

Transaction Processor

Version 4.3.x.x

Programmer's Guide
Revision 1.410

CLE|E|O

RESTRICTED RIGHTS

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (C)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Manufacturer is:

Cleo Communications

4203 Galleria Drive

Loves Park, Illinois 61111

Phone: 800.233.2536

Fax: 815.654.8294

Email: salesEN@cleo.com

Web Site: www.cleo.com

Support: 1.866.444.2536 or SupportEN@cleo.com

Cleo Communications reserves the right to, without notice, modify or revise all or part of this document and/or change product features or specifications and shall not be responsible for any loss, cost or damage, including consequential damage, caused by reliance on these materials.

This document may not be reproduced, stored in a retrieval system, or transmitted, in whole or in part, in any form or by any means (electronic, mechanical, photo-copied or otherwise) without the prior written permission of Cleo Communications.

©2012 Cleo Communications All rights reserved. Cleo is a registered trademark of Cleo Communications. All other brand names used are trademarks or registered trademarks of their respective companies.

Table of Contents

Introduction.....	6
Component Overview.....	8
Architectural Diagram.....	8
Component Descriptions.....	8
Functional Overview.....	9
Using the Transaction Processor Functions.....	12
Loading the Connector.....	13
Initializing Dynamic Failover.....	13
Connecting To a Host Session.....	14
Keeping a Session.....	15
Setting Transaction Input.....	16
Resetting Input Fields.....	16
Running a Transaction.....	17
Accessing Transaction Output.....	18
Obtaining Session Status.....	19
Obtaining the Current Screen Name.....	21
Obtaining the Base Screen Name.....	21
Obtaining Session Id.....	22
Turning off Java Garbage Collection.....	22
Freeing up a Host Session.....	23
Production Run.....	25
Processing and Application Considerations.....	27
Start Up/Shutdown Processing Considerations.....	27
Transaction Processor Startup Processing.....	27
Transaction Processor Shutdown Processing.....	27
Interactive Response Application Considerations.....	28
Considerations for 3270/5250 Transaction Development.....	28
Repark Transaction Development.....	28
Recovery Transaction Development.....	29
Defining Multiple Recovery Transactions.....	32
Alternative Access to the TP.....	35
Using the VoiceXML Connector.....	35
Overview.....	35
Getting Started With The VoiceXML Connector.....	35
Functions Provided.....	38
Using The XML Connector.....	40
Overview.....	40
Getting Started with the XML Connector.....	40
Invoking the XML Connector.....	40
Processing Return Data.....	43
Using the XML Connector with Avaya IR and Vonetix.....	45
Getting Started with the XML Connector, Avaya IR, and Vonetix.....	45
Accessing the XML Connector with IVR Designer.....	46

Using the Web Service Connector.....	52
Overview	52
Getting Started with the Web Service Connector	52
Invoking the Web Service Connector	52
Code Examples	53
Visual Basic .NET	54
Running a Simple Transaction	54
Running the Sample Transactions.....	55
Java	56
Using the TP Connector	56
Using the XML Connector.....	57
Perl.....	58
Using the Web Service Connector	58
VoiceXML.....	59
Using the VoiceXML Connector	59
Running the Sample Application.....	61
Disclaimer.....	61
Prerequisites.....	61
Getting Started Using the 3270 sample Application	61
Sample 3270 Transaction Files.....	63
login.xml	63
getlist.xml.....	63
getlist_park.xml.....	63
logout.xml	64
scdefs.xml.....	64
Troubleshooting	68
Troubleshooting Tips.....	68
File Location Information.....	68
Location of Log Files	68
Location of Cleo Configuration Files on Windows	69
Location of Cleo Configuration Files on UNIX/Linux	70
Avoiding Common Problems	70
Application Startup	70
Application Call Processing.....	71
Problems with Sessions getting Reparked	71
Error Handling.....	71
Return Code Chart.....	72
Log Messages	75
Log Format and Configuration Levels	75
Message Descriptions.....	76
Client Log Message Descriptions	107
Dynamic Failover	111
Description.....	111
System Overview.....	111
Component Diagram.....	112
Component Descriptions	112

Service Pool Management	113
Run Time Data Flow	114
Dynamic Fail Over On Reserve Session	115
Exception and Error Handling.....	115
Programming Considerations	117
Glossary	119

Introduction

Programs that access mainframe data with 3270 or 5250 applications generally use one of IBM's Application Program Interfaces (APIs) such as the High Level Language API (HLLAPI). Cleo's Transaction Development Kit is a set of tools that provides a simplified mechanism for defining and processing custom "transactions". These transactions contain all the information necessary to interact with a set of 3270 or 5250 display screens. Thus, once transactions have been created, the developer can send and extract mainframe data using fewer API calls than traditionally required. This simplifies the development process that in turn leads to reduced development time and higher reliability.

The *Transaction Processor* is one component of Cleo's *Transaction Development Kit*, a set of tools designed to simplify program access to mainframe data. This kit is composed of:

- *Transaction Designer* (TD) – records screens and creates transactions.
- *Transaction Processor* (TP) – processes the transactions in real time.
- *Transaction Processor Client* (TP Client) – provides the basic API functions.
- A set of *Transaction Processor Administration, Configuration, and Connection Manager* tools (TP Admin) – web based utilities used to set configuration parameters, provide session monitoring, display log messages and statistics for the TP Service and initialize the connection pools for redundant and secure client support.

Other supporting documents include:

Transaction Designer User's Guide

This guide explains how to record screen clips and create transactions.

Transaction Processor Quick Start Guide

This guide lists prerequisites and provides an installation procedure for the Transaction Processor.

Transaction Processor Client Quick Start Guide

This guide lists prerequisites and provides an installation procedure for the Transaction Processor Client.

Transaction Processor Administration Guide

This document describes the functions provided by the Transaction Processor Configuration, Administration, and Connection Manager Tools.

The ***Administration Guide*** and ***Programmer's Guide*** can be found on the installation CD and are installed in the %TP_HOME%\docs directory in PDF format. (For example: C:\Program Files\CleoTP\docs\AdminGuide.pdf)

This document, the *Cleo TP Programmer's Guide*, describes TP-supported functions. It also includes code snippets, a sample application, and troubleshooting information. There is also a Glossary of Terms at the end of the document for reference.

The intended audience of this document is presumed to be familiar with Visual Basic or Java application development, as well as having some knowledge of using 3270 application screens on a mainframe host, or 5250 application screens on a midrange host.

Component Overview

The following diagram illustrates the components involved when the Cleo Transaction Processor (TP) is run. The user application invokes Cleo TP functions through a Java, JSP (for VoiceXML), or .NET interface. In order to process these functions, the Transaction Processor reads configuration information from the file system (including the sessions file, which contains mapping of session ids to transaction set names). When a transaction is run, the Transaction Processor uses the information in the XML files for the specified transaction to interact with the host server for accessing mainframe data. The transaction XML files were previously stored in the Transaction Sets directory by the Transaction Designer (TD).

Architectural Diagram

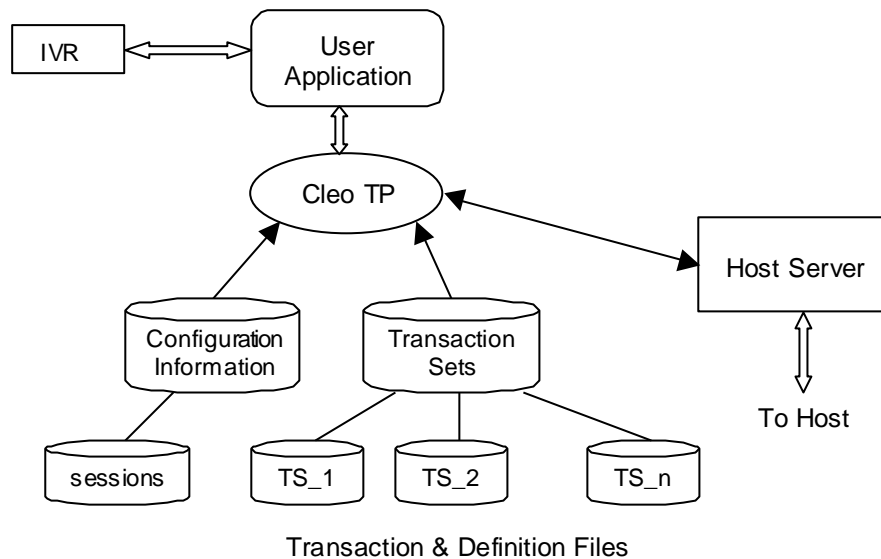


Figure 1: Architectural Diagram

Component Descriptions

<i>Component</i>	<i>Description</i>
Cleo TP	Cleo Transaction Processor that consists of a client JavaBean API, accessible directly through Java calls or indirectly through a .NET or JSP (for VoiceXML) layer that provides the core functions and a service component that runs transaction sequences against a host application.

<i>Component</i>	<i>Description</i>
Configuration Information	Directories and files containing information stored by the Cleo TP Administrator tool. Includes the sessions file that contains mapping of session ids to transaction set names.
Host Server	3270/5250 server used for accessing mainframe applications.
TD	Transaction Designer. A standalone GUI-based utility used to create recording clips that are then used to define transaction sequences run by the Transaction Processor.
Transaction Set	A collection of transaction sequences defined and created by the Transaction Designer. It consists of a directory containing transaction files and screen definitions that will be used during Transaction Processor execution.
Transaction Set directory	The top-level directory where all transaction sets will be accessed and stored. Each transaction set is stored in a subdirectory according to the transaction set name.
Transaction and Definition Files	Text files created by the Transaction Designer in XML format that contain screen definitions and transaction sequences defined using the TD. These files are used by the Transaction Processor to navigate through a particular mainframe application.
User Application	A user application, such as a voice response application (IVR) that handles phone calls

Functional Overview

The Cleo Transaction Processor (Cleo TP) provides access to 3270 mainframe data through a simple transaction-based interface. It provides access to this data from a Java application, JSP, VoiceXML, or .NET application as well as applications that can employ Web Services or XML.

The Transaction Processor consists of three internal components: a Connector, a Service and an Executor. The Service, once loaded, runs in the background, and is responsible for initializing, parking, and recovering host sessions in a session pool. The component that interfaces with the user's application is the Connector, a Java, .NET, VoiceXML, Web Service, or XML API. The user's application must call the Connector and request the execution of transactions through specific functions that the Connector recognizes. The following chart defines these functions.

<i>Function</i>	<i>Description</i>
<i>InitializeFailover</i>	No longer (since Version 4.3.0.8) required if dynamic failover to a secondary TP service is to occur. See the Administration Guide for how to configure the Connection Manager to provide this capability. This function is ignored. Dynamic Failover is configured to be ENABLED/DISABLED using the TP Admin.
<i>SetFailoverRetryInterval</i>	Used to change the frequency at which the Connection Manager checks that a connection is available to all TP services that have been configured.
<i>reserve</i>	Reserves an available host session. Specify a <i>transaction set name</i> for this session as an input parameter. This reserves (i.e., “connects to”) a session assigned to the named transaction set in the sessions file, and removes this session from the session pool.
<i>keepSession</i>	After a session is reserved, invocation of <i>keepSession</i> allows the application to keep control of the session, even when recovery is required. If <i>keepSession</i> is not invoked, then control of the session will be taken away from the application in a situation that requires recovery.
<i>addInput</i>	Specifies one or more input fields to be sent to the host. Invoke once for each input field, specifying the field name and value. Note that field names are unique across all screens in the transaction, so field names do not have to be associated with a screen or document name.
<i>resetInput</i>	Clears input field values once no longer needed or when running a new transaction.
<i>runTransaction</i>	Invokes a specified transaction by name. If the specified transaction requires input field values, specify using the <i>AddInput</i> function prior to invoking <i>RunTransaction</i>
<i>getOutput</i>	After successfully running a transaction, this function retrieves data received from the host. If a transaction returns multiple pieces of information, invoke <i>GetOutput</i> multiple times, once for each output field name. These values remain available until the next time <i>RunTransaction</i> is invoked.
<i>getOutputDelimited</i>	When a transaction is run recursively, the data for each output (fromHost) field is collected for each iteration of the transaction. Invoking this function will return all of the values collected for the particular field as a pipe delimited String.
<i>getOutputArray</i>	When a transaction is run recursively, the data for each output (fromHost) field is collected for each iteration of the transaction. Invoking this function will return all of the values collected for the particular field as an array.
<i>getCurrentScreen</i>	Returns the name of the screen that was last visited while running a transaction.

<i>Function</i>	<i>Description</i>
<i>getBaseScreen</i>	Returns the name of the base screen of the screen that was last visited while running a transaction.
<i>getSessionId</i>	Returns the session Id of the currently-reserved session. Useful for debugging purposes.
<i>getSessionStatus</i>	Returns the status of the session. See Obtaining Session Status for a description of all the status strings that can be returned.
<i>release</i>	Releases a host session (i.e., disconnects from a session) to make it available for access by other applications that issue the reserve function.
<i>CleoGCoff</i>	Optional turns off Java Garbage Collection from being done during a release API request.

