight, files are often referred to as "tables", and attributes are often referred to as "fields". You will find these terms used interchangeably throughout the documentation.

Connecting to the CTO

When OpenInsight is installed or upgraded on a users machine, several files relating to the CTO are also installed. The user communicates with the CTO using the REVCMD program, which is installed in the revcmd subdirectory under OpenInsight. The user may choose to make a "shortcut" to this program.

When invoked without any options, the revcmd program will attempt to communicate with the OpenEngine Server located on the current machine, at the default "socket port" of 8088. The user will then be prompted for the name of the application, the user name, and the password that is required to connect to that application.

Options that can be specified with revcmd can be displayed by typing in "revcmd -?" at the command prompt.

Alternatively, the revcmd interface has been “wrapped” in OpenInsight forms to allow for integration into the GUI environment. You may invoke the CTO_STANDALONE_FORM, or incorporate the CTO_CHILD_FORM in your own MDI application. Using the CTO_STANDALONE_FORM or CTO_CHILD_FORM allows you to use the CTO integration routines (described in the section “Interactions between OpenInsight and CTO” at the end of this manual).

If users wish to access the CTO environment on remote systems, a “telnet server” program is required. Revelation Software provides a telnet server specifically for use with the CTO, which must be installed and configured for remote access. For additional information, please see the section Technical Configuration for CTO.

When finished with the CTO environment, users may specify QUIT as the name of the application, and the revcmd program will terminate.

Using the CTO

Once connected and "logged in" to an application, users may invoke any stored procedures that have been written or converted for the CTO environment; PROCs or PAragraphs or Sentences; or the built-in commands, described in more detail below.

Note that, for commands containing dashes in their name (for example, "T-SELECT") users may specify _either_ the name with dashes, or the name with periods ("T.SELECT"), and the CTO will find the proper command.

The list of built-in commands currently includes:
- File (Table) and item commands: CREATE.FILE, CREATE.BFILE, CLEAR.FILE, COPY, DELETE, QSELECT, SELECT, SSELECT, COUNT, LIST, SORT;
- Media commands: ACCOUNT.RESTORE, ADD.ACCOUNTS, T.SELECT, T.READ, SEL.RESTORE, T.BACK, T.FWD, T.DUMP, T.LOAD, T.RDLBL, T.REW;
- Print management commands: STARTSPOOLER, STARTPTR, SP.ASSIGN, SP.EDIT, SP.OPEN, SP.CLOSE, LISTPTR, LISTPEQS, SP.KILL;
- Development commands: BASIC/COMPILE, CATALOG, RUN, ED;
- Process control commands: LISTE/LISTU, WHO, RESET.ENGINE, LOGTO, LOGON, OFF, QUIT, TERM;
- Information commands: LISTDICT, LISTFILES, VERSION

When any command or procedure generates output to the screen, this output may be paginated. If more than one page of output is generated, the CTO will "pause" at the bottom of the screen to accept the users response. Valid options include:
- <RETURN> to continue to the next page;

- Q or ^X to halt the output (and the program that is generating the output);
- ^F to find specific text in the output. You will be prompted for the search string, and output will be suppressed until the desired text is found
- N to continue in "no page" mode;

- any positive number (i.e., "10") to move to that page of the output;
- "+" or "-" followed by any number (i.e., "+5" or "-3") to move forwards or backwards that number of pages

When the output has completed, the user will be presented with a final pagination mark (an asterisk, "*") - this will allow them to move back through the output, if desired. Merely pressing <RETURN> one final time will return to TCL.

Standard Files Supplied In the CTO

When a new application is created (either through the OpenInsight graphical Presentation Manager, or through an account- or file-restore in the CTO), a table named MD (for Master Dictionary) is created. The MD table may contain the following types of items:

- "Q-pointers": these items (designated by a "Q" in field 1) allow users to create "synonyms" for their tables. The name of the item is the "synonym" name (the name by which you wish to refer to the table); field 1 contains a "Q", and field 3 contains the real table name. Field 2 contains either nothing (indicating that this q-pointer defines a table in the current application, or the name of the application where this table is defined.
- "V"erb definitions: these items (designated by a "V" in field 1) define certain "verbs" in the CTO environment.
- Control items: these items (usually designated by an "X" in field 1) are used to control the behavior of the CTO environment. Some of the control items you may wish to modify include:

a) PROCPATH: This item contains a list of the files the CTO environment should search when looking for a PROC or PAragraph to invoke (note that MD need not be listed here; the MD is always searched first). Field one should be “X”, and fields 2 through <n> contain the names of the files to process;
b) CASE.MODE: This item contains an “X” in field 1, and a "1" in field 2 to indicate "case sensitive" operations, or a "0" in field 2 to indicate "case insensitive" operations. (Note that this only applies to the command name itself; additional parameters may still behave in a case-sensitive fashion);
c) TODO: This item (created after an account- or file-restore) indicates files and items that require user attention;
d) Translation items (named <yourname>*TRANSLATE): this is used during account- or file-restore to specify the locations for newly-created files, and how such files should be created;
e) FORMATVARS: This optional item contains an “X” in field 1, and fields 2 through <n> contain the names of “formatting” variables used in basic+ programs. Specifying a variable name in this list allows the CTO precompiler to properly identify the variable and convert it to an OpenInsight-appropriate format;
f) CATALOG.ITEMS: This item contains a list of the cataloged pointers found during the account-restore operation. Field 1 contains an “X”, field 2 contains the list of tables, and field 3 contains an associated submultivalued list of the items that were cataloged (on the original system) for each table.

- "Catalog" pointers: these items (designated by a "P" in field 1) indicate that the associated BASIC+ routine has been compiled and 'cataloged' (that is, it may be invoked just by the name of the BASIC+ program - non-cataloged programs must be invoked with RUN). Field 5 contains the name of the table the BASIC+ routine has been cataloged from; field 4 contains the date and time of the cataloging; field 3 contains a "G" if this is a "GLOBAL" catalog entry.

Additionally, the MD table may contain PROCs, PAragraphs, or Sentences. These are "batch files" that are often used as simple "job control language" routines. They may reside either in the MD, or in any other table. If the PROCPATH control item has been set appropriately, typing in the name of the PROC, PAragraph, or Sentence at TCL will invoke the routine regardless of its actual location; if these routines reside in a table that is _not_ specified in PROCPATH, then an entry in the MD (or in another table that is listed in PROCPATH) must be created to transfer control to the routine.
g) RESTORED.DPOINTERS: This item contains a list of the “d-pointer” (file definition information) items found during the account-restore operation. Field 1 contains an “X”, field 2 contains a multivalued list of the dictionary name of the d-pointer, field 3 contains a multivalued (associated) list of the data name of the d-pointer, and fields 4 throught 14 (or beyond) contain a multivalued (associated) list of the d-pointer data.
h) SAVE.CONVERTED: This optional item contains an “X” in field 1, and a “0” in field 2 to instruct the CTO compiler to NOT write a copy of the converted BASIC source code in the SYSLISTS table. If this item is not present, or a “1” is in field 2, then the CTO compiler will write a copy of the converted BASIC source code into the SYSLISTS table, with the item id<tablename>_<programname> (e.g. “BP_SAMPLE”)
i) FLAVOR: This item identifies the emulation type desired in this CTO account. Field 1 contains an “X”, and field 2 contains the ‘flavor type’ (R83 by default). Valid values are currently R83, D3, and OI.

PROC

PROCs are procedural "batch files", with some control functionality. The CTO supports most of the standard PROC commands, with the exception of user exits. The use of PROCs is discouraged, and BASIC+ or OpenInsight routines should be used instead.

PAragraphs and Sentences

PAragraphs are procedural "batch files", with some control functionality. The CTO supports most of the standard PAragraph commands, with the exception of the formatting and conversion options on in-line prompts. The use of PAragraphs and sentences is discouraged, and BASIC+ or OpenInsight routines should be used instead.

CREATE.FILE and CREATE.BFILE

The CREATE.FILE command has the following syntax:

```
CREATE.FILE {DICT} <filename> {<location>} {(<options>}
```

This will create the specified file (either/both DICTIONARY and DATA portions, or DICTIONARY only) at the specified location. If no location is specified, the CTO will attempt to determine a location, and will then allow you to accept or change this location.

The CTO will also validate the name of the file; if it does not conform to OpenInsight requirements, a new suggested name will be returned, and you may choose to accept or change this (note that you may always create a "q-pointer" in the MD for your invalid file name).

If the "C" option is specified ("CREATE.FILE FILENAME (C"), the CTO will mark this as a BASIC+ program file. Compilation (with the BASIC command) will then be possible for this file.

The CREATE.BFILE command is identical to the CREATE.FILE command, except that it automatically includes the "C" option to indicate the creation of a BASIC+ file.

CLEAR.FILE

The CLEAR.FILE command has the following syntax:

```
CLEAR.FILE {DICT | DATA} <filename> {(I}
```

All items in the specified file will be deleted. Note that DATA is a "throwaway" word, and is ignored. If the I option is specified, no printed output is generated by the command.

COPY

The COPY command allows you to copy items within a file, copy items between files, and display items. The syntax for the COPY command is:

```
COPY {DICT} <filename> {<itemname> {<itemname> <itemname> ...} | *} {({O}{D}{I}{P|T}{S}}
```

The COPY routine will copy the specified item or items, or all the items in the file, from the specified source file. If a select list is active, the item list may be omitted. If DICT is specified, items will be copied from the dictionary; otherwise, items will be copied from the DATA section of the file.

If the “T” or “P” options are not specified, you will be prompted for a destination when you enter this command. If you specified the “T” option, or press <return> in response to the destination prompt, the specified items will be displayed on the screen. If the “P” option is specified, the specified items will be printed to the currently-active printer. If the “S” option is not specified, line numbers for each field will also be displayed.

You may specify a destination filename, by prefixing it with "(" - for example:

```
COPY BP *
To: (NEWBP
```

You may also specify new item names, in a one-to-one match with the source item names - for example:

```
COPY BP PROG1 PROG2 PROG3
To: NEWPROG1 NEWPROG2 NEWPROG3
```

By default, new items cannot overwrite existing items; if you wish to overwrite existing items, specify the "O" option. If you wish the original items to be deleted after conversion, specify the "D" option.
The COPY command normally displays information about the items it has copied. Specify the "I" option to suppress this output.

DELETE

The DELETE command has the following syntax:

DELETE {DICT} <filename> {<itemid> {<itemid> <itemid> ... } | * }
The specified item or items (or all items, if "*" is specified) will be deleted from the specified file.

T.SELECT

The T.SELECT command displays and optionally changes the currently-selected "virtual media" location. You may type in T.SELECT by itself, or optionally followed by the location (path and file name) you wish to use for the virtual media. If no location is specified, then the current location (if any) will be displayed, and you will be given the option to accept the current location. If you specify a new location, you will be prompted for the type of the media (either "D3" or "R83" compatible - note that "other" is currently unsupported). If the specified file does not exist at the specified location, you will be asked to verify your entry.

T.SELECT will also reset the location pointer in the "virtual media" back to position 0 (the beginning of the media) for subsequent use.

T.READ

The T.READ commands displays the information in the "virtual media" (as specified by T.SELECT).
You may optionally specify the "X" option ("T.READ (X") to display the information in both ASCII and HEX.
You may optionally specify the “F” option (“T.READ (F”) to search the virtual media data for a specified string. The T.READ process will prompt you for the string to search for, and will highlight (in reverse video) the specified text if it is found.

Tape Movement: T.REW, T.BACK, and T.FWD

The tape movement commands move the location pointer in the "virtual media" back to position 0 (T.REW), back one or more file blocks (T.BACK), or ahead one or more file blocks (T.FWD). Note that you must have previously selected the "virtual media" location with T.SELECT.

T.DUMP

The T.DUMP command has the following syntax:

T.DUMP {DICT} <filename> {<itemlist>} {(<options>}

This will write the selected items to the currently-selected 'virtual' tape media. You may specify an explicit list of item IDs to write, an "*" to indicate all items, or invoke this command with an active select list. You may optionally specify "I" to suppress the display of the item IDs that are being processed, or "B" to indicate that these are BASIC+ routines that should be read from the Repository (note that you still must specify a valid filename for Repository items, but the filename is ignored).