Using the Transaction Processor Functions

This section provides the syntax for each of the Transaction Processor functions, including required input parameters, possible return codes and examples of code. It also includes a flowchart of a typical production run sequence using these functions.

To invoke Transaction Processor functions, include them from within any Visual Basic 6.0/.NET application, VBA macro, JSP application, or Java application. Note that the illustrated code examples use Visual Basic .NET and Java. See the section on Alternative Access to the TP for a discussion on using the other available connectors (VoiceXML, XML, Web Service) for access from other environments. Be sure to first start the Transaction Processor service before attempting to use any of the Transaction Processor functions - see Start Up/Shutdown Processing Considerations.

A Transaction Set must have been created/defined using the Transaction Designer (TD). Transaction Set files must reside in a folder having the same name as the Transaction Set, which have been transferred into the Transaction Processor transaction directory (i.e., %TP_HOME%\trans where the default %TP_HOME% is *C:\Program Files\CleoTP* on windows, */opt/cleotp* on UNIX/Linux). Refer to the *Transaction Designer User's Guide* for detailed instructions on creating, publishing, and transferring transaction sets.

Refer to page 71, **Return Code Chart** section, for suggestions on handling the return codes.

Loading the Connector

In order to use the functions provided by the Transaction Processor (TP) when using Java, the cleotp.jar file must be included in the Java classpath. On Windows, this file exists in %TP_HOME%\lib (e.g. C:\Program Files\CleoTP\lib) and on UNIX/Linux, in /opt/cleotp/lib. Additionally, the TPConnector class must be imported by adding “import com.cleo.htp.TPConnector;” to the top of the calling class source file.

Before invoking Transaction Processor functions, first load the TP Connector:

C# .NET and Java Example:

```
TPConnector tp = new TPConnector(ipAddress, port);
```

The TP Connector is initialized through the use of one of three available constructors:

TPConnector() – sets up the TP Connector to connect to a local TP Service as defined in the cleotp.cfg file when a *reserve* is issued.

TPConnector(ipAddress, port) – sets up the TP Connector to connect to the TP Service accessible at the given IP address and port when a *reserve* is issued.

TPConnector(connpool) – sets up the TP Connector to use the secure connection and failover options as configured in the Connection Manager for the given TP connection pool when a *reserve* is issued.

Be sure to program a check into your user application to make sure the TP Connector is available. That is, include a check to verify that the TP Connector is **not null** before invoking its functions (not null=available; null=not available).

Initializing Dynamic Failover

The TP Client provides configurable failover support to multiple Transaction Processor Services through the use of the Connection Pool, set up using the Connection Manager. See “Using the Connection Manager” in the Administration Guide for instructions on configuring connections in the Connection Pool. See the section Dynamic Failover for a complete description and programming recommendations.

InitializeFailover()

Since Version 4.3.0.8 of the Cleo TP and Cleo TP Client, in order for an application to utilize dynamic failover, it NO LONGER must call this function once after creating the TPConnector. Prior to Version 4.3.0.8 an application needed to call this function to enable dynamic failover. Now dynamic failover is controlled using the TP Admin to either ENABLE or DISABLE the Cleo Fail Over Monitor Service. The Cleo Fail Over Monitor Service updates a file in the

“resources” directory, “connPool.xml”, that contains the current status of all connections as to being “In Service” or “Not In Service(unavailable). The TP Service software uses this status to determine whether to fail over to another connection or not.

The Cleo Failover Monitor Service, when ENABLED, periodically attempts to connect to each TP Service configured in the Connection Pool, marking those it cannot connect to as UNAVAILABLE. Thus, a list of AVAILABLE TP Services(in the connPool.xml file) is kept for the next time a session is requested using the reserve function. The default time for checking the connections is every 1 hour.

The *InitializeFailover()* function is no longer used by the Cleo TP Client software. However, the function will be processed and no action will be taken. So older applications do not need to be changed to remove the *InitializeFailover()* function calls.

Java Example:

```
tp.InitializeFailover();
```

SetFailoverRetryInterval(minutes)

This function is NO LONGER used to change the frequency at which the connections to the configured TP Services are checked for being accessible. Use the TP Admin to configure a Fail Over Retry Interval.

However, the function will be processed and no action will be taken. So older applications do not need to be changed to remove the *SetFailoverRetryInterval(minutes)* function calls.

```
tp.SetFailoverRetryInterval(3);
```

Connecting To a Host Session

reserve(transactionSetName)

where *transactionSetName* is a string containing the name of the transaction set as defined by the TD. This action establishes the connection to the TP Service if the connection is not already open. Additionally, it reserves an available host session. The session assigned to the named transaction set in the sessions file will be controlled by the application that called reserve and will be removed from the session pool so that no other applications can take control of the session.

Upon execution of the function, the Transaction Processor returns one of the following integer return codes:

- Completed with success (0)
- Completed with success after failover (100)
- Completed with success after failover due to sessions being host down (101)
- Error occurred:

- System error (-108)
- Host server error (-109)
- Session pool not available (-113)
- Session not configured or all in use (-114)
- Session has already been reserved (-123)
- Sessions are HOST_DOWN (-126)
- Unable to failover to secondary server (-127)
- Invalid pool name provided (-128)
- Lock Error on connPool.xml file (-129)
- Failover not initialized (Obsolete – no longer returned) (-130)

Refer to the **Return Code Chart** on page 71 for more information.

VB Example:

```
rc = TP.reserve("sample")
```

Java Example:

```
int rc = tp.reserve("sample");
```

Keeping a Session

keepSession()

No input parameters required. This function allows the application to keep control of a previously reserved session, even in a situation where recovery is required. A true or false return code indicates success or failure. If *keepSession* is not invoked, then control of the session will be taken away from the application in a situation that requires recovery, such as the arrival of an unexpected screen from the host.

However, if *keepSession* is invoked, the application is entirely responsible for recovery procedures, including putting the session at the **parked** screen, and returning the session to the pool via the *release* function.

WARNING: If the application terminates, normally or abnormally, without releasing the session, it will not be available for use by another application until the session is restarted.

VB Example:

```
TP.keepSession()
```

Java Example:

```
tp.keepSession();
```

Setting Transaction Input

addInput(fieldName, fieldValue)

where *fieldName* is the name of an input (toHost) field, as defined in the TD, and *fieldValue* is a string containing the field's contents to be sent to the host. Use this call repeatedly if there are multiple fields whose contents defined by the TD need to be overridden.

To clear the contents of an input field on a screen, use *addInput* and specify the *fieldValue* as "@F". This causes the HLLAPI Erase-to-End-of-Field function for the field specified by *fieldName*.

When an input field value contains the AT sign character (@), 2 AT signs must be specified in the value when using the TD Validator AND when using the TP on Linux and Solaris. The "@" should NOT be doubled when using the TP on Windows.

When using the TP on any platform, an input field value containing "@" must be set using the macro table OR by using the *addInput* function call, as shown below.

The following function call sets the contents of the "userid" field to "@logonid1":

Java Example:

```
tp.addInput("userid", "@logonid1"); // TP on Windows
tp.addInput("userid", "@@logonid1"); // TP on Solaris or Linux
```

The following function calls set the contents of the "command" field to "logontso":

VB Example:

```
TP.addInput("command", "logontso")
```

Java Example:

```
tp.addInput("command", "logontso");
```

Resetting Input Fields

resetInput()

No input parameters required. This function clears the input once it is no longer needed, or when it is necessary to restart the process of adding input. A true or false return code indicates success or failure.

To avoid errors from residual toHost field name/data, when running different transactions, the `resetInput` function must be invoked to clear out any previously set input field names and values before executing `addInput` for the next transaction to be run.

VB Example:

```
TP.resetInput( )
```

Java Example:

```
tp.resetInput( );
```

Running a Transaction

runTransaction(transName)

where *transName* is the name of a transaction sequence as defined in the TD. The Transaction Processor sends each of the input fields in the transaction to the host, waiting for the specified returned screen after each transmission. If the `addInput` function was used to set one or more field values before running the corresponding transaction, then those are the field values sent to the host.

If the transaction cannot be run successfully due to receipt of an unrecognized screen, that screen is stored in a file named `<transName>.htp` in the “scr_dump” folder where the TP was installed. Use the TD to read and display these screens. Each *.htp file will not grow in size beyond 1 megabyte. Once this size is reached, no further screens are written to the file – thus, it is recommended that these files be moved or deleted periodically.

Upon execution of the function, the Transaction Processor returns an integer return code, which corresponds to one of the following:

- Completed with success (0)
- Transaction partially run due to:
 - Host server error (-109)
 - Wrong screen (-103)
 - Unknown screen (-104)
 - Wait timeout (-110)
 - Undefined screen (-111)
 - Missing screen identifiers (-119)
 - OIA error (-116)
 - Invalid field value (-106)
- Transaction not run due to:
 - No session reserved (-102)
 - Invalid input field name (-105)
 - Unknown transaction (-107)
 - Ignorable transaction (-118)
 - Invalid XML (-122)

- System error (-108)
- Session not reserved or unavailable (-114)

Refer to the **Return Code Chart** on page 71 for more information.

VB Example:

```
rc = TP.runTransaction("login")
```

Java Example:

```
int rc = tp.runTransaction("login");
```

The Transaction Processor sends the fields defined in the “login” transaction to the host.

Accessing Transaction Output

getOutput(fieldName)

where *fieldName* is the name of an output field as defined in the TD. This function retrieves data received from the host for the specified field. This function can be called repeatedly to retrieve multiple pieces of information (once for each field name). These values remain available until the next time the same transaction is run. If the *fieldName* is invalid, the return value will be **null**. If the *fieldname* does not contain a value, perhaps due to the *runTransaction* failure, the return value will be an empty string (“”).

VB Example:

```
today = TP.getOutput("date")
```

Java Example:

```
String today = tp.getOutput("date");
```

This sets the string variable “today” to the contents of the “date” output field, retrieved from the host.

getOutputDelimited(fieldname)

getOutputArray(fieldname)

In some cases, the same transaction may be run recursively. In other words, the same transaction may be run over and over until a certain condition is met. In this situation, the transaction processor collects the value for each host field for every run of that transaction. Therefore, it is possible to have multiple values of the same output (fromHost) field. These multiple values can be accessed using these functions.

getOutputDelimited() returns a pipe delimited string of all of the values for that field.

VB Example:

```
theDay = TP.getOutputDelimited("date")
```

Java Example:

```
theDay = tp.getOutputDelimited("date")
```

Example Output: 12-12-04|12-13-04|12-14-04|12-15-04|12-16-04|12-17-04|12-18-04

The leftmost value represents the first run of the transaction while the rightmost value represents the last run of the transaction.

getOutputArray() returns an array containing of all of the values for that field.

VB Example:

```
theDay = TP.getOutputArray("date")
```

Java Example:

```
theDay = tp.getOutputArray("date")
```

Example Output: date[0] = 12-12-04
 date[1] = 12-13-04
 date[3] = 12-14-04
 date[4] = 12-15-04
 date[5] = 12-16-04

The first index in the array represents the first value of the field that was received while the last index in the array represents that last value of the field that was received.

NOTE: To further understand this subject, please see the section on Conditional Transactions in the Transaction Designer User's Guide.

Obtaining Session Status

getSessionStatus

This function returns a string indicating the status of the session reserved by the user application.

Commonly returned strings:

<i>String</i>	<i>Description</i>
ASSIGNED	Session has been reserved by an application.
PARKED	Session is in the session pool and available for use. The screen last received from the host for the session is the parked screen. The parked screen is defined to be the last screen in the login transaction, or park transaction if it exists.
RECOVERING	TP is attempting to restore the session to a PARKED state.
AVAILABLE	Session is being initialized during TP start-up.
DEACTIVATED	Session is not in the session pool and thus cannot be reserved. The session viewer can be used to view the screen.
IN_TRANSACTION	TP is running a transaction.

These strings are returned less frequently:

<i>String</i>	<i>Description</i>
RELEASED	Transaction Processor (TP) has completed running the login and optional park transactions
REPARKING	TP is running a <transaction>_park transaction
LOGGING_IN	TP is running the login transaction
LOGGED_IN	TP has completed running the login transaction
PARKING	TP is running the park transaction
LOGGING_OUT	TP is running the logout transaction
LOGGED_OUT	TP has complete running the logout transaction
COMM_CHECK	Host communications have failed for the session
KB_LOCKED	Session is not allowing input due to previous attempt to send invalid data to screen (e.g., field value is longer than field width on screen)
HOST_WAIT	Host application for this session is busy processing data
NO_SESSION_RESERVED	No session reserved

VB Example:

```
status = TP.getSessionStatus
```

Java Example:

```
String status = tp.getSessionStatus();
```

The variable *status* is defined as a string.

Obtaining the Current Screen Name

getCurrentScreen

This function returns the name of the most-recently visited screen for the currently-reserved session. If no session has been reserved, it returns that information. This is useful for diagnostic purposes.

An application can use this function to retrieve the name of the screen currently being displayed. In most cases, only one screen will be returned. However, if the last screen that was processed in the transaction was an “**AnyScreen**” screen, or the current screen does not match (i.e. contain the same identifiers as) the screen defined by the transaction, then the screen that most closely matches the actual screen may be returned. It is also possible that the screen is UNDETERMINED (does not match any defined screen) or UNAVAILABLE (session is not connected).

VB Example:

```
screenName = TP.getCurrentScreen
```

Java Example:

```
String screenName = tp.getCurrentScreen();
```

Obtaining the Base Screen Name

getBaseScreen

This function is used by an application to retrieve the name of the screen that defines the identifiers for the screen currently being displayed.

The screen name returned is determined by comparing the screen identifiers of the screen currently displayed to the identifiers of defined screens in the application, in the following order:

1. Base Screens in the current transaction.
2. Base Screens in other transactions that belong to the application.
3. Screens in the current transaction.
4. Screens in other transactions that belong to the application.

It is also possible that the screen name returned will be UNDETERMINED, if it does not match any defined base screen or transaction screen contained in the application.

If the session is not connected, the screen name returned will be UNAVAILABLE.

For transactions created with the Cleo Transaction Designer, prior to Version 3, when there were no "base" screens, the screen name returned will always be a "NULL". Refer to the Transaction Designer User's Guide for more information on "base" screens.

VB Example:

```
screenName = TP.getBaseScreen
```

Java Example:

```
String screenName = tp.getBaseScreen();
```

Obtaining Session Id

getSessionId

This function returns the Session Id of the currently-reserved session. If there is no currently-reserved session, it returns a zero. This may be useful for debugging purposes.

VB Example:

```
sessId = TP.getSessionId
```

The variable *sessId* is defined as an integer.

Java Example:

```
int sessionId = tp.getSessionId();
```

Turning off Java Garbage Collection

CleoGCoff(gcOFFON)

Where *gcOFFON* is a Boolean argument. This function, when called with a value of **TRUE** for *gcOFFON*, results in Java Garbage Collection **NOT** being done during a following *release()* API Request. When called with a value of **FALSE** for *gcOFFON*, the result is that Java Garbage Collection is done during a following *release()* API Request.

A *release()* API Request does do Java Garbage Collection as a default, unless overridden by a preceding *CleoGCoff(TRUE)* API Request.

Java Example:

```
int rc = tp.CleoGCoff(gcOFFON);
```

Upon execution of the function, the Transaction Processor returns an integer return code, which corresponds to one of the following:

- Completed with success (0)
- Failed with lost socket connection to TP Client (-100)
- Failed no previous reserve API request completed (-102)

Freeing up a Host Session

release()
release(disconnect)

This function is used to return a reserved session to the session pool. The TP checks if the last screen received from the host for the session is the parked screen, and if not, initiates the repark transaction. The repark transaction is the name of the last transaction run, appended with **_park**. If the repark transaction does not exist, and the session is not at the parked screen, the TP initiates **Recovery**. This process ensures that the session will be at the **parked** screen before it is returned to the session pool. Refer to the section Repark Transaction Development for more information.

Depending upon the nature of the application and the size of the installation, there are two types of releases that may be used.

Release() - returns the session to the session pool and disconnects the socket connection between the application and the Transaction Processor.

NOTE: This type of release insures that the socket connection cannot be used by external applications to circumvent security. However, by destroying the socket, the application is increasing the Transaction Processor overhead for both reserve (i.e. create a socket) and release (i.e. close the socket).

release(disconnect) – if disconnect is set to false, this function returns the session to the session pool and maintains the socket for subsequent use by the same application. Otherwise, the behavior is the same as the release without the disconnect parameter.

Upon execution of the function, the Transaction Processor returns an integer return code, which corresponds to one of the following:

- Completed with success (0)
- Completed with system error (-108)

Refer to the **Return Code Chart** on page 71 for details on return codes.

VB Example:

```
returnCode = TP.release
returnCode = TP.release(false)
```

Java Example:

```
int rc = tp.release( )
int rc = tp.release(false)
```

NOTE: Setting the Environment Variable *CLEOGC_OFF=off or =OFF*, will cause the *release()* API Request to NOT do Java Garbage Collection before returning. Doing Java Garbage

Collection before returning is the default. Or if `CleoGCoff(true)` precedes the `release()` API request, then Java Garbage Collection will NOT be done.

Production Run

In a production environment, a user application makes a request for information from the mainframe host. It is the user application's responsibility to execute the correct sequence of Transaction Processor function calls to retrieve the requested information. A typical flow diagram follows.

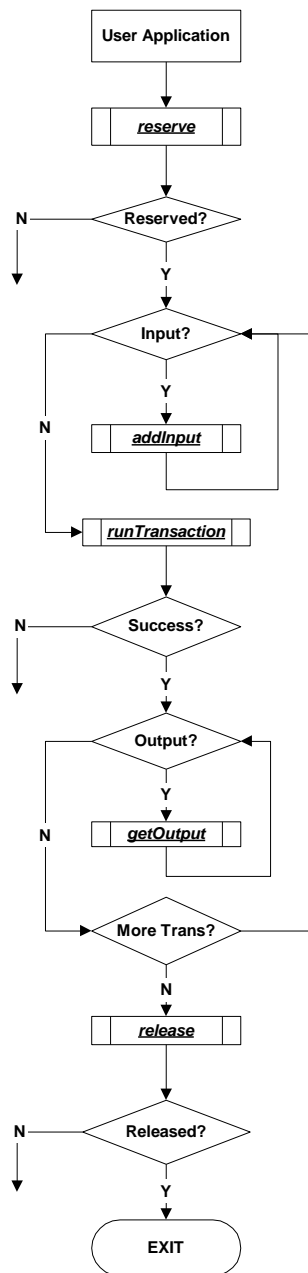


Figure 2: Cleo TP Function Call Sequence

This illustrates the Transaction Processor functions that are invoked when running a transaction to exchange information to and from a host application, assuming that a pool of sessions has been previously started. First the Transaction Processor *reserve* function is called to reserve an available session mapped to a particular transaction set. Then, if there is any input to be sent to the host when running this transaction, it is set by calling the Transaction Processor *addInput* function for each input field value to be set. The Transaction Processor *runTransaction* is called to run the specified transaction, such as when retrieving an account balance. If this is successful, then the Transaction Processor *getOutput* is called for each output field with returned host data. Note that all transactions, input, and output field names must have been defined previously using the TD.

The user application (e.g., an IVR) can run more transactions if desired. In particular, a **repark** transaction may be necessary to return the session screen to a **parked** state, ready for use by another IVR application.

Once all transactions have been run, the Transaction Processor *release* function is called to release the session and make it available to other user applications.

It is up to the user application to decide how to handle exception conditions, which are indicated by the dangling arrows in the flow diagram.

Processing and Application Considerations

This section identifies topics for consideration when setting up your user application:

- Service start up and shutdown considerations using the Web Admin
- Interactive response application considerations

Start Up/Shutdown Processing Considerations

The Transaction Processor Web Admin utility consists of a set of administration and configuration web pages, hereafter referred to as the Web Admin. The Web Admin provides a way for the user to manage and configure the Transaction Processor as described in the *Administration Guide*.

Transaction Processor Startup Processing

To access the Web Admin:

1. If on UNIX/Linux, the tomcat web server must be running. If it is not, issue the following command from the root user: **“tpadmin start”**
2. On Windows, check that the Transaction Processor Admin service is running using Services under Administrative Tools. If not, start it.
3. Enter the following URL into the address line of your Internet browser:
“http://<ipaddress>:38080” where <ipaddress> is the host name or IP address of the system where the Transaction Processor service is running.

The **“Start Service”** and **“Stop Service”** buttons on the administration page are used to start and stop the Transaction Processor. Once the Transaction Processor Service has started, it runs the **Login** and optional **Park** transactions for all configured TN host sessions.

To start the Transaction Processor Service on a UNIX/Linux system, from the root user type **“tpservice start”** at the UNIX/Linux prompt.

Whenever configuration changes are made to assign/unassign transaction sets to sessions, be sure to stop and then restart the Cleo Transaction Processor Service for those changes to take place.

Transaction Processor Shutdown Processing

To shutdown the Transaction Processor Service, simply click the **“Stop Service”** button on the Web Admin page. On a UNIX/Linux system, the Transaction Processor can also be stopped by typing **“tpservice stop”** from the root user.

The **Logout** transaction is run for all sessions assigned to a 3270/5250 transaction set during shutdown.

Interactive Response Application Considerations

For Interactive Voice Response (IVR) applications, it is generally recommended to answer the call with a brief greeting, and then use the *reserve* function to determine if the host connection is functioning properly (return code of 0) or not (return code of non-zero). If *reserve* is called before answering the phone, the caller could experience multiple rings before the phone is answered, since the *reserve* function can take several seconds to return.

The following are additional suggestions on application handling:

1. Voice Application Exit on Call Completion

The voice application needs to do a *release* function call before exiting, so that the session it was using can be used for another caller.

2. Voice Application Hang-up Handling

If a session has been reserved, it must be released before exiting the voice application.

3. Voice Application Multiple Similar Screen Handling

A common occurrence with host applications is to receive a sequence of similar screens (first screen, middle screens, last screen), where the voice application needs to retrieve the **From Host (Output)** field values from one or more of the returned screens. Recommendation: Use the branching capability described in the *Cleo Transaction Designer User's Guide*. A transaction can be defined to run a subsequent transaction based on the last screen returned from running the previous transaction.

Considerations for 3270/5250 Transaction Development

When developing 3270/5250 transactions to be used by the application, remember that the last screen in the transaction must match the first screen in any subsequent transaction to be run.

Repark Transaction Development

When creating 3270/5250 transactions using the TD, the transaction that will be used when the processing of a customer call is ended requires consideration. The transaction can either:

- include logic within the transaction to get the session back to the **parked** state, or
- be accompanied by a companion transaction, **<transaction_name>_park**.

Consider:

The **<transaction_name>_park** transaction will need to navigate screens to get back to the **parked** state screen. The **<transaction_name>__park** transaction is automatically run by the Transaction Processor when the *release* function is processed; this assumes that the **LAST** transaction processed before doing the *release* was **<transaction_name>**.

The Cleo Transaction Processor only runs one **<transaction_name>_park** transaction automatically when processing a *release* function.

There are two advantages to using a companion **<transaction_name>_park** transaction:

1. The Cleo Transaction Processor can be processing the **<transaction_name>_park** transaction to get the session back to a **parked** state, at the same time the voice application is gracefully exiting.
2. A voice application doesn't have to have extra logic to re-park the session, in case the caller chooses to make an early exit from a call.

NOTE: *If a transaction contains a branch or rule resulting in one or more subsequent transactions to be run, there must be a companion **_park** transaction for each of the possible transactions that could be run. The main **<transaction_name>__park** would never be run in this situation.*

.....

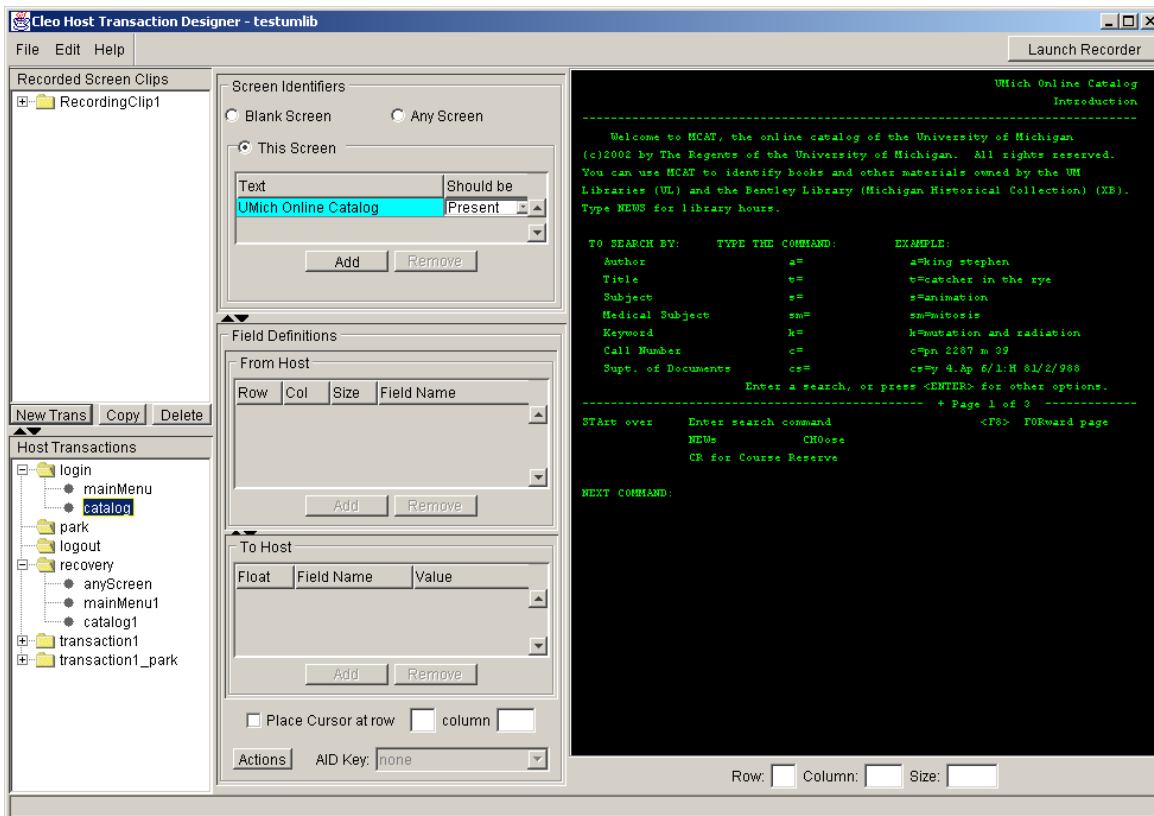
Recovery Transaction Development

Use of the recovery feature in the Cleo Transaction Processor, though strongly recommended, is optional. If a **recovery** transaction is not defined, the transaction processor will, by default, use Power Off followed by Power On session commands to attempt to recover the session and get it back to a **parked** state. Sending Power Off followed by Power On tells the host that the terminal session has been turned off, and then turned back on. This usually causes the initial host (i.e. banner) screen to be sent to the session. When the Transaction Processor detects this screen has been received, it runs the **login** transaction to put the session into the **parked** state.