T.LOAD

The T.LOAD command has the following syntax:
T.LOAD {DICT} <filename> {(<options>}

This will load in items from the currently-selected 'virtual' tape media file into the specified filename. You may optionally specify "O" to overwrite already-existing items in the file; "I" to suppress the display of item IDs that are being loaded; or "B" to specify that these are BASIC+ routines that should be loaded into the Repository

T.RDLBL

The T.RDLBL command reads and displays the tape label from the "virtual media" specified by T.SELECT.

ACCOUNT.RESTORE and ADD.ACCOUNTS

The ACCOUNT.RESTORE command allows you to restore a single account from an ACCOUNT-SAVE or FILE-SAVE "virtual" tape. ADD.ACCOUNTS allows you to restore all accounts from a FILE-SAVE "virtual" tape. Note that you must already have the virtual tape selected through the T.SELECT command.

The syntax for the ACCOUNT.RESTORE command is:
ACCOUNT.RESTORE <accountname> {(E{1|2}{O}{L}{S}}

Where <accountname> is the name of the application you wish to create. You will be prompted for the name of the account on the virtual tape. If you specify the E, E1, or E2 options, the ACCOUNT.RESTORE process will load the data into an already-existing application; for correct operation, you must run ACCOUNT.RESTORE from within the desired application. Specifying the “E1” or “E2” option instructs the ACCOUNT.RESTORE process not to load data into any dictionaries (thus preserving any manual changes). Specifying the “E2” option also deletes any items in data files, allowing for a “clean data load” from the virtual tape. Alternatively, or in addition, specifying the “O” option will allow for overwriting of the data (E1 option) or both dictionary and data (E2 option) items. Specifying the “S” option will suppress any post-processing dictionary conversions. Specifying the “L” option will display all account and file names, even those that are not selected for processing.

The syntax for the ADD.ACCOUNTS command is:

ADD.ACCOUNTS

Since legacy multivalue accounts have different naming conventions than OpenInsight, and OpenInsight requires that you specify the physical location for your tables, you must somehow indicate what name and location the legacy account, and what names and locations the legacy files, should "map" to. Therefore, for both the ACCOUNT.RESTORE and ADD.ACCOUNTS, you will be prompted to enter the name of the "translation item" you wish to use.

All accounts that are created through the ACCOUNT.RESTORE and ADD.ACCOUNTS processes must conform to OpenInsight account naming convetions. If the original account name contained characters that are not acceptable in OpenInsight application names, an entry will be created in the item SYSTEM in the SYSENV file. This item contains an “X” in field 1, the original account names multivalued in field 2, and the OpenInsight application names in field 3 (associated with multivalues in field 2).

Translation items reside in the MD file, and are named <itemname>*TRANSLATE. A default translation item (named DEFAULT*TRANSLATE) is included with the CTO; however, you should create your own translation items that has been customized for your environment.

The format of the translation item is as follows:
ID: <itemname>*TRANSLATE
1. X2
2. <INCOMING account names>]
3. <Action to take - 0=skip, 1=load>]
4. <NEW account name to use>]
5. <DEFAULT physical location for files in this account – use “*” to represent app name>]
6. <Application to inherit from (null = SYSPROG)]>

10. <Action to take for accounts NOT in field 2 - 0=skip, 1=load>
11. <characters to prefix>]<characters to replace>]<new characters to use>]<characters to suffix>
12. <DEFAULT physical location for files in accounts NOT in field 2 – use “*” to represent app name>
13. <Application to inherit from (null = SYSPROG)>

20. <INCOMING filename - either ACCOUNT*FILE or just *FILE>]
21. <Action to take - 0=skip, 1=load, 2=load into repository>]
22. <NEW filename to use (or extra repository info)>]
23. <Physical location for this file – use “*” to represent app name, “&” to represent file name>]
24. <Item handling - null=none, 0=uppercase ID, 1=lowercase ID, (xxx)=call routine xxx>]

30. <Other file names - either ACCOUNT* or just *>]
31. <Action to take for files NOT in field 20 - 0=skip, 1=load, 2=load into repository>]
32. <characters to prefix>\<characters to replace>\<new characters to use>\<characters to suffix> (or extra repository info)]
33. <Physical location for files NOT in field 20 – use “*” to represent app name, “&” to represent file name>]
34. <Item handling - null=none, 0=uppercase ID, 1=lowercase ID, (xxx)=call routine xxx>]

40. <File name for log><space><item name for log>

Note: If you only specify the file name for logging in field 40 of the TRASLATE item (rather than both the file name and item id), then a logging item will be created with an item id of <internal date>*<internal time> in the specified logging file.

SEL.RESTORE

The SEL.RESTORE command allows you to selectively restore dictionary or data items from an account save or file save "virtual" tape. It has the following syntax:

SEL.RESTORE {DICT} <filename> <itemlist> {(O|I}

You must be "logged in" to the account where you wish to restore this data, and you must specify a list of items to restore, or "*" to specify all items in the file. You will be prompted to enter the account and file names from which the data should be retrieved. You may optionally specify the "O" option to overwrite any existing items with the same item ID, and/or the "I" option to suppress the display of the item IDs that are being restored.

REFORMAT/SREFORMAT

The REFORMAT and SREFORMAT commands copy and convert data, either from one file to another, or within a single file. REFORMAT and SREFORMAT work identically; however, SREFORMAT first sorts the data by any “sort criteria” you may specify.

The syntax for the REFORMAT command is:

```
REFORMAT {DICT}<file>{<selection criteria>}<fieldname>{<fieldname>…}{(I}
```

The syntax for the SREFORMAT command is:

```
SREFORMAT {DICT}<file>{<sort criteria>}{<selection criteria>}<fieldname>{<fieldname>…}{(I}
```

For both REFORMAT and SREFORMAT, <selection criteria> and <sort criteria> are in the same format as you would use in a “regular” Retrieve/English/Access statement; {DICT} is the optional modifier DICT; and <file> is the name of the file you wish to reformat. Each <fieldname> that you specify as “output criteria” becomes a filed in the converted output file. The value generated by the first <fieldname> becomes field #1 in the outputfile; the value generated by the second <fieldname> becomes field #2 in the output file, etc. Note: If you specify the “(I” option, then the item-id is suppressed, and the value generated by the first <fieldname> becomes the new item-id.

You will be prompted to enter the output file. If you press <RETURN> without specifying a file name, then the input file is also used as the output file (note: this is not recommended).

BUILD_OUT2OI

The BUILD_OUT2OI command allows you to generate a “customized” output routine, to assist you in moving applications or data from other multivalue environments into OpenInsight and CTO. It converts the text document OUT2OI.TXT, which contains BASIC+ code for various different multivalue platforms, into platform-specific source code.

Type in BUILD_OUT2OI by itself to display a list of platform choices. Type in BUILD_OUT2OI <platform> (for example, “BUILD_OUT2OI UNIVERSE”), and when prompted specify the output location (path and file name) for the destination file, to create a text document with source code for the specified platform. You may then take this source code to your legacy system and generate a CTO-compatible “virtual tape” of your account.

IN2OI

The IN2OI routine reads and restores the items created by the OUT2OI routine. Note that it operates only on the “flat file” output of OUT2OI; if “account save” format is selected in OUT2OI, then you should use ADD.ACCOUNTS or ACCOUNT.RESTORE to restore those items.

Unlike ADD.ACCOUNTS and ACCOUNT.RESTORE, you must first manually create the application in which you wish to restore these items. Then, while logged in to that application, you may run IN2OI. You will be prompted for the path to the OUT2OI files, and whether existing files should first be cleared before processing. If the existing files are not cleared before processing, you will be prompted whether existing items should be overwritten. Finally, you will be prompted to enter the path where new tables should be created. The IN2OI routine will examine the “control” record created by OUT2 OI, and use this information to process each file generated by OUT2OI.

STARTSPOOLER

The STARTSPOOLER command takes one of two parameters, "C" or "I". When invoked with the C parameter, it deletes all printer definitions, and restarts printer#0 as the default printer. When invoked with the I parameter, it both deletes all printer definitions and resets the printer queue (including the deletion of all hold entries).

STARTPTR / STARTPTRX

The STARTPTR and STARTPTRX commands allow for the definition or redefinition of printers. They have the following format:

```
STARTPTR <ptr#>,<form#>,<#skip>,<pnamee>
```

```
STARTPTRX <ptr#>,<form#>,<#skip>,<subname>,<type>,<orient>,<font>,<size>,<pname>
```

```
STARTPTR <ptr#>,<form#>,<#skip>,<subname>,<type>,<orient>,<font>,<size>,<pname> (X
```

Using the 'standard' STARTPTR command will define this printer for output using DirectPrint; all control codes, escape sequences, etc. will be passed "as is" to the printer. Using the STARTPTRX command (or STARTPTR with the "(X" option) allows you to define additional features, including a subroutine that will be invoked when a print job begins and ends, and/or the use of OIPI for output generation.

For either form of the command, ptr# is the number you wish to assign to this printer; form# is the number or numbers of the form queues this printer should service; #skip is the number of pages to eject when each print job completes; and pname is the name of the printer.

When using STARTPTRX or the X option of STARTPTR, subname is the name of a printer support subroutine to call when the print job begins, and again when the print job ends; type is either the literal string OIPI or D for DirectPrint; orient is either "0" for portrait (the default) or "1" for landscape; font is the name of the font to use; and size is the font size to use. Please note that orient, font, and size are only relevant if type is OIPI, and pname is ignored (but still must be specified) if type is OIPI.

For OIPI output to be generated, you MUST be running your REVCMD session using either CTO_CHILD_FORM or CTO_STANDALONE_FORM; if you are using the 'regular' REVCMD interface, no output will be generated.

To use the printer 'support subroutine', you must have a subroutine that has three parameters. When invoked by the printer subsystem, the first parameter will be set to the form number, the second parameter will be "1" if this is a hold file entry (or "0" if this is regular printer output), and the third parameter will be "0" if this is the 'print job start' call, or "1" if this is the "print job end" call.

To specify multiple form queues, surround the form queue numbers in parenthesis. For example, to start printer #0 with form queues 0, 1, and 2, with 0 page skip, and have the output go to a printer named LASER, you would issue the command:

```
STARTPTR 0,(0,1,2),0,LASER
```

Note that by default, printer #0 will automatically be started when the spooler is initialized.

SP.EDIT

The SP.EDIT command allows you to display, spool, and delete previously generated "hold entries" (i.e., print files generated while your SP.ASSIGN state included "H"). It may be invoked with the following syntax:

```
SP.EDIT {<accountname>} {<n>{-<n>}} {D} {F<n>{-<n>}} {L} {MD} {MS} {P} {R} {U}
```

where the options are:
<accountname> - the name of the application (in quotes) that generated the hold file(s) you wish to examine;
<n>{-<n>} - either the entry number, or a range of entry numbers, to examine;
D - the desired action is to DELETE the hold entries;
F<n>{-<n>} - either the form number, or a range of form numbers, associated with the hold file(s) you wish to examine;
L - the desired action is to LOOK at the hold entries;
MD - Delete the selected hold entries;
MS - Spool the selected hold entries;
P - the desired action is to PRINT the hold entries;
R - change the assignment information for the selected entries to the current SP.ASSIGN state;
U - allow access to all entries (not just those from the current application)

Once the desired entries have been selected, you will be prompted, for each entry, to select your desired action - to display the first 500 bytes of the record, to spool it to the printer, or to delete it. You may also choose to exit from the SP.EDIT process by entering "X", or proceed to the next entry by pressing RETURN. Note that if you have specified on the command line either D, L, or S, then you will be presented only with the specified option - Delete, Look (display first 500 bytes), or Spool the entry - otherwise, you will be prompted in turn for each of the available actions.

Specifying either MD or MS will skip the prompt, and execute the desired behavior without verification.
If you choose to Spool the entry, you will be given the choice of spooling to the printer, to the screen (by selecting T for "Terminal"), to the screen without pagination (by selecting "TN"), or to a file (by selecting "F"). If you spool the entry to a file, you will be further prompted for the file and item name.

SP.OPEN

The SP.OPEN command will instruct the spooler to keep the next, and subsequent, print jobs "open". The subsequent print jobs will be concatenated together into the single print file. To close the print job, issue the SP.CLOSE or SP.EDIT command.

SP.CLOSE

The SP.CLOSE command will close the open print jobs. The newly-closed job will spool to the printer at this time, if print output was selected.

LISTPTR

`The LISTPTR command will list all the currently-defined printers.`

LISTPEQS

The LISTPEQS command will list any print jobs currently in the queue. The listing will display the entry number, the user and station names of the creator, the date and time the job was created, the output form queue number, whether this is a hold file, and the status (O=open and in-process, C=closed)

SP.KILL

The SP.KILL command will delete a specific printer. It has the following syntax:

```
SP.KILL D<#>
```

Where <#> is the printer number you wish to delete.

SP.ASSIGN

The SP.ASSIGN command instructs the print processor in the CTO where to send output specified for the printer. You may choose to have printer output sent either to a physical printer, or to a file on your hard drive. The syntax for the SP.ASSIGN command is:

```
SP.ASSIGN {H|S|F<n>|M|O|?}
```

These options may be combined, or issued individually.

The "?" option will display the currently-assigned settings. The "H" option specifies that output to hold file is selected. The "S" option indicates that output to the physical printer should be suppressed. The "F" option, followed by a form queue number, specifies that you wish output to go to the specified form queue. The "M" message will suppress the print job entry display for subsequent print jobs. The "O" option will keep the print job open (please see SP.OPEN for more details).

When issued without any options, SP.ASSIGN will set the currently-selected options to "F0" (printed output to form queue 0).

Examples of SP.ASSIGN commands include:

SP.ASSIGN F3?
Set the currently-selected options to form queue 3, and display the current settings

SP.ASSIGN HS
Set the currently-selected options to hold file, without any physical print output

TERM

The TERM command displays or modifies your current settings for printer and "terminal" width and depth. When invoked with no parameters, the TERM command will display your current terminal and printer width and depth. You may also specify additional parameters; the format of the TERM command is:

```
TERM <termwidth>,<termdepth>,<ignored>,<ignored>,<ignored>,<ignored>,<printerwidth>,<printerdepth>
```

You may skip any parameter you do not wish to change, but you must include the comma delimiter between fields. That is, to specify a printer width of 132 and depth of 60, but to leave the rest of the settings untouched, you could issue the command:

```
TERM ,,,,,,132,60
```

BASIC/COMPILE

The syntax for the BASIC command (and its synonym COMPILE) is:

```
BASIC <filename> {<itemname> {<itemname> <itemname> ... } | *}
```

The BASIC command will compile the specified item or items, found in the specified file. Note that if the specified file is anything other than SYSPROCS, the stored procedure will actually be compiled with the name <filename>_<itemname>. After compilation, an entry in the SYSLISTS table (if the filename is SYSPROCS), or in the DICT of the specified filename, will be created named "*"<itemname>. This item will contain a copy of the output from the BASIC command.

Also note that, although you may RUN your compiled Basic program from any application (through the use of q-pointers), you must compile your routines in the application where the Basic source file is located; an attempt to compile a routine in a different application will generate an error message.

If the filename is anything other than SYSPROCS, then you must use the RUN command to invoke this compiled procedure, or you must CATALOG the procedure if you wish to invoke it directly by its name. If the procedure has been cataloged, you do not need to re-catalog the procedure if you recompile it with the BASIC command again, unless it has been marked as a "GLOBAL" catalog pointer (see the CATALOG command for more information). Globally-cataloged routines require that you re-catalog the procedure if you re-compile it with the BASIC command; if you do not recatalog the routine, when you invoke it by its name only you will run the previously cataloged object code.

During compilation, the CTO compiler converts any “foreign” data/basic syntax into OpenInsight-compatible BASIC+. This often means that the source code that is actually compiled differs from the original users’ source code. The source code that is actually compiled is stored in the SYSLISTS table, with the item name (prefixed with the filename and underscore, unless the source table name is SYSPROCS) as the item id. You may refer to this “intermediate” source code item if you wish, to see how the CTO precompiler altered the code. Alternatively, if you do not wish the CTO compiler to record this intermediate source code, you may create an item named SAVE.CONVERTED in the MD file. Put an “X” in field 1, and a “0” in field 2, to override the CTO compiler’s default behavior. If any errors are encountered during the compilation process, the line number in the original source is displayed, and the line number in the “intermediate” source is displayed (in parenthesis) as well. Any errors that are encountered during the compilation process are both displayed on-screen and recorded in the “*<itemname>” record.

The CTO compiler may occasionally generate spurious errors, most likely due to uncertainty regarding the use of Pick-style format codes. While every attempt is made to determine the proper intent, the use of format variables (as opposed to explicit format codes) may be rejected by the precompiler. You can override this behaviour by editing the item FORMATVARS in the MD. This item should have an “X” in field 1, and then, separately on each line, the variable name that you use as a format variable in your source code. For example, if your source code had the following lines:

```
FMTVAR = “R#3”
PRINT OUTVAR FMTVAR
FMTVAR2 = “L#10”
PRINT OTHERVAR FMTVAR2
```

you could update the FORMATVARS entry in the MD so that it contained the following:

```
X
FMTVAR
FMTVAR2
```

CATALOG

The syntax for the CATALOG command is:

```
CATALOG <filename> {<itemname> {<itemname> <itemname> ... } | *} {(G}
```

The CATALOG command will record a copy of the compiled code for the specified item<s>. Unlike the BASIC command, the compiled code will be named <itemname> (not <filename>_<itemname>), therefore allowing for the direct invocation of the procedure by its name alone.

The "G" option indicates that this is a GLOBALLY cataloged entry. If this is global catalog entry, you MUST reissue the CATALOG command after you change the source to the basic+ routine; it is not sufficient to just reissue the BASIC command.   Invoking the procedure by name will invoke the last CATALOGED version of the routine.

If you do not specify the "G" option, the system will create a LOCAL catalog entry. Local catalog entries do not need to be recataloged when you recompile the source code; the changes to the source code are automatically reflected when you run the procedure by its name alone.

Note also that you MUST catalog any procedures that are used as subroutines in other procedures.

DECATALOG

The syntax for the DECATALOG command is:

```
DECATALOG <filename> {<itemname> {<itemname> <itemname> ... } | *}
```

The DECATALOG command will remove the compiled code for the specified item<s>. After the DECATALOG command has been issued, you cannot invoke the specified procedure, either directly or through the RUN command. You must reissue the BASIC command to recompile the procedure if you wish to be able to invoke it through the RUN command.

CONVERT

The CONVERT command syntax is similar to the COPY command:

```
CONVERT {DICT} <filename> {<itemname> {<itemname> <itemname> ...} | *} {({O|D}}
```

The CONVERT routine will convert the specified item or items, or all the items in the file, from "traditional" multivalue format to OpenInsight format. If DICT is specified, multivalue dictionary items will be converted to OpenInsight dictionaries; if DICT is not specified, then multivalue basic programs will be converted to BASIC+.

When you enter this command, you will be prompted for a destination. If you press <return>, you will be asked to verify that you wish to overwrite the current items with the newly converted information. You may also specify a destination filename, by prefixing it with "(" - for example:

```
CONVERT BP *
To: (NEWBP
```

You may also specify new item names, in a one-to-one match with the source item names - for example:

```
CONVERT BP PROG1 PROG2 PROG3
To: NEWPROG1 NEWPROG2 NEWPROG3
```

By default, new items cannot overwrite existing items; if you wish to overwrite existing items, specify the "O" option. If you wish the original items to be deleted after conversion, specify the "D" option.

SORT and LIST

The LIST processor allows you to generate output based on sort and selection criteria. Please refer to your OpenInsight guides for the syntax of the LIST command.

The SORT command is identical to the LIST command. Additionally, the SORT command will sort the selected data by item name if no other sort criteria is specified.

SSELECT and SELECT

The SELECT processor allows you to generate a 'select list' based on selection criteria. Please refer to your OpenInsight guides for the syntax of the SELECT command.

The SSELECT command is identical to the SELECT command. Additionally, the SSELECT command will sort the selected data by item name if no other sort criteria is specified.

The select list generated by SELECT or SSELECT is only valid until the next executed command. If you wish to "preserve" the select list for future use, please see the SAVE.LIST command.

When your select list is “active”, the command prompt will change from “>” to “>>”. As long as “>>” is your command prompt, you have an active select list. This list will remain active until it is “consumed” by a process that can use an active select list (such as ED, LIST, etc.) or until the CLEARSELECT command is issued.

QSELECT

The QSELECT command has the following syntax:

```
QSELECT {DICT} <filename> <itemid> {<itemid><itemid>…|*} {(<fieldno>}
```

This command creates a select list from the specified filename and itemname(s) entered. Each attribute and value within the item(s) is used to build the select list. You may specify one or more items, or an “*” for all items in the file.

If you specify the optional <fieldno>, then the specified field, and any contained values, is used to build the select list.

COUNT

The COUNT command will count the number of items in the specified table, after applying any optional selection criteria. The syntax for the COUNT command is:

```
COUNT {DICT} <filename> {<selection criteria>}
```

For example, to count all the items in file EMPLOYEES, you may issue the following command:

```
COUNT EMPLOYEES
```

To count all the employees in group "A", you may issue the following command:

```
COUNT EMPLOYEES WITH GROUP = "A"
```

SUM

The SUM command has the following syntax:

```
SUM {DICT} <filename> <fieldname> {<fieldname>}
```

It will display the sum of each of the specified columns.

SAVE.LIST

The SAVE.LIST command allows you to preserve an active select list. It has the following format:

```
SAVE.LIST <listname>
```

Note that saving the list also "consumes" the list, so it is no longer available for further processing. When you wish to reactivate the list, you must use the GET.LIST command.

CLEARSELECT

The CLEARSELECT command 'consumes' the currently active select list, so it is no longer available for processing.

GET.LIST

The GET.LIST command retrieves a saved select list and "re-activates" it. After using GET.LIST, the list is available for further processing. The syntax of the GET.LIST command is:

```
GET.LIST <listname>
```

DELETE.LIST

The DELETE.LIST command deletes a previously-saved list. After using DELETE.LIST, you can no longer retrieve the saved list. The syntax of the DELETE.LIST command is:

```
DELETE.LIST <listname>
```

RUN

The syntax for the RUN command is:

```
RUN <filename> <procedurename>
```

The RUN command will invoke the specified procedure, as found in the specified file.

ED

The ED line editor allows you to edit or create one or more items in a file. It is invoked with the following syntax:

```
ED {DICT} <filename> {<itemid> {<itemid> <itemid> ...} | *} {({T}{N}}
```

Alternatively, if there is an active "select list" when ED is invoked, the list of items to operate on will be drawn from the select list.

The "T" option instructs the editor to convert any tabs in the item to a space. The "N" option forces the editor to display dictionary items in OpenInsight format, even if they were originally entered in "traditional" format.

If a specified item does not already exist, the "New Item" message will be displayed. In both the "new item" and existing item cases, you will then be placed at the "top" of the item (before field 1), and the editor prompt "." will be displayed.

From the editor prompt, you may move through the item in a variety of ways. The editor tracks your current location; editing and inserting operations (for example, changing some characters in a field) occur on the current line. You may change your location in the item with the following commands:

T - return to the Top (field 1);
B - move to the bottom (the last field of the record);
{G}<n> - move to field number <n>;
U{<n>} - move up <n> lines (<n> defaults to 1 if not specified);
+<n> - move down <n> lines;
-<n> - move up <n> lines;
<cr> - move down one line;
L{<n>} - display and move down <n> lines (<n> defaults to 21 if not specified);
P - display and move down 21 lines;

You may also change your location in the item by searching through the item. The search commands include:
L:<text> - Move to the line that begins with <text>;

F<space><text> - Move to the line that begins with <text>;
L{<n>}<delim><text> - Move to the line that contains <text>. If <n> is specified, search through the next <n> lines for the specified <text>, and display the occurrences (current location moves to the end of the search region);
A - Repeat last search again

The following commands allow you to modify the item starting at the current line:
D{E}{<n>} - Delete the current line, or the current and next <n> lines;
R{U}{<n>}<delim>{<oldtext>}<delim>{<newtext>}{<delim>} - Replace the first occurrence, or all occurrences (if U specified), on the current line or the current and next <n> lines, of the specified <oldtext> with <newtext>. The <delim> character may be any character (other than "U" or a number) that does not occur in the <oldtext> or <newtext>. Leaving out <oldtext> will insert <newtext> at the beginning of the line or lines. Leaving out <newtext> will delete <oldtext> in the line or lines;

R<space><text> - Replaces the current line with the specified <text>;
R - Prompts for the entry of the text to replace the current line;
I<space><text> - Insert the specified <text> after the current line;
I - Prompts for the entry of the text to insert after the current line. The editor remains in "insert" mode until a <cr> is entered on a line by itself;

Please note that if you enter just a single space as the text for a line when in “insert” mode, the editor will convert this into an empty line. This is the only way to generate an empty line in insert mode; if you just press <cr>, you will exit insert mode. If you actually wish to create a line with a single space, you can either use the Replace function after you have finished with Insert mode, or you can enter the space using the control code syntax (^032) discussed below.
Whenever you are modifying, searching or inserting text in a record, you may specify "control characters" using the special syntax "^xxx", where xxx is the 3 digit decimal representation of the character. For example, "^254" would represent a field mark, "^253" would represent a value mark, etc. You may use this special representation when searching, inserting, or eplacing text.
By default, searching and replacing operates in a "case sensitive" mode. You may toggle this behavior by entering the "C" command. When in "case insensitive" mode, searches and replaces will match the specified text regardless of its case.