It is important to point out that the automatic recovery is not guaranteed to work in every environment. Thus, we recommend that this feature be tested during application development to determine if it will operate in the desired manner, or if a user-defined recovery transaction is even required.

The following TD screen shot shows a transaction set that used a public host. The public host is NO LONGER accessible. The TD screen shot is shown for explanation purposes only.

The screen selected (named “catalog”) is the **parked** screen.



Alternatively, consider implementing a user-defined **recovery** transaction, which the Transaction Processor would automatically run when the session is put into the **recovery** state due to an error. The first screen of a **recovery** transaction could be defined in the TD as an "Any" screen, meaning that the Transaction Processor does not check the contents of the screen before running the transaction. Depending on the host application, an AID key, such as PA1, can be specified to cause the host to navigate to a known screen. This screen would be defined in the TD as the next screen of the **recovery** transaction. Then, navigation to the **parked** screen must occur, as in the **login** transaction. Once the recovery transaction has been successfully run, the session is ready for use by an IVR call.

The following TD screen shot shows a transaction set that used a public host. The public host is NO LONGER accessible. The TD screen shot is shown for explanation purposes only.

In the TD screen shot, the method used to return to the initial screen of the public host (“mainMenu”) is to enter “sta” at the “NEXT COMMAND” prompt. This is shown as a **ToHost**, or input field in the TD screen shot (following) for the 1st screen of the **recovery** transaction. The AID key used in this scenario is the “Enter” key. Also note that the “Any Screen” radio button has been selected in the “Screen Identifiers” panel to indicate to the Transaction Processor that no checking of the screen contents should be done. The last two screens in the **recovery** transaction are the same as those in the **login** transaction. In effect, this part of the recovery transaction is to login again, necessary to return the session to the **parked** state.

The screenshot displays the Cleo Host Transaction Designer interface. The left pane shows a tree view of Host Transactions, with the 'recovery' transaction selected. The 'Screen Identifiers' panel has 'Any Screen' selected. The 'Field Definitions' panel shows a 'To Host' field named 'command' with the value 'sta'. The 'Actions' panel shows 'AID Key' set to 'ENTER'. The right pane shows a terminal window with search results for 'K=IRAQ' from the 'UMich Online Catalog'. The terminal output includes a list of search results with columns for DATE, TITLE, and AUTHOR, and a 'NEXT COMMAND:' prompt highlighted in blue.

Row	Col	Size	Field Name

Float	Field Name	Value
	command	sta

```

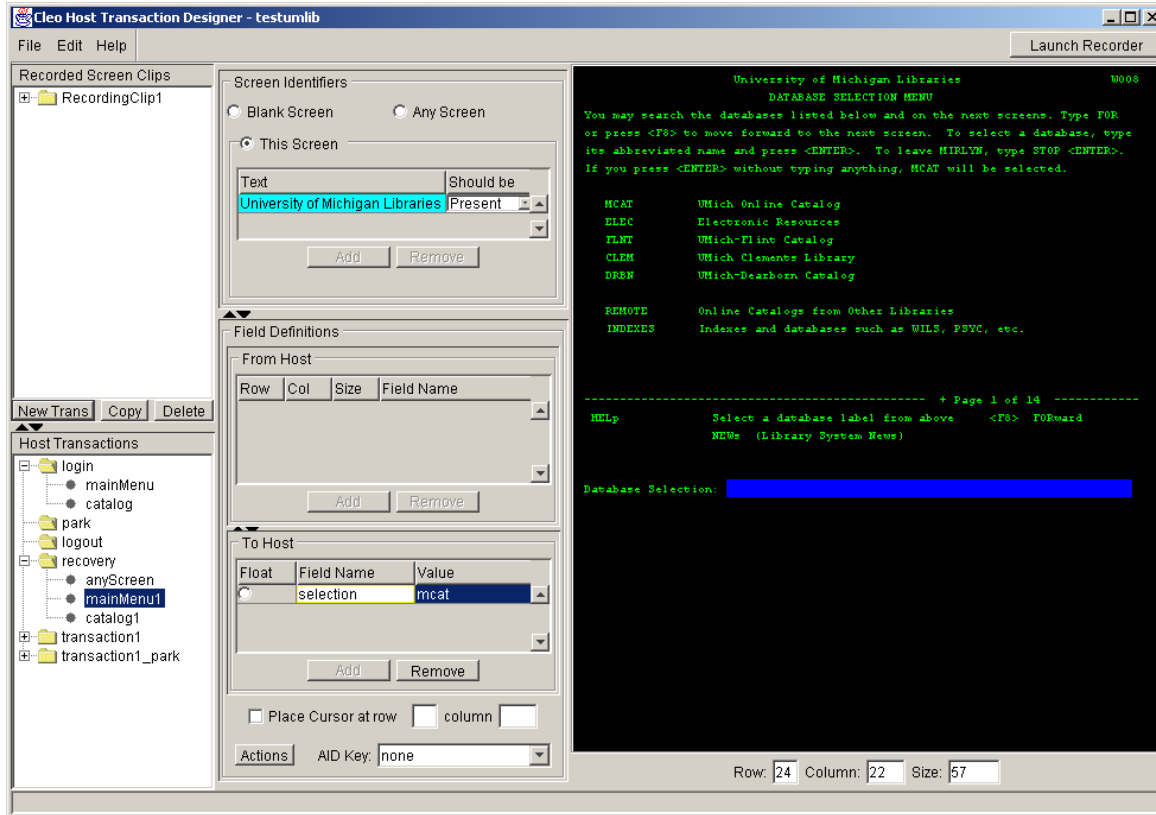
Search Request: K=IRAQ                                UMich Online Catalog
Search Results: 2987 Entries Found                    Keyword Index
-----
DATE  TITLE:                                           AUTHOR:
1 2003 AUTHORIZATION FOR USE OF MILITARY FORCE AG      UL
2 2003 The Bushayhid dynasty in Iraq 204 H./245 to   Donohue, John J  UL
3 2003 Central and Southern Iraq, Scale 1:1 <map>    UL
4 2003 Central and Southern Iraq, Scale 1:1 <map>    UL
5 2003 Eighth-century Iraqi grammar : a critical    Talmon, Rafael  UL
6 2003 Guiding principles for U.S. post-conflict    UL
7 2003 Iraq: Country Profile, January 2003 <map>    UL
8 2003 Iraq : issues, historical background, bibl   UL
9 2003 Iraq under siege : the deadly impact of wa   UL
10 2003 Iraq, 802749AI (C00819), January 200 <map>  UL
11 2003 PERIODIC REPORT ON THE NATIONAL EMERGENCY    UL
12 2003 Reconstructing Iraq : insights, challenges  Crane, Conrad C  UL
13 2003 Report On The Continuation Of The National  UL
14 2003 The United Nations and Iraq : defanging th  Krasno, Jean E., 1 UL
-----
START over      Type number to display record      <F6> Forward page
HELP           CHOOSE
OTHER options

NEXT COMMAND:

```

Row: 24 Column: 16 Size: 63

The screen named “mainMenu” follows.



Defining Multiple Recovery Transactions

A single **recovery** transaction is sufficient in cases where a single action or series of actions can be performed from any screen in order to return the session to a **parked** state.

However, in some cases it may be necessary to provide a more flexible means of recovery by which a different recovery transaction is executed depending on the current screen. Let us assume we have 3 different screens, each requiring a different sequence of events to reach the parked screen.

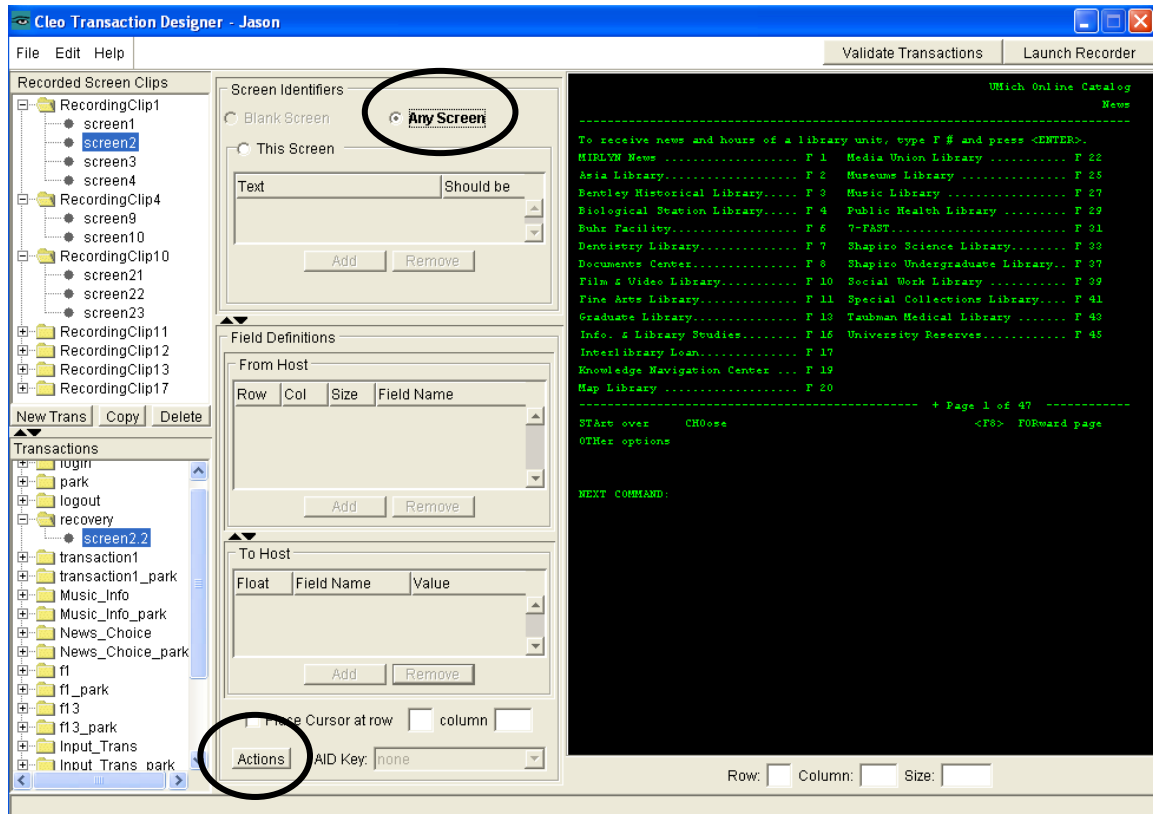
Screen A requires sequence A

Screen B requires sequence B

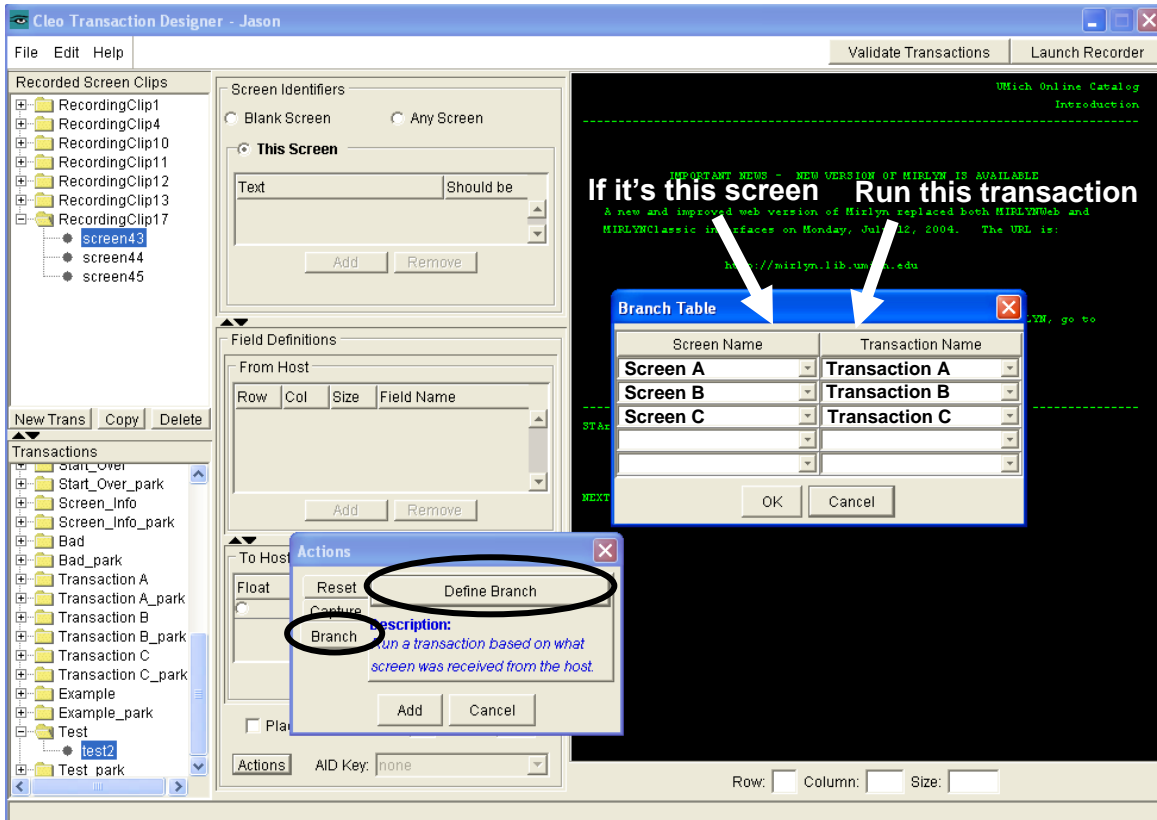
Screen C requires sequence C

In order define a recovery transaction to handle this situation, take the following steps.

1. Make the first screen of the recovery transaction an **any** screen and then click *Actions*.



2. Click on "Branch" and then "Define Branch" to open the *Branch Table* dialog. In the left column put the screen names, and in the right column put the transaction names to run. In our case we will put Screen A, Screen B, and Screen C in the *Screen Names* column and Transaction A, Transaction B, and Transaction C in the *Transaction Names* column. If Screen A is encountered, Transaction A will be run, and so on. You can find additional information on Branching in the *Transaction Designer User's Guide*.



Alternative Access to the TP

Besides Java and .NET, several alternative connectors (or interfaces) for accessing the TP are included. They are:

- VoiceXML Connector
- XML Connector
- Web Service Connector

Depending on what environment the application will be run in, one or more connectors may be required.

Using the VoiceXML Connector

Overview

The VoiceXML Connector is a simple JSP interface that provides a mechanism for VoiceXML applications to interact with the Transaction Processor through the use of subdialog calls.

It is installed with any TP Service or TP Client installation. See the appropriate Quick Start Guide for installation and setup instructions.

Getting Started With The VoiceXML Connector

The VoiceXML Connector is a precompiled JSP application that runs under a different application context in the same Tomcat instance as the TP Web Admin. Therefore, use of the VoiceXML Connector requires that the Transaction Processor Admin Service is running. The VoiceXML Connector can be accessed from any VoiceXML application that supports subdialogs.

Before invoking the VoiceXML Connector, declare the following required application variables in the VoiceXML application

```
<?xml version = "1.0"?>
<vxml version = "2.0">
...
  <var name = "sessionName" expr = "'session_name'"/>
  <var name = "transSetName" expr = "'trans_set_name'"/>
  <var name = "featureName"/>
  <var name = "status"/>
```

(where *session_name* is any unique name that will identify the caller to the Transaction Processor. By providing a session name, the associated TP session (obtained by a reserve) is guaranteed to be used for each subsequent call into the VoiceXML Connector.

and

trans_set_name is the name of the transaction set that has been created by the Transaction Designer and transferred into the Transaction Processor %TP_HOME%\trans directory that contains the transactions to be run)

as well as any form variables that may be assigned values (such as pool, returnCode, sessionId, currentScreen, sessionStatus, transaction, and any *To-Host* and *From-Host* fields as defined by the Transaction Designer).

Also, 2 request parameters are available that can be used to request the Cleo TP Server's host IP and/or port used.

```
gethostip
getport
```

By setting either or both of these request parameters to a value of 'y', will result in the Cleo TP Server's host IP and/or port used being returned in the subdialog return data.

When either or both of these request parameters are specified, and set to a value of 'n', then no data is returned in the subdialog return data.

```
<form>
...
  <var name = "pool"/>
  <var name = "returnCode"/>
  <var name = "sessionId"/>
  <var name = "currentScreen"/>
  <var name = "transaction"/>
  <var name = "sessionStatus"/>
  <var name = "gethostip"/>
  <var name = "getport"/>
  <var name = "to-host-field_1"/>
  ...
  <var name = "to-host-field_n"/>
  <var name = "from-host-field_1"/>
  ...
  <var name = "from-host-field_n"/>
```

The VoiceXML application can then invoke any of the functions provided by the VoiceXML Connector. The VoiceXML Connector provides a subset of the standard Java API Connector functions. They are invoked through individual subdialogs with the following basic structure,

```
<block><assign name = "featureName" expr = "' feature_name'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"feature_name parameter_name_1 ... parameter_name_n">
```

```
<filled>
  <assign name = "status" expr = "subdialog_name.status"/>
  <assign name = "variable_name_1" expr =
"subdialog_name.variable_name_1"/>
  ...
  <assign name = "variable_name_n" expr =
"subdialog_name.variable_name_n"/>
</filled>
</subdialog>
```

where

feature_name is one of the following:

- reserve
- getSessionId
- getSessionStatus
- getCurrentScreen
- runTransaction
- release

subdialog_name is any unique name describing the subdialog. It is recommended that the name meaningfully describes the action that the subdialog will perform (ex. reserveSample or runSearchTransaction).

vxml_connector_url is the URL of the VoiceXML Connector:

- **http://<remote-server>:38080/vxml/VXMLConnector** if the VoiceXML Connector is running on a different system than the VoiceXML application
- **http://localhost:38080/vxml/VXMLConnector** if running on the same system that the VoiceXML application is running on.
- **VXMLConnector** if the VoiceXML application is located in the same context as the VoiceXML Connector.

parameter_name_1 ... parameter_name_n are the required and optional parameters sent to the VoiceXML Connector for processing.

variable_name_1 ... variable_name_n are the variables/values returned from the VoiceXML Connector when the request has completed.

Functions Provided

reserve

Option 1 – standard connection

```
<block><assign name = "featureName" expr = "'reserve'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName transSetName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "returnCode" expr = "subdialog_name.returnCode"/>
  </filled>
</subdialog>
```

Option 2 – pooled connection

```
<block><assign name = "featureName" expr = "'reserve'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName pool transSetName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "returnCode" expr = "subdialog_name.returnCode"/>
  </filled>
</subdialog>
```

getSessionId

```
<block><assign name = "featureName" expr = "'getSessionId'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "sessionId" expr = "subdialog_name.sessionId"/>
  </filled>
</subdialog>
```

getSessionStatus

```
<block><assign name = "featureName" expr = "'getSessionStatus'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "sessionStatus" expr =
"subdialog_name.sessionStatus"/>
```

```

    </filled>
</subdialog>

```

getCurrentScreen

```

<block><assign name = "featureName" expr = "'getCurrentScreen'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "currentScreen" expr =
"subdialog_name.currentScreen"/>
  </filled>
</subdialog>

```

runTransaction

```

<block><assign name = "featureName" expr = "'runTransaction'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName transaction [to_host-field_1 ...
to_host-field_n]">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "returnCode" expr = "subdialog_name.returnCode"/>
    [<assign name = "from-host-field_1" expr = "subdialog_name.from-
host-field_1"/>
    ...
    <assign name = "from-host-field_n" expr = "subdialog_name.from-
host-field_n"/>]
  </filled>
</subdialog>

```

NOTE: *to_host_field_1* to *to_host_field_n* and *from-host-field_1* to *from-host-field_n* are optional. They should match the names of the form variables that were described in the Getting Started With The VoiceXML Connector section.

release

```

<block><assign name = "featureName" expr = "'release'"/></block>
<subdialog name = "subdialog_name" src = "vxml_connector_url" namelist =
"featureName sessionName">
  <filled>
    <assign name = "status" expr = "subdialog_name.status"/>
    <assign name = "returnCode" expr = "subdialog_name.returnCode"/>
  </filled>
</subdialog>

```

Using The XML Connector

Overview

The XML Connector for Transaction Processor provides access to a subset of the Transaction Processor API functions (reserve, addInput, runTransaction, getOutput, release) from any application that can employ XML and HTTP, including a custom built application or the Vonetix Document Plug-in. A series of API calls can be executed through a single HTTP request by posting a specialized XML document to the XML Connector. This specialized XML document contains the functions to invoke and the associated arguments to each. Depending on what functions were defined in the XML document sent to the XML Connector, the XML Connector will return another specialized XML document that contains the values for the requested output along with any return codes. The XML document that is returned can then be parsed using any XML parser including, but not limited to, JDOM.

It is installed with any TP Service or TP Client installation. See the appropriate Quick Start Guide for installation and setup instructions.

Getting Started with the XML Connector

Before an application can access the XML Connector, the Transaction Processor and XML Connector must be running. By default, the XML Connector is installed with the Transaction Processor in the same Tomcat instance as the Transaction Processor Admin, but under a different context. It will be started every time the Transaction Processor Admin is started.

Invoking the XML Connector

Although the XML Connector is written in Java, any application that employs HTTP and XML can use the XML Connector.

The XML Connector can be invoked by posting a specially formed XML document over HTTP to the URL where the XML Connector is running. This XML document (request) contains the information required by the XML Connector to invoke the appropriate TP functions. The request document can either be read in from the file system or built “on-the-fly”. However, the XML Connector will NOT run if sent a request document that does not follow the specified format (see below).

Request Document

NOTE: The functions that are defined in the request document will be invoked in the order they appear.

The basic format of the XML document that is recognized by the XML Connector is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<calls>
  [<init id="initID" sessname="sessionName" host="ipAddress"
port="portNumber"/> |
  <init id="initID" pool="poolName"/>]
  <reserve id="reserveID" sessname="sessionName"
transSet="transSetName"/>
  <addInput id="addInputID" sessname="sessionName"
inputName="inputName" inputValue="inputValue"/>
  <runTransaction id="runTransID" sessname="sessionName"
transName="transactionName"/>
  <getOutput id="getOutputID" sessname="sessionName"
outputName="outputName"/>
  ...
  <release id="releaseID" sessname="sessionName"/>
</calls>
```

where,

- **calls** (root element) encloses all the functions to be invoked and tells the XML Connector that a series of function “calls” are being requested.
- **init** (element) tells the XML Connector to invoke the constructor (i.e. prepare the XML Connector to make calls into the TP).
 - **id** (attribute) a unique identifier used to associate *return codes* with the particular init call.
 - **sessname** (attribute) associates any subsequent calls using the same sessname with the same TP connection.
 - **host** (attribute) the ip address or host name of the system running the TP Service that will be connected to.
 - **port** (attribute) port number of the TP Service (default 7561).
- **reserve** (element) tells the XML Connector to invoke the “reserve” function on the TP.
 - **id** (attribute) a unique identifier used to associate *return codes* with the particular “reserve” call.
 - **sessname** (attribute) associates any subsequent calls using the same sessname with the same TP session.

- **transSet** (attribute) tells the XML Connector which transaction set to reserve a session against.
- **addInput** (element) tells the XML Connector to invoke the “addInput” function on the TP.
 - **id** (attribute) a unique identifier used to associate *return codes* with the particular “addInput” call. (**NOTE:** Currently there are no return codes associated with addInput. This attribute is required solely for consistency and for potential future enhancements.)
 - **sessname** (attribute) ensures that the “addInput” will be made against the same TP session reserved by a previous “reserve” call using the same sessname.
 - **inputName** (attribute) name of the input field (as defined by the TD) that will be overwritten by the provided user input.
 - **inputValue** (attribute) user input used to overwrite the default value (as defined by the TD) for the given input field.
- **runTransaction** (element) tells the XML Connector to invoke the “runTransaction” function on the TP.
 - **id** (attribute) a unique identifier used to associate *return codes* with the particular “runTransaction” call.
 - **sessname** (attribute) ensures that the “runTransaction” will be made against the same TP session reserved by a previous “reserve” call using the same sessname.
 - **transName** (attribute) name of the transaction to run.
- **getOutput** (element) tells the XML Connector to invoke the “getOutput” function on the TP.
 - **id** (attribute) a unique identifier used to associate *values* with the particular “getOutput” call.
 - **sessname** (attribute) ensures that the “getOutput” will be made against the same TP session reserved by a previous “reserve” call using the same sessname.
 - **outputName** (attribute) name of the output field (as defined by the TD) to retrieve data for.
- **release** (element) tells the XML Connector to invoke the “release” function on the TP.
 - **id** (attribute) a unique identifier used to associate *return codes* with the particular “release” call.
 - **sessname** (attribute) ensures that the same TP session reserved by a previous “reserve” call using the same sessname will be released.

Example Request

In the following example,

1. The connection to the TP running at IP address 10.1.2.19 and port 7561 is established.
2. A session configured against the (TN) "sample" transaction set is reserved.
3. The "keyword" input field is given a value of "cleo".
4. The "getlist" transaction is run.
5. The value for "list1" output field is requested.
6. The session reserved in (2) is released.