You may also modify the item using "block" commands. The block commands allow you to specify a region of the item that you wish to effect. You define the block by moving to the first line of the block and entering "<". You then move to the last line of the block and enter ">". If the block only contains one line (the first and last line of the block are the same), you may enter "<>". The following commands operate on these blocks:
DROP - delete the lines within the block;

MOVE - copy the lines from within the block to the current location, and then delete the original block;
COPY - copy the lines from within the block to the current location. The original block is unchanged;

It is sometimes necessary or desirable to combine text from another item with the current item. The following command allows you to extract text from another item and insert it at the current location:
ME{<n>}<delim>{<location>}<delim>{<startingline>} - Merges 1 line, or <n> lines, from the item specified by <location>. <Location> may be just an item name (for an item in the current file), or the filename and itemname, separated by a space. If <location> is omitted, the text is merged from the current item. The text is extracted starting from line <startingline>, or 1 if <startingline> is omitted;

At any time, you may choose to "undo" the last change. You may enter the command "OOPS" or “X” to return the item to the state it was in before your most recent change. You may issue the "OOPS" or “X” command repeatedly to undo successively earlier changes.

It is sometimes convenient, when in the editor, to execute a command (for example, to perform a copy, or edit up an unrelated item, etc.). You may execute any CTO command from within the editor by using the “>” or “$” when at the editor prompt. For example, you may type in “$TIME” or “>TIME” from within the editor to see the current date and time.

When you have finished modifying your item, you may choose to save your changes, save your changes and exit, or discard your changes. The following commands allow you to either save or discard your changes:
FI - Save the changes and exit this item. If there were other items specified when ED was invoked, the next item in the list will be presented for editing;
FIB - Save the changes, exit this item, and compile it. If there were other items specified when ED was invoked, the next item in the list will be presented for editing;
FIBC - Save the changes, exit this item, compile and catalog it. If there were other items specified when ED was invoked, the next item in the list will be presented for editing;
FIC – Save the changes, exit this item, and convert it to an OpenInsight-format dictionary item. Note that this is only applicable if you are editing a new item in the dictionary of a file;

EX - Exit without saving changes. If you have made any changes to the record, you will be prompted to confirm this action. If there were other items specified when ED was invoked, the next item in the list will be presented for editing;
FD{Y} - Delete the record. If Y is not specified in the command, you will be prompted to confirm this action. If there were other items specified when ED was invoked, the next item in the list will be presented for editing;

If you wish to save or discard your changes, and exit the editor, discarding any further items specified when ED was invoked, you may add a "K" to the end of the command (i.e., FIK, EXK, and FDK).

If you wish to save the item you are currently working on, but do not wish to exit the item at this time, you may use the following commands:

FS{O}{<itemname> | (<filename><space><itemname>} - Saves the current item without exiting. If <itemname>, or <filename> and <itemname>, is specified, then the current item is written out to the specified new location. If an item with the new name already exists, the save fails unless the "O" option is also specified;

SAVE {{DICT} <filename>} <itemname> - Saves the current item to the specified itemname, in the current file (or in the file <filename> if specified), without exiting. Note that this command always overwrites any existing item with the same name;

Normally, as you proceed through the item, the field number associated with each line is displayed. You may enter "S" at the editor prompt to "toggle" the line number display on and off. Similarly, you may wish to toggle on and off the display of control characters (these are characters that are not normally printable). By default, control characters will not be displayed in a special, person-readable format; however, the "^" command will toggle the control character display mode.

If you wish to display information about the current record, you may enter either of the following commands:
? - displays the current file name, item name, and line number. If this item is part of an active list of items, this also displays the current item count and the total number of items;
S? - displays the size of the current item

Finally, the editor has the ability to save and reissue a sequence of commands. These sequences of commands are saved as a "prestore", and are defined with the following syntax:
P<#><space><cmd>{]<cmd>]<cmd>...}

where <#> is the prestore number you wish to save this as, and <cmd> is the editor command you wish to invoke when this prestore is recalled. Multiple editor commands may be specified in a single prestore by separating them with the ESC character (represented by] above). To invoke a prestore, enter the following command:
P<#>
where <#> is the prestore number previously used. To delete a previously defined prestore, issue the command:
P<#><space>

EDIT.LIST

The EDIT.LIST command allows you to edit an existing, or create a new, saved list. Note that the list must be saved with the SAVE.LIST command (that is, EDIT.LIST will not work on the currently-active select list). Also note that lists saved with the SAVE.LIST command will typically contain "control" information in field 1, which should not be altered; lists that you create yourself need not have this information on field 1.

LISTE/LISTU

The LISTE (and its synonym, LISTU) command will display the engines currently in use by the OpenEngine Server. The current engine used by the CTO will be designated by an asterisk ("*").

WHO

The WHO command will return information about the engine currently used by the CTO.

VERSION

The VERSION command will return the current version number of the CTO.

RESET.ENGINE / LOGOFF

The RESET.ENGINE and LOGOFF commands will allow administrative users to reset any engine that may be performing improperly or that you otherwise wish to shut down. Note that these commands can only be run by users with administrator privileges.

By default, the RESET.ENGINE command will operate only on "asynchronous" engines (i.e., those that are being used for web interaction). To reset other CTO sessions, you must specify the "U" option when invoking the command ("RESET.ENGINE (U"). You will be prompted for the name of the session to reset; press RETURN for all engines, or the name of the application you wish to reset. You must also provide the supervisor password as defined for that OpenEngine Server (by default, this is REVSOFT).

The LOGOFF command will prompt the user for the name of the process to log off, and the OpenEngine Server supervisor password. It will terminate the CTO or asynchronous sessions that have the specified process name.

LOGTO / LOGTOX

The LOGTO and LOGTOX commands will allow you, from TCL, to leave the current application and enter a different application. You may either specify LOGTO by itself, or use the following syntax:

LOGTO <application>{,<username>,<password>}

You will be prompted for any parameters you do not supply on the command line. Note that if you specify an application that doesn't exist, or you incorrectly enter your username or password, you will be in a "logged off" state, and must actually log off and reconnect to resume.

If the application, username, and password are the same, you may use the LOGTOX command. LOGTOX will automatically use the specified application name to fill in the username and password parameters. Use the following syntax:

LOGTOX<application>

For example, entering the command

LOGTO MYAPP,MYAPP,MYAPP

LOGON

The LOGON command allows you to create a 'background' or 'phantom' process. You may either specify LOGON by itself, or use the following syntax:

LOGON <processed>{,<application>{,<username>{,<password>}}}

You will be prompted for any parameters you do not supply on the command line.
You must specify a "processID" so that you might uniquely identify the phantom process, and the name of the application and user you wish to invoke. There MUST be a "login process" associated with the application or user. The "phantom" process will run the login process, and any other procedures the login process may call. Once the login process has terminated, the 'phantom' process will be reset. Note that the phantom process CANNOT request any
input (unless that request is satisfied by DATA statements); any INPUT statement will cause the phantom process to terminate.

OFF

The OFF command indicates to the CTO that you are finished with the current application. You will be returned to the "login" screen of the revcmd program.

QUIT

The QUIT command indicates to the CTO that you are finished with the current application and the CTO environment in general. The revcmd program will terminate.

LISTDICT

The LISTDICT command has the following syntax:

LISTDICT <filename>

The full sorted list of dictionaries for the specified file will be displayed.

LISTFILES

The LISTFILES command displays the full sorted list of tables that are available to you in the current application.

ECHO

The ECHO command turns on and off character echoing (i.e., the characters that are typed are either displayed, or not displayed, as you type them). You may issue the command:

ECHO OFF
to turn off character echoing, or

ECHO ON
to turn on character echoing (the default)

If you issue the ECHO command without any parameters, the current state of echoing will be toggled.

BREAK

The BREAK command enables or disables the function of the "break key" (set to ^C by default). You may issue the command:

BREAK OFF

or

BREAK 0
to disable the break key, or

BREAK ON

or

BREAK 1

to enable the break key (the default)

P

The P command toggles display output off and on. When in "P" mode, all output generated by the CTO is discarded.

TIME

The TIME command displays the current time and date. You may also optionally enter a time, and the TIME command will display the internal time value associated with your entry. You may also optionally enter an internal time value, and TIME will display the "user readable" time associated with your entry.

```
TIME
15:35:42 21 OCT 2005

TIME 32000
08:53:20AM
 TIME 10:30PM
81000
```

DATE

The DATE command displays the current date. You may also optionally enter a date, and the DATE command will display the internal date value associated with your entry. You may also optionally enter an internal date value, and DATE will display the "user readable" date associated with your entry.

```
DATE
21 OCT 2005

DATE 9872
10 JAN 1995

DATE 4/22/1963
-1714
```

CTO_SLEEP

The CTO_SLEEP command will put a process to sleep until the time specified.

From a BASIC program:
```
CALL CTO_SLEEP(“11:45:30PM”)
```

From TCL:
```
CTO_SLEEP 11:45:30PM
```

FIND

The FIND function allows you to find all items in a specified table that contain the specified text. The syntax of this command is:

```
  FIND {DICT} <filename> <text>
```

If any items in the specified file contain the specified text, an active select list of those items will be returned. Note that the text may optionally be enclosed in quotes.

SET.OPTIONS

The SET.OPTIONS command allows you to set or display account options. It has the following syntax:

```
  SET.OPTIONS {<optionname>|<optionnumber>,{<value>|?}}
```

When issued without parameters, SET.OPTIONS will display a full list of options available, their possible values, and their current values. Each option will have an option number preceding the option name, followed by the list of possible values. The default value for the option will be indicated by parenthesis.

For example, entering the command:

```
SET.OPTIONS
```

Will produce a display similar to:

```
1.    FLAVOR [AREV32|D3|OI|(R83)|UNIVERSE]=R83
2.    SAVE_STACKED_COMMANDS [(0)|1]=0
3.    EXECUTE_ALL_STACKED_COMMANDS [0|(1)]=1
```

To examine a single options value, you may specify either the option name or option number followed by ",?". For example, to examine the current account flavor, you may enter the command:

```
SET.OPTIONS FLAVOR,?
```

Or

```
SET.OPTIONS 1,?
```

In our example, this would return:

```
R83
```

To set an option value, you may specify either the option name or option number, followed by a comma and an acceptable option value. For example, to set the current account flavor to AREV32, you may enter the command:

```
SET.OPTIONS FLAVOR,AREV32
```

Or

```
SET.OPTIONS 1,AREV32
```

Option changes remain in effect only throughout the current login session.

It is also possible to query or set options while in Basic+ procedures. Please see SET.OPTIONS in the BASIC+ SUPPORT ROUTINES section.

Current options include:

1. FLAVOR: Modifies compiler and I/O settings to comply with the specified type of mutivalue system;
2. SAVE_STACKED_COMMANS: Indicates whether commands executes in PROCs are recorded in the TCL stack;
3. EXECUTE_ALL_STACKED_COMMANDS: Indicates whether any pending commands in the PROC buffer are executed when the PROC terminates without an active select list

U2 Compatibility

The following functions have been implemented in order to support UniVerse and UniData users who are moving to OpenInsight and CTO:

```
FUNCTION CONVERT(OLDVAL, NEWVAL, SOURCE)
FUNCTION LOWER(SOURCE)
FUNCTION RAISE(SOURCE)
FUNCTION SUBR(ROUTINENAME {, PARAM1 {, PARAM2 {,…}}})
FUNCTION TRANS(FILENAME, KEY, FIELD, TYPE)
```

Debugger Commands

Command	Description of the Command
-----------------	---
#	Displays stats about memory usage
$ and ?	Displays the current line and program name
/{var}	Display var in ASCII (/) format - \ alone displays popup
\{var}	Display var in HEX (\) format - / alone displays popup
B$=n	Breaks if line number = n
B@=program	Breaks if program name = program
Bx operator const	Breaks when expression x operator constant is true
D	Displays break and trace tables
E{n}	Sets multi-step line counter to n
En!	Sets multi-step line counter to n, without stepping into any external subroutine calls
END or press ESC	Aborts program and returns to the command window
EXECUTE command	Executes command as though at the command window
G	Go, resumes program control
Gn	Goes to line n in source code
HM	Extended help screen (AREV32 only)
K{n}	Clears the break table, n = clear only specified nbr
Ln	Lists n lines from current source file
Ln-m	Lists lines n through m from current source file
O{FF}	Returns to operating system
PS	Displays the program stack
PSP	Displays the program stack in a popup (AREV32 only)
S	Displays screen image that preceded drop to debugger (AREV32 only)
SO{URCE}	Specifies filename and program of source code
ST	Toggles the source trace mode off or on
T	Toggle the trace mode off or on
T$	Traces the source of a program line
T@	Traces a program source
Tvariable	Traces the specified variable
U{n}	Clears the trace table, n = clear only specified nbr
V{var}	Displays var in window - V alone displays popup of vars (AREV32 only)
X	Displays the command window (AREV32 only)

Setting the debugger

FUNCTION RTI_SET_DEBUGGER(MODE {, REPLACEMENT_ROUTINE_NAME})

Call RTI_SET_DEBUGGER to dynamically change the debug mode, and optionally change the name of the debugger replacement routine that is called in “intercept” mode. Set mode to “0” to disable the debugger, “2” to enable the alternate debugger, or “1” (or anything else) to reset to the normal OI debugger. When mode is “2”, you may optionally specify the name of a stored procedure to use as the debugger replacement routine. If no routine is specified, then the debugger replacement routine is unchanged; if the debugger replacement routine is invalid, then the default routine DEBUGGER_REPLACEMENT will be used. Changes to the debug mode will last until the next RTI_SET_DEBUGGER call, or until the user logs out and logs back in.

The return value of the function is the previous debug mode, and the previous debugger replacement routine’s name is returned in the replacement_routine_name parameter.

Example:

Declare Function RTI_SET_DEBUGGER

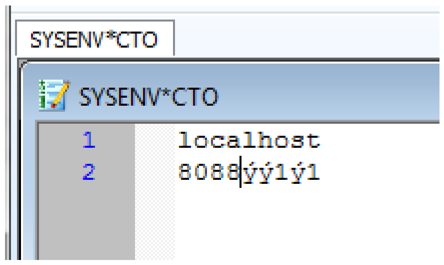
routineName = “MY_DEBUGGER_CODE”
oldMode = RTI_SET_DEBUGGER(“2”, routineName) ;* oldMode retains the previous debug mode, and routineName is changed to the prior debugger replacement routine name
* … perform actions that require debug mode 2 …
Dummy = RTI_SET_DEBUGGER(oldMode, routineName) ;* reset to prior values to clean up

When using CTO or AREV32, it is now possible to debug your processes using the graphical OpenInsight debugger (as opposed to the character-based alternate debugger that is used by default in CTO and AREV32). You must be using an EngineServer connection (as opposed to the “direct”, local connection specified with “.” in the CTO configuration record (found in SYSENV or the MD table of the application)).

Make sure you’re running the EngineServer on your desktop, in a DOS command box using the java command, and not as a service.

Edit your CTO configuration record (named CTO*<username> in the MD table of the application you’re running in, or just CTO in the MD table, or CTO in the SYSENV table), verifying that field 1 contains your local machine ID (either “127.0.0.1”, or “localhost”, or the actual IP address or name of the local machine), and field 2 contains the port that the EngineServer will be “listening” on (8088 by default) in its first value. The third and fourth values contain the startup and shutdown flags, respectively; set them both to “1”, and save your configuration record.

Your CTO record should now look like this:



```
1   localhost
2   8088ýý1ý1
```

Once you have saved your modified CTO record, you can start up CTO or AREV32. You should see a visible OpenEngine (OEngine) start on your desktop.

By default, when CTO or AREV32 starts up, the alternate debugger is automatically enabled. OpenInsight 9.2.1 and above allow you to change this dynamically with the RTI_SET_DEBUGGER command. From TCL in CTO or AREV32, issue the command:

RTI_SET_DEBUGGER “1”

Or from within a stored procedure add the following line:

CALL RTI_SET_DEBUGGER(“1”)

This will re-enable the normal OpenInsight graphical debugger. Now, when you encounter a run-time error, or a DEBUG statement, the OEngine on your desktop will launch the graphical debugger, allowing you to step through your CTO or AREV32 routine.

BASIC+ Support Routines

The following routines are supplied with the CTO to provide compatibility with "traditional" multivalue systems. Developers will not necessarily need to be familiar with these routines, as the BASIC compiler will normally convert traditional multivalue BASIC to OpenInsight's BASIC+; however, they are documented here should developers wish to work on post-converted code.

SUBROUTINE CURSOR_SELECTVAR_SWAP(<cursornumber>,<selectvar>,<direction>)
- Transfers the contents of the select list stored in the specified cursor into the specified select variable (if <direction> is null or 0), or transfers the contents from the select variable into the specified cursor (if <direction> is 1), or swaps the information in the cursor and select variable (if <direction> is 2)

FUNCTION DO_AT(<xparam>{,<yparam>})

- generate "escape sequences" for cursor positioning and screen control. You may specify a negative value for the <xparam> to generate the following actions:

Clear Screen:	-1
Position cursor at home position:	-2
Clear to end of screen:	-3
Clear to end of line:	-4
Blinking text on:	-5
Blinking text off:	-6
Protect on:	-7
Protect off:	-8
Cursor back one character:	-9
Cursor up one line:	-10
Cursor forward one character:	-11
Cursor down one line:	-12
Reverse video text on:	-13
Reverse video text off:	-14
Underline text on:	-15
Underline text off:	-16

SUBROUTINE DO_CRT(<output>{,<holdatend>)
- Displays specified <output> at the current cursor position on the CTO screen. Optionally suppresses line feed/carriage return at end of output (<holdatend> set to "1")

SUBROUTINE DO_DATA(<param>)
- Puts <param> into "data stack"; this data will be retrieved by the next call to DO_INPUT or - if the invoking program terminates - the value of the <param> will be taken to be the name of the next program to invoke.

SUBROUTINE DO_ECHO(<state>)
- Turns echoing of characters when typing on (<state> set to "1") or off (<state> set to "0")

SUBROUTINE DO_EXECUTE(<cmd>{<capture?>,<capturevar>{,<return?>,<returnvar>}})
- Invokes the specified <cmd>, either by CALL (if this is a BASIC+ routine), or by using the PROC or PAragraph interpreter, or by using RList (if this is a LIST or SELECT statement). If <capture?> is set to "1", then output from the command is stored in the <capturevar> variable, rather than displayed on-screen. If <return?> is set to "1", then any return values generated by the command will be stored in the <returnvar> variable.

SUBROUTINE DO_FOOTING(<text>{,<doforce>)
- Sets or resets the "footing" text for all output. You may specify the footing string using the same special characters that are used in a LIST statement's HEADING and FOOTING specification. To disable footing, call DO_FOOTING with <text> set to null. Calling DO_FOOTING with <doforce> set to "1" forces the use of a footing even if <text> is null.

FUNCTION DO_FORMAT(<inparam>,<formatstring>)

- Returns the text in the parameter <inparam> after formatting it according to the specified <formatstring>. DO_FORMAT supports OpenInsight, U2, and "traditional" multivalue format codes.

SUBROUTINE DO_HEADING(<text>{,<doforce>})
- Sets or resets the "footing" text for all output. You may specify the footing string using the same special characters that are used in a LIST statement's HEADING and FOOTING specification. To disable footing, call DO_FOOTING with <text> set to null. Calling DO_FOOTING with <doforce> set to "1" forces the use of a footing even if <text> is null.

SUBROUTINE DO_HEADING(<text>{,<doforce>})
- Sets or resets the "footing" text for all output. You may specify the footing string using the same special characters that are used in a LIST statement's HEADING and FOOTING specification. To disable footing, call DO_FOOTING with <text> set to null. Calling DO_FOOTING with <doforce> set to "1" forces the use of a footing even if <text> is null.

SUBROUTINE DO_HUSH
- Toggles the "output suppress" flag. When in "output suppress" mode, all output from the CTO is discarded rather than displayed.
FUNCTION DO_INPUT({<holdatend>{,<maxlen>{,<posn>{,<origvalue>{,<delay>{,<beepatmax>{,<doraw>}}}}}}})
- Returns input as entered by the user in the CTO, or as extracted from the "data stack" (if present). If <holdatend> is set to "1", then the final cr/lf typed by the user will be suppressed. If <maxlen> is specified, then no more than <maxlen> characters may be entered; if <beepatmax> is set to "1", then the CTO will beep if more than <maxlen> characters are typed. If <posn> is specified, the cursor will first be moved to the specified screen position (NOTE: <posn> should be the result of a DO_AT() call). If <origvalue> is specified, then the text in the <origvalue> parameter will be displayed at the specified <posn>. If <delay> is specified, the CTO will wait no more than <delay> seconds for a response. If <doraw> is set to "1", only a single character will be input; its ASCII code value will be returned to the calling routine.

SUBROUTINE DO_INPUTERR(<outstr>)
- Displays the specified <outstr> at the bottom of the screen. This message remains displayed until another INPUT @ is executed.

SUBROUTINE DO_PAGE({<pgno>})
- This routine ejects the current page, and – if <pgno> is specified – sets the current page to the desired <pgno>.

SUBROUTINE INPUTCLEAR
- Clears the typeahead buffer.

FUNCTION DO_OPEN(<dictname>,<dataname>,<filevar>{,<current_filename>,{real_tablename}}))
- Returns "0" if unable to open the file specified in <dictname>,<dataname>; otherwise, returns "1" and puts the "file handle" to the opened file into <filevar>. Note that DO_OPEN respects any Q-Pointers that may exist in the MD. If the specified file is a Q-pointer, its local filename may be returned in <current_filename>, and the filename in the application where it actually resides may be returned in <real_tablename>.

SUBROUTINE DO_PRINT(<output>{,<holdatend>)
- Displays specified <output> at the current cursor position on the CTO screen or to the currently selected print output device (depending on whether DO_PRINTER_ON has been invoked). Optionally suppresses line feed/carriage return at end of output (<holdatend> set to "1")

SUBROUTINE DO_PRINTER_CLOSE
- Closes any currently-generating print jobs

SUBROUTINE DO_PRINTER_OFF
- Temporarily suspends output to the currently selected print output device. Subsequent DO_PRINT statements will generate output to the CTO screen until DO_PRINTER_ON has been invoked again.

SUBROUTINE DO_PRINTER_ON
- Indicates that subsequent output from DO_PRINT should be directed to the currently selected print output device.

SUBROUTINE DO_PROMPT(<promptchar>{,<rememberflag>})
- Sets the prompt character for the DO_INPUT command to <promptchar>. If <rememberflag> is set to "0", or not specified, then the specified prompt character is also "saved" for later recall; if <rememberflag> is set to "1", then the prompt character is actually set to the "saved" value.

FUNCTION DO_READNEXT(<resultvar>{,<svalresult>,<dosval>}{,<selectvar>}})
- Returns 0 if there are no more item IDs in the current select list; otherwise, returns 1, and sets <resultvar> to the next item ID. If <dosval> is set to 1, and the select list is "by-exploding", the <svalresult> will be set to the value number. If <selectvar> is specified, the item ID is read from the specified select variable instead of cursor #0.

SUBROUTINE DO_SELECT(<filevar>, <selectvar>)
- Creates a non-sorted select list from the specified file variable, and stores the select list information in the specified select variable.