```
<?xml version="1.0" encoding="UTF-8"?>
<calls>
  <init id="initTP" sessname="testsess" host="10.1.2.19"
port="7561"/>
  <reserve id="reserve_sample" sessname="testsess"
transSet="sample"/>
  <addInput id="add_request" sessname="testsess"
inputName="keyword" inputValue="cleo"/>
  <runTransaction id="runTrans_getlist" sessname="testsess"
transName="getlist"/>
  <getOutput id="get_list1" sessname="testsess"
outputName="list1"/>
  <getOutput id="get_list2" sessname="testsess"
outputName="list2"/>
  <release id="release_sample" sessname="testsess"/>
</calls>
```

Processing Return Data

Once the XML Connector has finished processing the incoming request document and invoking the requested TP functions, it will return a specially formed XML document back to the calling application. This XML document (response) contains any associated return codes and/or values for each of the invoked TP functions.

Errors that occur during processing will be reflected in an appropriate error code being returned and are detailed in the Error Handling section. It is the responsibility of the calling application to properly handle any of these error conditions.

Response Document

The basic format of the XML document returned by the XML Connector is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<returns>
  <id_RC>0</id_RC>
  ...
  <id_value>ADMIN - CICSPRD1</id_value>
  ...
</returns>
```

where,

- **returns** (root element) encloses the return codes and values for each function that was invoked by the XML Connector.
- **id_RC** (element) value is the return code associated with the call with the given id (as defined in the request document).
- **id_value** (element) value is the return value associated with the call with the given id (as defined in the request document).

Example Response

The following response results from the XML Connector processing the Example Request on page 43.

```
<?xml version="1.0" encoding="UTF-8"?>
<returns>
  <initTP_RC>0</initTP_RC>
  <reserve_sample_RC>0</reserve_sample_RC>
  <runTrans_search_RC>0</runTrans_getlist_RC>
  <get_list1_value>    Admin    - CICSPRD1
</get_list1_value>
  <get_list2_value>    NERCICS  - CICSPRD1
</get_list2_value>
  <release_sample_RC>0</release_sample_RC>
</returns>
```

In this case,

1. The connection to the TP was successfully established (“0”).
2. The call to “reserve” was successful (“0”).
3. The call to “runTransaction” was successful (“0”).
4. The value for “list1” came back as “Admin - CICSPRD1”.
5. The value for “list2” came back as “NERCICS - CICSPRD1”.
6. The call to “release” was successful (“0”).

Using the XML Connector with Avaya IR and Vonetix

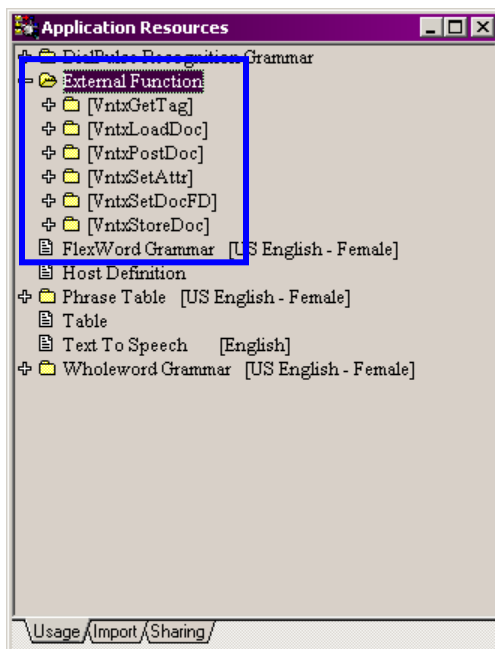
This section details the steps necessary for integrating the XML Connector with Avaya IR through Vonetix and is a supplement to the previous section, Using The XML Connector. It may be safely skipped for those not interested in Avaya IR or Vonetix.

NOTE: This section is not intended to be a general purpose developer's guide for either Avaya IR or Vonetix. The reader is assumed to have a working knowledge of application development using Avaya IVR Designer. Therefore, this section will only cover the minimal information required for the integration. (Please refer to the respective Avaya or Vonetix documentation for any details that are not covered in this section.)

Getting Started with the XML Connector, Avaya IR, and Vonetix

In order to use the XML Connector with Avaya IR, the Vonetix Document Plug-in must be installed, configured, and running against the Avaya IR system. (Consult the Vonetix Document Plug-in Developer's Guide for installation and setup instructions.) The Vonetix Document Plug-in supports XML over HTTP and is capable of translating Avaya IR specific data into a form (XML) recognized by the XML Connector.

Before the XML Connector can be accessed using IVR Designer, the following Vonetix Document Plug-in external functions must be imported: *VntxGetTag*, *VntxLoadDoc*, *VntxPostDoc*, *VntxSetAttr*, *VntxSetDocFD*, *VntxStoreDoc*.



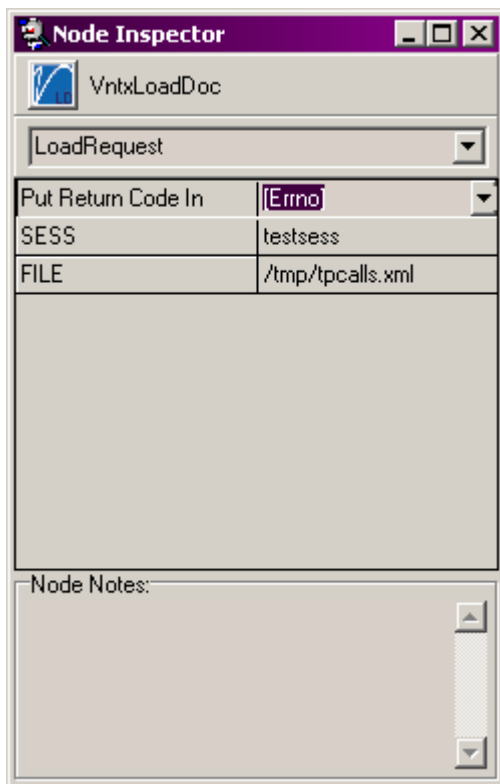
NOTE: Only TAS applications can access the XML Connector through Vonetix. To access the Transaction Processor functions with VXML, use the VoiceXML Connector.

Accessing the XML Connector with IVR Designer

The external functions – *VntxGetTag*, *VntxLoadDoc*, *VntxPostDoc*, *VntxSetAttr*, *VntxSetDocFD*, *VntxStoreDoc* – can be used to access the XML Connector. Their particular usages are detailed in this section.

Load Request Document (*VntxLoadDoc*)

The Request Document can be loaded using the *VntxLoadDoc* external function.

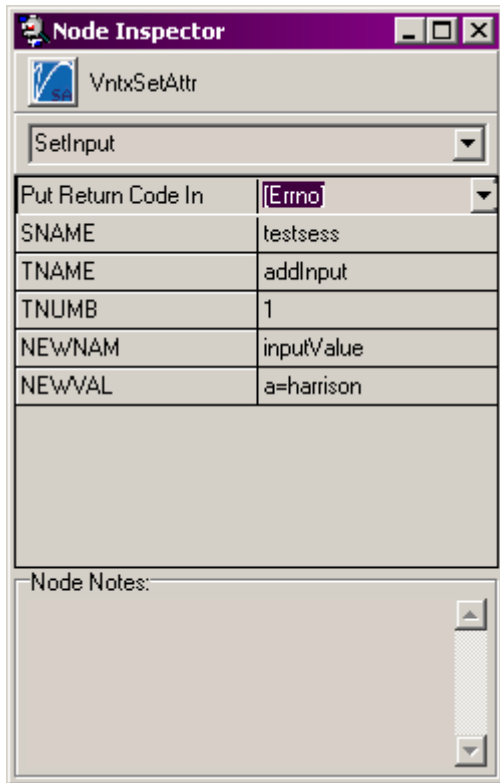


SESS (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

FILE: path to request document file. (This document must follow the same format as described in the Request Document section.)

Set Request Input (*VntxSetAttr*)

Setting an input value requires the use of the *VntxSetAttr* external function.



SNAME (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

TNAME (tag/element name): used to identify an element with a particular name. Must be set to “**addInput**”.

TNUMB (tag/element number): used to identify a particular instance of an element. Depending on how many “addInput” elements are defined, this number will vary.

NEWNAM (attribute name): the name of the attribute whose value will be set. Must be set to “**inputValue**”.

NEWVAL (attribute value): the value that the attribute will be set to. This value will overwrite the value for the specified input field determined by TNUMB.

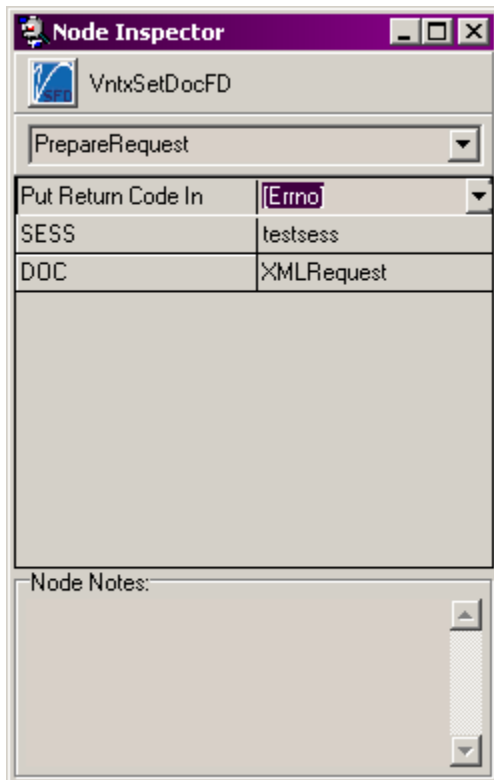
For instance, if there are 3 “addInput” calls defined in the request document:

```
...
<addInput id="add_request1" sessname="testsess" inputName="request"
inputValue="a=jefferson"/>
...
<addInput id="add_request2" sessname="testsess" inputName="option"
inputValue="2"/>
<addInput id="add_request3" sessname="testsess" inputName="menu"
inputValue="luis"/>
...
```

Then, with TNUM set to “2”, the external function call will set the value for the input field with name “option”.

Prepare Request For Sending (VntxSetDocFD)

In order to send the request document over HTTP, it first must be converted into form data. The *VntxSetDocFD* external function is used for this conversion process.

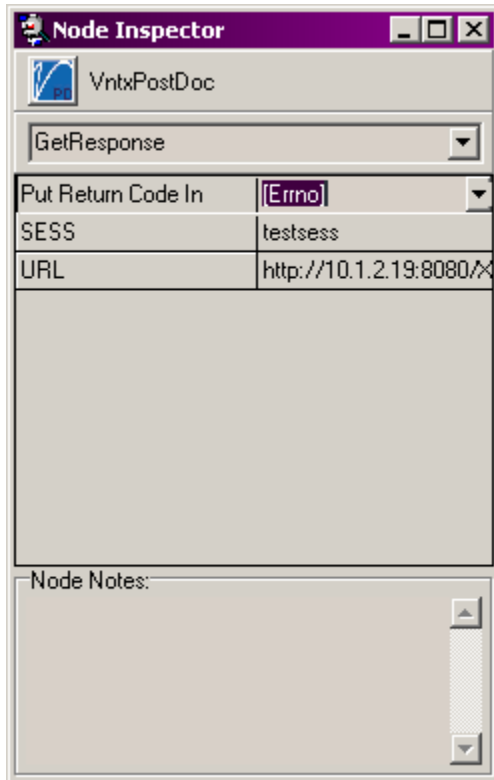


SESS (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

DOC: HTTP request parameter name for the loaded document. Must be “**XMLRequest**”.

Get Response Document (VntxPostDoc)

The *VntxPostDoc* external function can be used to send the request document to the XML Connector and obtain the resulting Response Document.