SUBROUTINE DO_SHELL(<state>)

- Enables (<state> set to "1") or disables (<state> set to "0") the availability of command-line "shell" commands for the current CTO session.

FUNCTION DO_SYSTEM(<code>)
- Returns the state of the system variables associated with the specified <code> value. Valid options for <code> are:
   1: In "printer on" mode?
   2. Page width
   3. Page length
   7. Terminal type
   8. Tape block size
   10. Items in "data stack"?
   11. Active select list?
   12. System time (in milliseconds)
   13. Sleep for 1 second
   14. Return number of characters in the typeahead buffer
   17. 'RETURN' result from previous execute

FUNCTION UCASE(<text>)
- Returns the characters contained in <text> converted to upper-case

FUNCTION LCASE(<text>)
 - Returns the characters contained in <text> converted to lower-case

FUNCTION READ_LOCK(<filevar>,<itemid>{,<att#>{,<noitemflag>{,<waitforlock>}}})
 - Attempts to read and lock the item specified in <itemid> from the file <filevar>. If <att#> is not specified, the entire record is returned; otherwise, only the specified field is returned. If the desired record is already locked, the function will return (if <waitforlock> is set to "0") or retry the lock each second, beeping to indicate it is currently locked (if <waitforlock> is set to "1", the default). If the desired record is not on file, the function will return a null string and keep the lock set (if <noitemflag> is set to "0", the default) or release the lock (if <noitemflag> is set to "1").

SUBROUTINE RELEASE_LOCK(<filevar>, <itemid>)
 - Releases the lock held on item <itemid> in file <filevar>

SUBROUTINE WRITE_LOCK(<filevar>, <itemid>, <itemtext> {, <att#> {, <keeplock> }})
- This routine updates the item named <itemid> in file <filevar> with the text specified in <itemtext>. If <att#> is 0 or not specified, the entire item is written; if <att#> is non-zero, then only the specified attribute in the item is written. If <keeplock> is set to "0" (the default), any locks on this item are released; if <keeplock> is set to "1", the item lock remains in effect.

FUNCTION SET.OPTIONS(<option>,<optionvalue>{,<nodisplay>})
- This function allows a BASIC+ routine to examine or modify current account options. For information on the <option> (either the option name or option number) and <optionvalue> parameters, please see the section SET.OPTIONS in the main manual. As described in that section, use “?” as the <optionvalue> parameter to examine the current state of the specified option. The <nodisplay> parameter would normally be set to “1” to suppress any printed output of the option information; any values are instead returned only as the function result.

Interactions between OpenInsight and CTO

To aid in debugging your CTO routines, you may invoke them from the OpenInsight environment. Using the System Monitor (from the Tolls / Advanced OpenInsight menu), type in RUN DO_CTO “<command to run>”. All output will be displayed in the System Monitor window; any requests for input will cause a Message Box to appear.

You may “embed” a CTO screen in an OpenInsight GUI form. OpenInsight currently ships with two forms which have this functionality – CTO_STANDALONE_FORM and CTO_CHILD_FORM. The CTO_STANDALONE form is designed to allow you to launch a CTO session from the main GUI application screen; CTO_CHILD_FORM is designed to allow you to write OpenInsight “MDI” frames that include green screen functionality.

Please note: You must create a record named CTO in the SYSENV table before using the CTO_STANDALONE_FORM. At a minimum, this record must contain the IP address of the OpenInsight system (the system where the OEngineServer is running) in field 1. Failure to create this record may cause unpredictable behaviour.

If you either the CTO_STANDALONE_FORM or the CTO_CHILD_FORM, you may also take advantage of some additional CTO/OpenInsight connectivity routines. These currently include:

FUNCTION CTO_MSG(<PARAMS>)
- This routine opens up a MSG window on your OpenInsight form. The <PARAMS> parameter is identical to the <msgstructure> parameter in the OpenInsight MSG fuction. This function returns the same values as the MSG function.

FUNCTION CTO_POPUP(<DEFN>,<POPUPNAME>)
- This routine opens up a PopUp window on your OpenInsight form. The <DEFN> parameter is identical to the <typeoverride> parameter in the OpenInsight POPUP function. This function returns the same values as the POPUP function.

FUNCTION CTO_RUNWIN(<CMD>,<MODE>)
- This routine executes the specified <CMD> in the underlying Operating System environment. Both <CMD> and <MODE> are passed to the UTILITY function, with the RUNWIN argument. Please see RUNWIN in the OpenInsight help for more details on the <MODE> parameter. This function currently returns 0.
FUNCTION CTO_SET_PRINTER(<CTYPE>,<PARAM1>,<PARAM2>,…<PARAM8>)
- This routine calls OpenInsight’s SET_PRINTER function, passing it any parameters that are specified. This function returns the same values as the SET_PRINTER routine.

FUNCTION CTO_GUI_FUNCTION(<FNAME>,<PARAM1>,<PARAM2>,…<PARAM20>)
- This routine calls any OpenInsight function that requires an “event context”. Specify the name of the function as the <FNAME> parameter, and any required or desired parameters as <PARAM1>, <PARAM2>, etc. This function will return the return value of the specified <FNAME> function.

FUNCTION CTO_SET_PROPERTY(OBJNAME,PROPNAME,PROPVAL)

- This routine can be used to set a property in the GUI environment. Specify the name of the object in the OBJNAME parameter, the name of the property in the PROPNAME parameter, and the value you wish to set in the PROPVAL parameter. Please note that the CTO routines, run in the character context, do NOT have access to the @window variable, and therefore you cannot specify @window (or anything derived from @window) explicily as the object name. However, if you ise the literal "@" character, CTO_SET_PROPERTY will substitute the current @window value for the "@" literal when it invokes the actual SET_PROPERTY function in the GUI context.

FUNCTION CTO_GET_PROPERTY(OBJNAME,PROPNAME)
- This routine can be used to get a property in the GUI environment. Specify the name of the object in the OBJNAME parameter, and the name of the property to retrieve in the PROPNAME parameter. Please note that the CTO routines, run in the character context, do NOT have access to the @window variable, and therefore you cannot specify @window (or anything derived from @window) explicitly as the object name. However, if you use the literal "@" character, CTO_GET_PROPERTY will substitute the current @window value for the "@" literal when it invokes the actual GET_PROPERTY function in the GUI context.

SUBROUTINE SET_OIPI_PROPERTIES(FileName, Title, PageInfo, PageSetup, PrintSetup)
- This routine allows you to customize the OIPI output generated through the CTO print routines. The parameters correspond to the first five parameters of the INIT Set_Printer message. Please refer to your OpenInsight help for the INIT Set_Printer Message for details.

SUBROUTINE CTO_GUI_SUBROUTINE(<SNAME>,<PARAM1>,<PARAM2>,…<PARAM20>)
- This routine calls any OpenInsight subroutine that requires an "event context". Specify the name of the subroutine as the <SNAME> parameter, and any required or desired parameters as <PARAM1>, <PARAM2>, etc. This subroutine calls SET_STATUS to set the status code with the status of the <SNAME> subroutine.

As an alternative to using CTO_CHILD_FORM, you may include in your form the following call:

FUNCTION RUN_CHARACTER(<ctlname>,<cmd>)

The RUN_CHARACTER function will set the properties on the specified control to allow for a connection to the sample application as the GUI form is currently running in, and will invoke the specified command. If <cmd> is not specified, the user will be placed at TCL; if <cmd> is specified, the CTO session will terminate when the command terminates.

You should also be sure to have the "Close" property of your window programmatically send the "disconnect" message to the OLE control to avoid having "orphaned" CTO sessions.

Technical Configuration for CTO

The CTO (Character to OpenInsight) environment is designed to emulate a traditional multivalue ‘green screen’ environment. It is one of several user interfaces that interact with the OpenInsight OEngine Server.

Requests from various user interfaces are sent, via tcp/ip, to the central “OEngine Server” (which must be run in the same directory as OpenInsight). The OEngine Server is responsible both for communicating with the user interfaces, and for creating, communicating with, and shutting down OpenInsight’s OpenEngine. The OEngine Server also manages “caching” of OpenEngines, for efficient performance in non-persistent applications (ie, web requests).

The OEngineServer can be run either as a service, or through a command-line interface. Note that the OEngineServer, as a java application, requires that the latest java runtime be installed on the same machine as OpenInsight.

To run the OEngineServer on the command line, issue the command “java –jar oesocketserver.jar” in the OpenInsight directory. Note that you may need to more fully describe the path to the java executable on some systems. If desired, you may add an optional parameter (“-d 1”) to display some debugging output. To terminate the command-line version, type ^C to stop the OEngineServer.

To run the OEngineServer as a service, you must run the command InstallApp.bat in the OEServer subdirectory of your OpenInsight directory. Note that this *_installs_* the service, but does not initially start it; use the Services applet in Windows to actually start the service, or reboot your system to have the OEngineServer service automatically start.

The OEngineServer reads configuration information from the file “eserver.cfg”, also located in your OpenInsight directory. You may modify its default behaviour by changing the settings in this file. The default values in this file are initially set as follows:

```
// configuration file for oesocketserver.jar
// invocation:
// java - jar oesocketserver.jar {-l <configfilename>} {-d {<debuglevel}}
//
// which port to listen on
PortNumber=8088
// maximum number of sockets connected at any time (0=unlimited)
MaxConnections=0
// maximum number of engines to keep in queue (for stateless connections ONLY)
MaxEngines=10
// time (in minutes) to wait before killing idle engines from queue
IdleTimeout=15
// default operating mode:
// "" - JD3 mode
// 0  - block mode
// 1  - character mode
// 2  - 'stateless' block mode
Mode=
// Encoding option (defaults to 8859_1)
// Encoding=UTF8
// Control Password option (defaults to WINWIN)
// ControlPassword=WINWIN
// KeepAlive option (defaults to 0) - determines whether dropped, active connections
// are kept alive as "zombies" (allowing for reconnecting) or terminated
// KeepAlive=1

// Default values: may be set on per-mode, or overall, basis
// defaults for mode "" (JD3 mode)
Procedure_=JD3_LISTENER

// defaults for mode "0" (block mode)

// defaults for mode "1" (character mode)
Procedure_1=REVCMD_LISTENER

// defaults for mode "2" ('stateless' block mode)

// overall defaults

// default name of application
```

```
Application=
// default 'dispatch procedure' to run
Procedure=
// default user name
UserName=
// default password
Password=
// default engine name
Server=
// default startup flags for engine
StartupFlags=65
// default shutdown flags for engine
ShutdownFlags=1
```

All values are described with comments, and should be self-explanatory. Note that changing any values in the eserver.cfg file will require that you restart the OEngineServer (either by stopping and restarting the command line, or stopping and restarting the service, as appropriate).

A telnet server has also been created to allow remote users to access the OEngineServer functionality. Please note that currently the OETelnetServer only supports the following terminal types: VT100, ansi, xterm

The OETelnetServer may also be invoked via a command line, or started as a service. To run on the command line, specify the command "java –jar oetelnetserver.jar" from within the OpenInsight directory. To terminate the command-line instance of the telnet server, type ^C.

To run as a service, you must first run the program "InstallAppT.bat" in the OEServer subdirectory of your OpenInsight directory. Note that this _*installs*_ the service, but does not initially start it; use the Services applet in Windows to actually start the service, or reboot your system to have the OETelnetServer service automatically start.