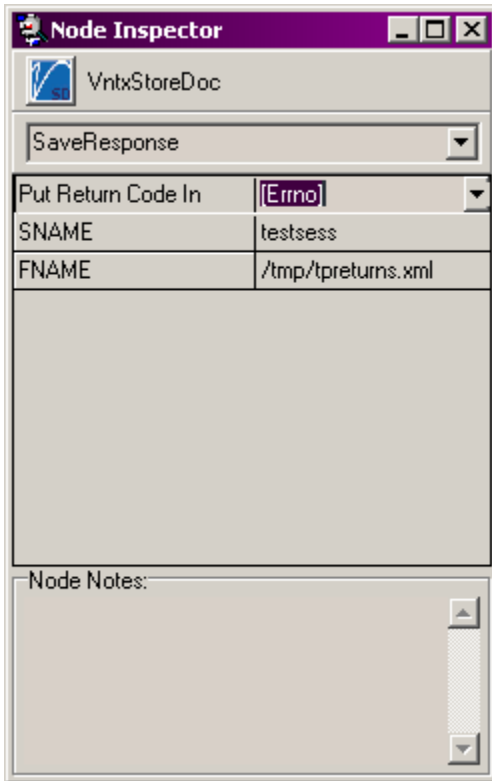


SESS (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

URL: the URL of the web application to post the loaded document to. In this case, the URL will be that of the XML Connector (e.g. <http://localhost:38080/xml/XMLConnector>).

Save Response Document (*VntxStoreDoc*)

There may be instances, particularly during debugging, when the response document that is returned from the XML Connector needs to be stored to the file system so that it can be viewed at a later time. This action can be accomplished using the *VntxStoreDoc* external function.

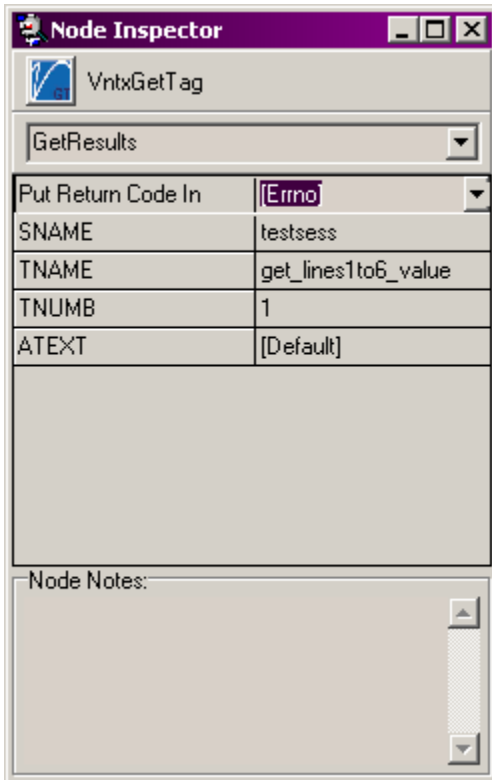


SNAME (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

FNAME (file name): path on the file system to store the document.

Get Results (*VntxGetTag*)

Once the response document is returned, return code and output values can be used in application logic and/or presented to the caller. To isolate the relevant information from the response document, the *VntxGetTag* external function can be used.



SNAME (session name): used by Vonetix to associate the same HTTP session with each call. (**Caution:** This session name is not the same as the session name required in the request document described starting on page 41.)

TNAME (tag/element name): used to identify an element with a particular name. For return codes, set TNAME to *id_RC*. For values, set TNAME to *id_value*. For both, *id* refers to the unique id required in the request document described starting on page 41.

TNUMB (tag/element number): used to identify a particular instance of an element. Normally, this number could vary. However, in this case, since the TNAME is guaranteed to be unique (due to *id* being unique), there will be only one instance of each element. Therefore, TNUMB must be set to “1”.

ATEXT (variable): name of the variable to store the results. This variable can then be used in subsequent prompts and or conditions.

Using the Web Service Connector

Overview

The Web Service Connector for Transaction Processor provides access to any of the Transaction Processor API functions (described in Using the Transaction Processor Functions) from any application that can employ Web Services (using SOAP and WSDL).

NOTE: Familiarity with SOAP, HTTP, and WSDL is assumed, as this section will not go into details on those areas.

The Web Service Connector is installed with any TP Service or TP Client installation. See the appropriate Quick Start Guide for installation and setup instructions.

Getting Started with the Web Service Connector

Before an application can access the Web Service Connector, the Transaction Processor and Web Service Connector must be running. By default, the Web Service Connector is installed with the Transaction Processor in the same Tomcat instance as the Transaction Processor Admin, but under a different context. It will be started every time the Transaction Processor Admin is started.

Invoking the Web Service Connector

The WSDL is available at <http://<ipaddress>:38080/ws/services/WSCconnector?wsdl>, where <ipaddress> is the host name or IP address of the system where the Transaction Processor Admin service is running. Any standard SOAP interface can be used to invoke the necessary TP functions based on the endpoints and parameters defined by the WSDL.

Code Examples

This section provides sample code for the following programming languages:

- Visual Basic .NET, including:
 - running a simple transaction
 - running the sample transactions
-

- Java
 - using the TP Connector
 - using the XML Connector
- Perl
 - using the Web Service Connector
- VoiceXML
 - using the VoiceXML Connector

Visual Basic .NET

The following code samples assume that the Transaction Processor has been previously started, and that the TPConnector library has been imported into the developer's .NET environment. See the section, **The TP .NET Component**, in the *QuickStart Guide for Windows* for instructions.

Running a Simple Transaction

```
' GetList button - runs getlist in sample application
Dim runner As New TPConnector("10.1.2.10", "7561")

If (IsNothing(runner)) Then
    TextBox1.Text = TextBox1.Text & "Unable to create cleoTP object" & vbCrLf
    Return
End If
rc = runner.reserve("sample")
If (rc = 0) Then
    TextBox1.Text = TextBox1.Text & "Reserve successful to session " & runner.getSessionId & vbCrLf
    If (rc = 0) Then
        rc = runner.runTransaction("getlist")
        If (rc = 0) Then
            TextBox1.Text = TextBox1.Text & "System 1 is: " & runner.getOutput("list1") & vbCrLf
            TextBox1.Text = TextBox1.Text & "System 2 is:" & runner.getOutput("list2") & vbCrLf
        End If
    End If
    rc = runner.release()
Else
    TextBox1.Text = TextBox1.Text & "Reserve failed with rc = " & rc & vbCrLf
End If
runner = Nothing

End Sub
```

Running the Sample Transactions

The following sample code uses a session connected to a public host. It reserves the host session, accepts input for a keyword from the user, sends a request to get a list of Mainframe System names based on the keyword entered by the user, and retrieves and displays the first two Mainframe System names in the list returned by the Mainframe. Then the session is released.

It is assumed that the sample application has been previously defined using the Transaction Designer (delivered with the Cleo TP product), and contain a transaction named **getlist**. The field named "keyword" was defined to be a toHost, or input field in the **getlist** transaction, and the fields named "list1", and "list2" were defined to be fromHost fields. The fromHost fields "list1" and "list2" are defined on the return screen(named "menu") in the **getlist** transaction.

It is helpful to use the TD to look at these transactions that were delivered in the sample application folder.

```

'Run list of Mainframe System Names
Dim keywordReq As String
Dim i, n As Integer
Dim moreData As Boolean = True
Dim runner As New TPCConnector("10.1.2.10", "7561")

If (IsNothing(runner)) Then
    TextBox1.Text = TextBox1.Text & "Unable to create cleoTP object" & vbCrLf
    Return
End If
rc = runner.reserve("sample")
If (rc = 0) Then
    keywordReq = InputBox("Enter keyword to use for Mainframe System List", "Keyword Request")
    If (keywordReq.Length <= 0) Then
        keywordReq = "cleo" 'keyword default in TD sample application
    End If
    Runner.addInput("keyword", keywordReq)
    rc = runner.runTransaction("getlist")
    If (rc = 0) Then
        TextBox1.Text = TextBox1.Text & runner.getOutput("list1 ") & vbCrLf
        TextBox1.Text = TextBox1.Text & runner.getOutput("list2 ") & vbCrLf
    Else
        TextBox1.Text = TextBox1.Text & "Transaction search failed with rc " & rc & vbCrLf
    End If
    TextBox1.Text = TextBox1.Text & keywordReq & " getlist complete." & vbCrLf
    runner.release()
Else
    TextBox1.Text = TextBox1.Text & "Reserve returned rc = " & rc & vbCrLf
End If
runner = Nothing
End Sub

```

Java

The following sample code connects to a Cleo TP Server located at IP Address: 10.10.66.64, and requests to reserve a host session on the University of Florida's mainframe. Then a request to get a list of Available Mainframe Systems, for the keyword "cleo" is done, and the sample code retrieves and displays the first Mainframe System Name in the list. After the list has been received from the host and displayed, the host session is released to the session pool for use by another application.

Using the TP Connector

```
public static void main(String[] args) {
    int sess = 1;

    if (args.length > 0) {
        sess = Integer.parseInt(args[0]);
    }
    TPConnector runner = new TPConnector();
    int rc;

    rc = runner.reserve("sample");
    if (rc == 0) {
        int session = runner.getSessionId();
        String status = runner.getSessionStatus();
        System.out.println("Session Status: " + status);
        System.out.println("Assigned session " + session + ".");
        runner.addInput("keyword", "cleo");
        System.out.println("Running getlist transaction...");
        rc = runner.runTransaction("getlist");
        if (rc == 0) {
            String screenName = runner.getCurrentScreen();
            System.out.println("Screen Name: " + screenName);
            String output = runner.getOutput("list1");
            System.out.println("Available Mainframe Systems List:\n" + output);
            runner.resetInput();
        } else {
            System.out.println("Run of getlist transaction failed! RC=" + rc);
        }
        System.out.println("Releasing session...");
        rc = runner.release();
        System.out.println("Release RC: " + rc);
    } else {
        System.out.println("Reserve failed! RC=" + rc);
        return;
    }
}
```

Using the XML Connector

```
...
// Create a URL connection to the XML Connector
URL url = new URL("http://localhost:38080/xml/XMLConnector");
URLConnection connection = url.openConnection();
connection.setDoInput(true);

// For this example, the XML will be generated "on the fly". However,
// the XML can also be read in from a file and then modified using
// JDOM, SAX, etc.
StringBuffer xml = new StringBuffer("XMLRequest=");
xml.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
xml.append("<calls>");
xml.append("    <init id=\"initTP\" sessname=\"testsess\" host=\"10.1.2.19\" port=\"7561\"/>");
xml.append("    <reserve id=\"reserve_sample\" sessname=\"testsess\" transSet=\"sample\"/>");
xml.append("    <addInput id=\"add_keyword\" sessname=\"testsess\" inputName=\"keyword\"
inputValue=\"cleo\"/>");
xml.append("    <runTransaction id=\"runTrans_getlist\" sessname=\"testsess\"
transName=\"getlist\"/>");
xml.append("    <getOutput id=\"get_list1\" sessname=\"testsess\" outputName=\"list1\"/>");
xml.append("    <release id=\"release_sample\" sessname=\"testsess\"/>");
xml.append("</calls>");

// Post the XML request
PrintWriter writer = new PrintWriter(connection.getOutputStream());
writer.print(xml);
writer.close();

connection.doSetOutput(true);

// Process response using JDOM
BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(reader);
Element returnsEl = doc.getRootElement();

Element runTransEl = returnsEl.getChild("runTrans_getlist_RC");

// First, check that the call to "runTransaction" was successful
if (runTransEl.getText().equals("0")) {
    // Display value for "list1"
    Element list1El = returnsEl.getChild("get_list1_value");
    System.out.println("List1:\n" + list1El.getText());
}
}
...
```

Perl

Using the Web Service Connector

```
#!/perl -w
use SOAP::Lite;

$sessname = shift;
$keyword_req = shift;

print "Request is $getlist_req\n";

$soap_response = SOAP::Lite
  -> uri('http://ws.cleo.com')
  -> proxy('http://localhost:38080/ws/services/WSCconnector')
  -> reserve(
    SOAP::Data->name("sessname" => $sessname),
    SOAP::Data->name("pool" => "test"),
    SOAP::Data->name("transSetName" => "sample")
  );
$res = $soap_response->result;
print "Reserve returned $res\n";

$soap_response = SOAP::Lite
  -> uri('http://ws.cleo.com')
  -> proxy('http://localhost:38080/ws/services/WSCconnector')
  -> addInput(
    SOAP::Data->name("sessname" => $sessname),
    SOAP::Data->name("name" => "keyword"),
    SOAP::Data->name("value" => $keyword_req)
  );

$soap_response = SOAP::Lite
  -> uri('http://ws.cleo.com')
  -> proxy('http://localhost:38080/ws/services/WSCconnector')
  -> runTransaction(
    SOAP::Data->name("sessname" => $sessname),
    SOAP::Data->name("transName" => "getlist")
  );

$res = $soap_response->result;
print "Run transaction returned $res\n";

$soap_response = SOAP::Lite
  -> uri('http://ws.cleo.com')
  -> proxy('http://localhost:38080/ws/services/WSCconnector')
  -> getOutput(
    SOAP::Data->name("sessname" => $sessname),
    SOAP::Data->name("name" => "list1")
  );

$res = $soap_response->result;
print "list1 is $res\n";

$soap_response = SOAP::Lite
  -> uri('http://ws.cleo.com')
  -> proxy('http://localhost:38080/ws/services/WSCconnector')
  -> release(
    SOAP::Data->name("sessname" => $sessname),
    SOAP::Data->name("disconnect" => SOAP::Data->type(boolean => "0"))
  );

$res = $soap_response->result;
print "Release returned $res\n";
```