The OETelnetServer reads configuration information from the file "oetelnetserver.properties", also located in your OpenInsight directory. You may modify its default behaviour by changing the settings in this file. The default values in this file are initially set as follows:

```
#Unified telnet proxy properties
#Daemon configuration example.
#Created: 15/11/2004 wimpi
#Modified: 10/01/2005 bzs

############################
# Telnet daemon properties #
############################

#####################
# Terminals Section #
#####################

# List of terminals available and defined below
terminals=vt100,ansi,windoof,xterm

# vt100 implementation and aliases
term.vt100.class=net.wimpi.telnetd.io.terminal.vt100
term.vt100.aliases=default,vt100-am,vt102,dec-vt100

# ansi implementation and aliases
term.ansi.class=net.wimpi.telnetd.io.terminal.ansi
term.ansi.aliases=color-xterm,xterm-color,vt320,vt220,linux,screen

# windoof implementation and aliases
term.windoof.class=net.wimpi.telnetd.io.terminal.Windoof
term.windoof.aliases=

# xterm implementation and aliases
term.xterm.class=net.wimpi.telnetd.io.terminal.xterm
term.xterm.aliases=

##################
```

```
# Shells Section #
##################

# List of shells available and defined below
shells=RevShell

# shell implementations
shell.RevShell.class=net.wimpi.telnetd.shell.RevShell

##################
# Tuning Section #
##################
tune.delay=0

#####################
# Listeners Section #
#####################
#example: listeners=std,port2,port3
listeners=std

 # std listener specific properties

#Basic listener and connection management settings
std.port=23
std.floodprotection=5
std.maxcon=25
std.servername=localhost
std.serverport=8088

# Timeout Settings for connections (ms)
std.time_to_warning=3600000
std.time_to_timedout=60000

# Housekeeping thread active every 1 secs
std.housekeepinginterval=1000
std.inputmode=character

# Login shell
std.loginshell=RevShell

# Connection filter class
std.connectionfilter=none

# EXAMPLE: port2 listener specific properties
#Basic listener and connection management settings
port2.port=2023
port2.floodprotection=5
port2.maxcon=1
port2.servername=localhost
port2.serverport=8089

# Timeout Settings for connections (ms)
port2.time_to_warning=3600000
port2.time_to_timedout=60000

# Housekeeping thread active every 1 secs
port2.housekeepinginterval=1000

port2.inputmode=character

# Login shell
port2.loginshell=RevShell

# Connection filter class
port2.connectionfilter=none
```

```
# EXAMPLE: port3 listener specific properties

#Basic listener and connection management settings
port3.port=2024
port3.floodprotection=5
port3.maxcon=25
port3.servername=localhost
port3.serverport=8089

# Timeout Settings for connections (ms)
port3.time_to_warning=3600000
port3.time_to_timedout=60000

# Housekeeping thread active every 1 secs
port3.housekeepinginterval=1000

port3.inputmode=character

# Login shell
port3.loginshell=RevShell

# Connection filter class
std.connectionfilter=none
```

You may specify one or more "listeners", which the OETelnetServer uses to listen on various ports. By default, port 23 is used by the "std" listener, though the configuration file also shows how you could configure multiple listeners on other ports. The properties that you may wish to set include:

Port: the tcp/ip port where the telnet server will "listen" for requests
Maxcon: the maximum number of simultaneous connections
Servername: the ip address or name of the OEngineServer the telnet server should communicate with
Serverport: the tcp/ip port of the OEngineServer the telnet server should communicate with
Application: the default application to connect to
UserName: the default username to connect as
Password: the default password to use for connecting

If you configure additional listeners, and set the MAXCON property for a listener to 1, when a user connects to that specific listener, the telnet server will use that listener name as the "process number" in the CTO environment. In this way, it is possible to explicitly control which "process number" each telnet user is assigned to by specifying these alternate telnet port listeners.

For each listener, you may set various properties by prefixing the name of the listener to the property name (ie, "std.serverport", "std.servername", etc.). Note that by setting the various default information for application, username, and password, you can configure either full or partial "auto-connect" into an OpenInsight CTO application.

Local users can interact with the OEngineServer through the "REVCMD" (Revelation Command Line) interface. This is available either as a complete standalone routine (REVCMD.EXE), as a standalone OI form (CTO_STANDALONE_FORM), or as an OI "component" which can be included in other GUI forms (CTO_CHILD_FORM).

When run as a standalone routine, the REVCMD.EXE has the following command-line options:

```
-a <applicationname>
-u <username>
-p <password>
-e <hostaddress>
-n <hostport>
-r <routine to run>
-s <enginename>
-f <startupflags>
-d <shutdownflags>
-w
```

The –a, -u, and –p parameters allow you to "auto-login" to the specified OI application; if set, they will cause REVCMD.EXE to skip prompting for the application, user name, and/or password. The –e allows you to specify the ip address or name of the OEngineServer (the default is "localhost"); the –n allows you to specify the tcp/ip port of the OEngineServer (the default is 8088). The –r parameter allows you to specify the name of a routine to run when first logging in; if the –w option is specified, then REVCMD.EXE will wait until this routine has terminated, and then disconnect. The –s option allows you to specify a specific engine name to connect to, and the –f and –d parameters allow you to define specific startup and shutdown flags used by the OEngineServer.

Any values that are not set will assume the 'default' values as specified in the eserver.cfg file for the OEngineServer, with the exception of username, password, and application; these will be prompted for. If the user enters <return> for any of these values, then the defaults (if any) from the eserver.cfg file will be used.

Note that you can configure each REVCMD session so that it has a specific "process number" in the CTO environment by explicitly passing in the desired process number as the "engine name" parameter (as specified with the –s flag). The engine name should be unique, and should begin with `\\.\` (for example, `\\.\10` will instruct CTO to define this connection as process number 10).

When run as a standalone OI form, the CTO_STANDALONE_FORM combines, in order, the record CTO in the SYSENV table, the record CTO in the MD table of the current application, and the record CTO*<current user name> in the MD table of the current application, to determine its behaviour. This allows for user-level, application-level, and system-level configuration of the interface. The record in all three locations has the following layout:

<1> IP address or name of OEngineServer (default is localhost), or path to revcap32.dll
<2> TCP/IP port of OEngineServer (default is 8088)
<3> foreground color name (default is GREEN)
<4> background color name (default is BLACK)
<5> title (default is OpenInsight Character Interface: <appname>)
<6> program to run (default is none)
<7> font name

The CTO_STANDALONE_FORM, when invoked, will log in to the current application, with the current username and password, and then (if specified) run the program defined in field 6 of the configuration record, and then terminate. If no program is defined in field 6, then 'normal' processing will result (ie, if a login proc/paragraph/command is found, then it will be run, otherwise the user will be left at TCL)

When run as an OI component, users may configure properties of the CTO_CHILD_FORM either in the form designer, or via SET_PROPERTY calls in the CTO_CHILD_FORM commuter module. Other GUI elements can interact with the CTO_CHILD_FORM by the use of the following subroutines from your character programs: CTO_DIALOG_BOX, CTO_GUI_FUNCTION, CTO_GUI_SUBROUTINE, CTO_MSG, CTO_POPUP, CTO_RUNWIN and CTO_SET_PRINTER.

MIOS

When inputting or outputting data in CTO, it is possible to "intercept" and modify the data before it is processed by the normal CTO routines. This MIOS (Modified I/O System) is therefore comparable to OpenInsight's MFS (Modified Filing System). Possible uses of this feature might be to implement i/o redirection, support for alternative input sources, or additional logging of output. To use this functionality, you must create a routine (or routines) that have the following parameters:

SUBROUTINE <routinename>(<code>,<mios chain>,<param1>,<param2>,<param3>,<param4>,<param5>,<param6>,<param7>,<param8>,<flag>)

Where the parameters are:

<code>	: -1 = before input;
	0 = input;
	1 = before print;
	2 = print

<mios chain>	: Subvalue mark delimited list of other mios routines.
<param1> …	: Parameters passed into DO_INPUT or DO_PRINT (depending on wheter this is –1/0 or ½ code)
<flag>	: Set by the invoked MIOS routine. 0 indicates failure, any other value indicates success

You must specify that you want this routine invoked by calling SET_MIOS. SET_MIOS "attaches" your routine to the input/output chain of events. SET_MIOS has the following syntax:

SET_MIOS <routinename>,<calltype>

Calltype must be one of the following values:

1 – Add this routine to the end of the input/output chain
2 – Add this routine to the beginning of the input/output chain
3 – Add this routine to the front of the input/output chain (same as 2)
4 – Set this routine as the only routine in the input/output chain
5 – Remove this routine from the input/output chain

6 – Remove all routines from the input/output chain

Note that you must invoke SET_MIOS each time you log in to your application; therefore, you may wish to add this call to your LOGON proc or paragraph.

Special Filing Systems: DOS, DOSTXT, URL, URLB

CTO and AREV32 support several special filing systems that allow you to access non-traditional data as though it were normal OpenInsight data. The DOS and DOSTXT filing systems allow the user to read, write, and delete Windows files, while the URL and URLB filing systems allow the user to "read" and "write" to web addresses (URLs).

To access the DOS or DOSTXT filing systems, the user would use the table name "DOS" or "DOSTXT", and the "record name" or "record id" should be the Windows file name (either relative to your OpenInsight directory, or with a fully qualified path). The user can then read this record, write to this record, or delete this record programmatically, or through the use of any commands that normally accept a tablename and record id (eg, COPY, ED/EDIT, etc.). Note that on the command line in CTO, the user should quote the file path to ensure proper parsing.

When using the DOSTXT filing system, any carriage return/line feed delimiters in the record are converted to field marks when the record is read; any field marks are converted to carriage return/line feed delimiters when the record is written. When using the DOS filing system, no conversions of the data are performed at all.

Reading from a record whose record name specifies a directory will return special information. The first field will be the word DIR; the second field will be a value-mark-delimited list of the files in that directory; the third field will be a value-mark-delimited list of the subdirectories of that directory.

Examples:

1. Read a record (without any conversions) into a variable:

```
OPEN "DOS" TO DOS.FL THEN
  READ BINARY.INFO FROM DOS.FL,"C:\SOMEIMG.JPG" ELSE BINARY.INFO = ""
```

2. Read a record (with text conversions) into a variable:

```
OPEN 'DOSTXT' TO TXT.FL THEN
  READ RECORD FROM TXT.FL,'C:\SOME.TXT' ELSE RECORD = ''
```

3. Return and process some directory information:

```
OPEN 'DOSTXT' TO DOS.FL THEN
  READ DIR.INFO FROM DOS.FL, "C:\SOMEDIR\*.TXT" THEN
    NUM.FILES = COUNT(DIR.INFO<2>, @VM) + 1
    FOR EACH.FILE = 1 TO NUM.FILES
      THIS.FILENAME = DIR.INFO<2,EACH.FILE>
      READ INFO FROM DOS.FL, THIS.FILENAME ELSE INFO = ""
```

To access the URL or URLB filing systems, the user would use the table name "URL" or "URLB", and the "record name" or "record id" should be the full URL. The user can then either read this record (eg, use an HTTP GET to retrieve the URL information) or "write" to this record (eg, use an HTTP PUT to send information to this URL) programmatically, or through the use of any commands that normally accept a tablename and record id (eg, COPY, ED/EDIT, etc.). Note that on the command line in CTO, the user should quote the URL to ensure proper parsing.

When using the URL filing system, any carriage return, line feed, or carriage return/line feed delimiters in the record are converted to field marks when the record is retrieved; any field marks are converted to carriage return/line feed delimiters when the record is "written". When using the URLB filing system, no conversions of the data are performed at all, making this the appropriate "table" to use when accessing binary data.

After "writing" a record, the user may wish to retrieve the response generated by the URL. This information is available in a common variable named URL_WRITE_RESPONSE@. To access this variable, please insert the URLBFS_COMMON insert record.

Examples:

1. Retrieve a web page:

```
OPEN “URL” TO URL.FL THEN
  READ PAGE FROM URL.FL, “http://www.revelation.com” ELSE PAGE = “”
```

2. Retrieve a binary result from a URL:

```
OPEN “URLB” TO URL.FL THEN
  READ IMG FROM URL.FL, “http://www.revelation.com/icon.jpg” ELSE IMG = “”
```

3. Send information to a URL:

```
$INSERT URLBFS_COMMON
OPEN “URL” TO URL.FL THEN
  WRITE ORDERINFO TO URL.FL, “http://www.somepage.com/REST/neworder”
  * Examine result of the call
  RESPONSE = URL_WRITE_RESPONSE@
```

Precompiler options

```
*#INSERT <type>
```

Where <type> is:
0: Don’t touch $INSERT at all; add in any $INCLUDE items into the mainline code 1x per insert (current normal OI-mode behavior)
1: Add in any $INSERT or $INCLUDE items into the mainline code, 1x per insert (current normal CTO and AREV32 behavior)
2: Add in any $INSERT or $INCLUDE items into the mainline code as many times as they are specified in the code
3: Don’t touch $INSERT at all; add in any $INCLUDE items into the mainline code as many times as they are specified in the code

So, in a CTO or AREV32 program that uses $INSERT and you _know_ the insert item is already in OI format and doesn’t need to be precompiled, you could specify:

```
*#INSERT 0
```

To have the precompiler skip over those inserts.

If, instead, you had a program where you wanted the insert items to be converted, and to be inserted however many times and wherever they were specified, you could specify:

```
*#INSERT 2
```

The Master Dictionary (MD)

The MD table was originally introduced to support CTO conversions from "traditional" multivalue databases, but has functionality that may be useful to all OpenInsight developers. The MD table is in many respects comparable to the VOC table, though the records stored in the MD table are typically more compatible with the original PICK (R83) MD file.

Records in the MD table fall into four broad classes:
1. "Pointers" to executable routines;
2. Batch routines (or "pointers" to batch routines);
3. "Pointers" to tables; and
4. User-defined records

OpenInsight, CTO, and AREV32 look in both the MD and VOC table for any relevant record when a stored procedure is invoked, or when a table is about to be opened.

The order in which these tables are searched will vary depending on whether the invocation of the stored procedure occurs in the OpenInsight, CTO, or AREV32 environmen - in CTO, the MD table is searched before the VOC table; in OpenInsight and AREV32, the VOC table is searched before the MD table.

It is important to remember that, since OpenInsight, CTO, and AREV32 check BOTH the MD and VOC tables by default, you may experience unexpected behavior if records with the same name are stored in both the MD and VOC tables.

In all cases, though, the designated table(s) are searched for the executable pointers or batch routines, or table pointers, as appropriate; if no entries are found in these tables, then the default system behavior occurs.

It is also possible to customize which table(s) are checked by CTO and AREV32 for executable pointers. If CTO or AREV32 find a record named PROCPATH in the MD table, the list of tables specified in field 2 of this record (@VM-delimited) defines which tables should be checked for executable and batch routines and pointers.

A special table named MD_MASTER contains the basic, default information that should be contained in any application's MD table; if for any reason an application does not have an MD table, or must have a new MD table created, the contents of the MD_MASTER table should be copied into the application-specific MD table for proper system operation.

Pointers To Executable Routines

When OpenInsight prepares to run a stored procedure, it first locates where the "object code" for that stored procedure should be loaded from. By default, this is the SYSBOJ table; however, the MD table allows you to specify explicitly the table (and even the record name) of the object code to load. Using these "catalog pointers", developers can store object code in non-SYSOBJ tables. The layout of the catalog pointer is:

ID: Name of stored procedure
1. P
2.
3.
4.
5. <object code table name>
6. <object code record name>

This is comparable to the "catalog pointers" that are normally found in the VOC table (as generated by AREV/AREV32), though the VOC record layout is:

ID: Name of stored procedure
1. RBASIC (or VERB)
2.
3. <object code table name>
4. <object code record name>

Pointers to U2 executables (accessed via the U2 BFS) can also be defined; these are identified by "U2" in field 1 of the record.

Addionally, internal CTO executable pointers are located in the MD table; these are identified by a "V" in field 1 of the record.

Batch Routines (Or Pointers To Batch Routines)

The MD table allows you to define batch routines that can perform tasks nearly as complicated as basic+ stored procedures. The MD table supports both PROC and PARAGRAPH batch syntax. A PROC can be identified by the letters "PQ" in field 1 of the MD record; a PARAGRAPH will contain the letters "PA" or "S" in field 1.

Though a full discussion of the PROC and PARAGRAPH syntax is beyond the scope of this document, the following functionality is supported in PROCS:

Proc Chaining - [], ()
User Exits - U01AD, U31AD, U01A6
Primary and Secondary Buffer Manipulation - A, B, F, +, -, R, S
Labels/GoTo
Output Buffer Manipulation - H, STON, STOFF, P, PW, PX, PH
Console Input and Output - O, T, I, D, IH, IP, IS, IN
Comments - C
Conditional Evaluation - IF {#}{S|E|A} {=|>|<|]|[|#}

The following functionality is supported in PARAGRAPHS:

Console input and output - CLEARPROMPTS, DISPLAY, <<prompts>>
Execution control - ABORT, LOOP, REPEAT, GO

Conditional Evaluation - IF
Output Buffer Manipulation - DATA

Pointers To Tables

The internal processing of tables in OpenInsight has been enhanced to allow for "redirection" through MD records. When a table is accessed, OpenInsight will look in the MD table to see if there is a pointer record with that table name; if a pointer record does exist, then the information in that record will be used to find the proper table to open. The layout of these pointer records is:

ID: Name of table
1. Q
2. {<application name>}
3. {<real table name>}

Because the first field of this pointer record contains a "Q", these are often referred to as "q-pointer records". If an application name is specified in field 2, then that application must contain the requested table (either attached to the database definition, or defined in that application's MD table); if left blank, then the current application is assumed. If the real table name is specified, then that is the actual table that will be accessed; if left blank, the originally requested name is used.

For example, the MD of the SYSPROG application might contain the following record:

ID: CUST
1. Q
2. EXAMPLES
3. CUSTOMERS

Any requests to access a table named CUST while in the SYSPROG application will actually access the CUSTOMERS table in the EXAMPLES application.

RTP9's q-pointer handling, by default, will work in the “normal PICK way”: any time a q-pointer record is accessed, it is “resolved” at that time, and so any changes to any q-pointers are immediately reflected. This provides the ultimate flexibility, but at the cost of performance – each time the q-pointer is accessed, we must trace through and find where it *_really_* lives.

Now, If desired, developers/users can set a flag in SYSENV (CFG_QPTR, and its various permutations – CFG_QPTR*app*user, CFG_QPTR**user, CFG_QPTR*app) to allow for the q-pointer information to be retained after it has been resolved. This means that any changes to the q-pointer, after it has been accessed, will NOT be immediately seen; instead, users will need to exit/re-enter OI/CTO/AREV32/whatever to see their changed q-pointer, or they can run DETACH_TABLE on the q-pointer. This provides much better performance, at the cost of the flexibility that “traditional MV” users expect.

User-Defined Records

Any other user or configuration record might be stored in the MD table; traditionally, these should contain an "X" in field 1.

OpenInsight Tools for U2 Integration

It is now possible to interactively run U2 programs and commands from within the CTO environment. At the standard CTO TCL prompt (“>”), issue the command:

```
>U2TCL
```

If this is the first time you are issuing the command during the current session, you will be prompted for the name of the previously-defined U2 Connection to use. If successful, you will be presented with the U2TCL TCL prompt:

```
U>
```

From this prompt, any command you issue will be executed on the host system defined by the U2 Connection. You will remain at the U2TCL prompt, able to run multiple commands, until you exit. When you wish to exit the U2TCL and return to CTO, issue either the command:

```
U>QUIT
```

Or

```
U>EXIT
```

If you wish to re-enter U2TCL during the same TCL session, just reissue the command:

```
>U2TCL
```

And you will be reconnected using the previously-defined U2 Connection. If you wish to change the connection you are using, issue the U2TCL command along with the name of the U2 Connection you wish to use:

```
>U2TCL OtherU2ConnectionName
```

Programs run in the U2TCL can also take advantage of some advanced OpenInsight functionality. A series of support routines allows developers to include, in their U2 programs, calls to OpenInsight’s messagebox, popup, and other GUI functions. To install these support routines on a connection, issue the OIDEPLOY command from within the U2TCL:

```
U>OIDEPLOY
```

You will be prompted to enter the name of the U2 file where you want these support programs to be installed. U2TCL will put the support programs in this file, compile, and catalog them. For your convenience, there is also a record created in this file, named CTO_FUNCTIONS, which contains all the function definitions and that can be included in your U2 programs. The support programs include CTO_MSG, CTO_POPUP, CTO_RUNWIN, CTO_RUNWIN_WAIT, CTO_DIALOG_BOX, CTO_GET_PROPERTY, CTO_SET_PROPERTY, CTO_SET_PRINTER, CTO_GUI_FUNCTION, and CTO_GUI_SUBROUTINE. For more information on these routines, please see <<insert reference here>>

It is also possible to invoke the U2TCL to execute a single command and then terminate. The syntax is:

```
U2TCL <connectionName>,<command to run>
```

For example:

```
>U2TCL MyU2ConnectionName,RUN BP SOMEPROGRAM
```

Will run the program SOMEPROGRAM located in the BP file on the host defined by MyU2ConnectionName.

If there are U2 commands that you will run frequently, you can define MD entries that make running these commands more convenient. These are comparable to “cataloging” a CTO program, or creating a CTO “paragraph” or “proc”. The record layout of these MD entries is:

ID: <commandname>
1. U2
2. <connectionname>
3. <full command to run>

For example, to define a CTO command called U2TEST that runs on the connection MyU2ConnectionName and runs the U2 command RUN BP SOMEPROGRAM, you would create an MD record that looked like:

ID: U2TEST

1. U2
2. MyU2ConnectionName
3. RUN BP SOMEPROGRAM

As another convenience, if you have a U2 program file attached to your OpenInsight system, and you wish to run a program from that file, you can issue the U2RUN command. The syntax for the U2RUN command is similar to the "normal" CTO RUN command; you must specify the name of the file containing the program you wish to run, and the program name. For example:

U2RUN BP SOMEPROGRAM

The U2RUN command will retrieve the information about the U2 Connection from the definition of the U2 file contained in the OpenInsight database manager; it will (if necessary) open the connection to that U2 Connection, and invoke U2TCL to run the specified program.

U2/OI Integration Routines

You may run U2 programs via U2TCL (or via U2RUN, discussed below), without major modifications. However, if these programs will be run in the CTO environment exclusively, you can enhance them so that they can use OpenInsight functionality, such as popups, message boxes, and OIPI printer management. This functionality is available by adding calls to CTO_DIALOG_BOX, CTO_GET_PROPERTY, CTO_GUI_FUNCTION, CTO_GUI_SUBROUTINE, CTO_MSG, CTO_POPUP, CTO_RUNWIN, CTO_RUNWIN_WAIT, CTO_SET_PRINTER, and CTO_SET_PROPERTY. See the standard CTO documentation for information on how to use these functions.

In order to make these functions available to the U2 programs, you must put these support routines on the U2 system. This needs to be done only on those systems where you will modify the U2 programs to call these functions. From within U2TCL, you can issue the special command OIDEPLOY. You will be prompted to enter the name of the U2 program file where these should be created. U2TCL will then verify that the file is accessible, copy these routines to the U2 system, compile, and catalog them.

U2RUN

U2RUN allows you to run U2 programs without entering the U2TCL environment. The syntax for using U2RUN is:

U2RUN <OITableNameOfU2File> <ProgramName>

For U2RUN to work, you must have defined (via the OpenInsight Table Manager) a "connection" to the U2 system, and then defined (as accessible in OpenInsight) the file where the desired program resides. U2RUN will use the connection and table information to connect to the U2 system and run the specified program.

"Cataloged" U2 Programs

CTO supports the ability to "catalog" a U2 program so that it can be invoked just by entering the catalog name at CTO TCL. To create a "cataloged" U2 program, you must manually create a record in the MD. Name this MD record whatever you wish to use when invoking the U2 program; field 1 of the record must contain a "U2", field 2 contains the connection name to the U2 system, and field 3 contains the full statement to run on the U2 system.

Unsupported Functionality

Since the U2 integration features only offer a basic level of U2 emulation, the following functionality is _*not*_ supported via U2TCL, U2RUN, or cataloged U2 programs:

- U2 pagination is not passed through to the U2TCL routine – for example, routines like LISTF keep going until they're complete;
- In an associated issue, single-character input is not supported;
- Keystrokes entered in response to prompts from the U2 system may be echoed in the U2TCL environment

Index

REVELATION
S O F T W A R E

Revelation Software, Inc
99 Kinderkamack Road, Suite 109
Westwood, NJ 07675
U.S.A
Toll Free: 800-262-4747
Phone: 201-594-1422
Fax: 201-722-9815
www.revelation.com

Revelation Software Ltd.
45 St Mary's Road
Ealing,
London, W5 5RG
U.K.
Phone: +44 0 208 912 1000
Fax: +44 0 208 912 1001
www.revsoft.co.uk

Software Australia Pty Ltd.5
PO Box 300
Brookvale, NSW 2100
Australia
Phone: +61 2 8003 4199
Fax: +61 2 9932 6099
www.revelationsoftware.com.au

Revelation Software is a division of Revelation Technologies, Inc.

Part No.125-826