VoiceXML

Using the VoiceXML Connector

```

<?xml version = "1.0"?>
<vxml version = "2.0" xmlns="http://www.w3.org/2001/vxml">
  <var name = "sessionName" expr = "'testsess'"/>
  <var name = "transSetName" expr = "'sample'"/>
  <var name = "pool" expr="'test'"/>
  <var name = "featureName"/>
  <var name = "status"/>
  <var name = "gethostip" expr="'y'"/>
  <var name = "getport" expr = "'y'"/>
  <form>
    <var name = "list1"/>
    <var name = "transaction"/>
    <var name = "currentScreen"/>
    <var name = "sessionId"/>
    <var name = "keyword"/>
    <var name = "returnCode"/>
    <block>
      <assign name = "featureName" expr = "'reserve'"/>
    </block>
    <subdialog name = "reserve_0" src = "VXMLConnector" namelist = "featureName sessionName
pool transSetName gethostip getport">
      <filled>
        <assign name = "status" expr = "reserve_0.status"/>
        <assign name = "returnCode" expr = "reserve_0.returnCode"/>
        <assign name = "gethostip" expr = "reserve_0.gethostip"/>
        <assign name = "getport" expr = "reserve_0.getport"/>
      </filled>
    </subdialog>
    <block>
      status is <value expr = "status"/>
      return code is <value expr = "returnCode"/>
      gethostip is <value expr="'10.9.1.5'"/>
      getport is <value expr="'7561'"/>
    </block>
    <block>
      <assign name = "featureName" expr = "'getSessionId'"/>
    </block>
    <subdialog name = "getSessionId_1" src = "VXMLConnector" namelist = "featureName
sessionName">
      <filled>
        <assign name = "status" expr = "getSessionId_1.status"/>
        <assign name = "sessionId" expr = "getSessionId_1.sessionId"/>
      </filled>
    </subdialog>
    <block>
      status is <value expr = "status"/>
      session id is <value expr = "sessionId"/>
    </block>
    <block>
      <assign name = "featureName" expr = "'runTransaction'"/>
      <assign name = "transaction" expr = "'getList'"/>
      <assign name = "keyword" expr = "'cleo'"/>
    </block>
    <subdialog name = "runTransaction_2" src = "VXMLConnector" namelist = "featureName
sessionName transaction getList">
      <filled>
        <assign name = "status" expr = "runTransaction_2.status"/>
        <assign name = "returnCode" expr = "runTransaction_2.returnCode"/>
        <assign name = "list1" expr = "runTransaction_2.list1"/>
      </filled>
    </subdialog>
  </form>

```

```
</subdialog>
<block>
  status is <value expr = "status"/>
  return code is <value expr = "returnCode"/>
  list1 is <value expr = "list1"/>
</block>
<block>
  <assign name = "featureName" expr = "'getCurrentScreen'"/>
</block>
<subdialog name = "getCurrentScreen_3" src = "VXMLConnector" namelist = "featureName
sessionName">
  <filled>
    <assign name = "status" expr = "getCurrentScreen_3.status"/>
    <assign name = "currentScreen" expr = "getCurrentScreen_3.currentScreen"/>
  </filled>
</subdialog>
<block>
  status is <value expr = "status"/>
  current screen is <value expr = "currentScreen"/>
</block>
<block>
  <assign name = "featureName" expr = "'release'"/>
</block>
<subdialog name = "release_4" src = "VXMLConnector" namelist = "featureName sessionName">
  <filled>
    <assign name = "status" expr = "release_4.status"/>
    <assign name = "returnCode" expr = "release_4.returnCode"/>
  </filled>
</subdialog>
<block>
  status is <value expr = "status"/>
  return code is <value expr = "returnCode"/>
</block>
</form>
</vxml>
```

Running the Sample Application

Disclaimer

Sample code is provided as a guide to aid in the development of your application. It is not guaranteed to work or continue to work in the future due to changing circumstances with the host.

Prerequisites

- Install the Transaction Processor software and secure the proper licensing. (Refer to the *Quick Start Guide* for details.)
- Be sure you can successfully connect to the sample host (nermvs.nerdc.ufl.edu:23).

Getting Started Using the 3270 sample Application

The sample application's transaction set includes transactions created with the Transaction Designer (TD) tool. To view the screens and transactions that make up the sample application, run the TD and choose sample from the initial pull-down box.

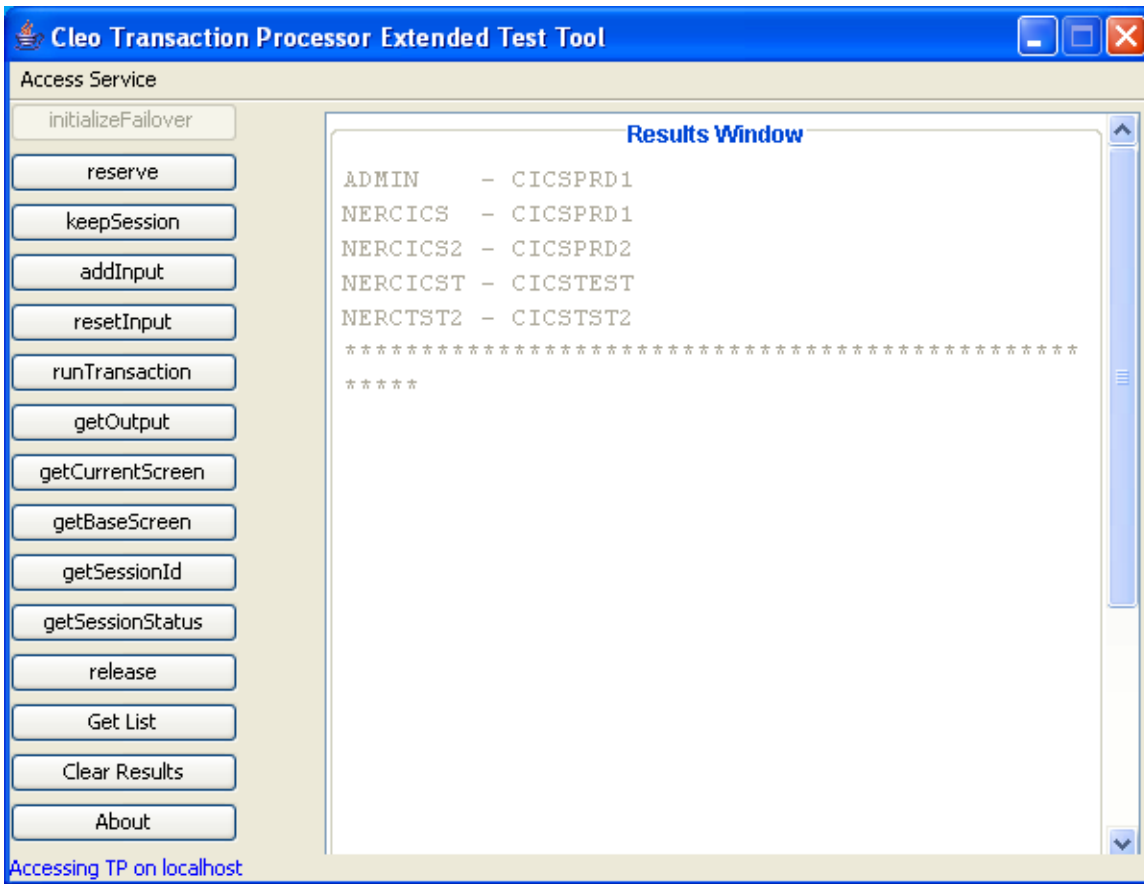
Here is a list of the defined transactions:

- login (which specifies the parked screen as the initial main screen)
- logout
- getlist (which creates a list of available Mainframe Systems when a 4-6 character keyword is sent to the mainframe)
- getlist_park (navigates back to the parked screen)

To run the above transactions against the sample host:

1. Launch the Java Tester.
2. Click **Get List**.
3. Enter a 4-6 character keyword(eg. cleo). If no name is entered (or Cancel is selected), the value for the request field specified in the TD is used ("cleo").
4. Click **OK**.

NOTE: For step-by-step instructions on launching the Java Tester, refer to the *Quick Start Guide*.



Results of the getlist will display in the “Results Window”, similar to the above illustration.

For Java source code similar to what executes when the “Get List” button is clicked, see page 56.

Sample 3270 Transaction Files

The Transaction Designer (TD) generates files containing XML that are used by the Transaction Processor (TP) to access host application screens. For example, the login.xml file is used to log into the host. Following are the sample application XML files that were created by a developer using the TD.

login.xml

The following transaction contains one screen indicating the parked screen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- DOCTYPE screenset-transaction -->
<screenset version="2.0">
  <screensetname value="login" />
  <request>
    <screenname value="FirstScreen" />
  </request>
</screenset>
```

getlist.xml

The following user-defined transaction uses an input keyword to produce a list of available Mainframe Systems:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- DOCTYPE screenset-transaction -->
<screenset version="2.0">
  <screensetname value="getlist" />
  <request>
    <screenname value="FirstScreen.3" />
    <field name="keyword" value="cleo"/>
    <aidkey value="@E" />
  </request>
  <request>
    <screenname value="menu" />
  </request>
</screenset>
```

getlist_park.xml

The following user-defined transaction is run automatically by the TP only after the session has been released IF getlist was the last transaction run.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- DOCTYPE screenset-transaction -->
<screenset version="2.0">
```

```

<screensetname value="getlist_park" />
<request>
  <screenname value="anyscreen" />
  <aidkey value="@C" />
</request>
<request>
  <screenname value="FirstScreen.4" />
</request>
</screenset>

```

logout.xml

The following transaction contains one screen identifying the logged-out screen.

```

<?xml version="1.0" encoding="UTF-8" ?>
<!-- DOCTYPE screenset-transaction -->
<screenset version="2.0">
  <screensetname value="logout" />
  <request>
    <screenname value="FirstScreen.2" />
  </request>
</screenset>

```

scdefs.xml

The following contains all screen and field definitions for the sample transaction set:

```

<?xml version="1.0" encoding="UTF-8" ?>
<!-- DOCTYPE screen-definitions -->
<screen-definitions version="2.0">
  <screen>
    <screenname value="FirstScreen.1" base="FirstScreen" />
    <identification startrow="1" startcol="16" endrow="1" endcol="29" value="SSS
    Computing" shouldAppear="true" acceptAny="false" isBlank="false"/>
    <identification startrow="2" startcol="21" endrow="2" endcol="32" value="&
    Networking" shouldAppear="true" acceptAny="false" isBlank="false"/>
    <identification startrow="3" startcol="16" endrow="3" endcol="32" value="SSS
    Services" shouldAppear="true" acceptAny="false" isBlank="false"/>
    <identification startrow="5" startcol="16" endrow="5" endcol="42" value="SSS
    CNS Menu" shouldAppear="true" acceptAny="false" isBlank="false"/>
    <identification startrow="23" startcol="2" endrow="23" endcol="40" value="Enter
    Command or Service name or LOGOFF" shouldAppear="true" acceptAny="false"
    isBlank="false"/>
    <identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
    shouldAppear="true" acceptAny="false" isBlank="false"/>
  </screen>
  <screen>
    <screenname value="anyscreen" base="AnyScreen" />

```

```

<identification startrow="0" startcol="0" endrow="0" endcol="0" value=""
  shouldAppear="true" acceptAny="true" isBlank="false"/>
<field name="list.1" startrow="7" startcol="2" endrow="14" endcol="80" type="rec"
  float="no"/>
<field name="parkit" startrow="24" startcol="7" endrow="24" endcol="80"
type="send" float="no"/>
</screen>
<screen>
  <screenname value="FirstScreen" base="FirstScreen" />
  <identification startrow="1" startcol="16" endrow="1" endcol="29" value="SSS
  Computing" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="2" startcol="21" endrow="2" endcol="32" value="&
  Networking" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="3" startcol="16" endrow="3" endcol="32" value="SSS
  Services" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="5" startcol="16" endrow="5" endcol="42" value="SSS
  CNS Menu" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="23" startcol="2" endrow="23" endcol="40" value="Enter
  Command or Service name or LOGOFF" shouldAppear="true" acceptAny="false"
  isBlank="false"/>
  <identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
  shouldAppear="true" acceptAny="false" isBlank="false"/>
</screen>
<screen>
  <screenname value="FirstScreen.4" base="FirstScreen" />
  <identification startrow="1" startcol="16" endrow="1" endcol="29" value="SSS
  Computing" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="2" startcol="21" endrow="2" endcol="32" value="&
  Networking" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="3" startcol="16" endrow="3" endcol="32" value="SSS
  Services" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="5" startcol="16" endrow="5" endcol="42" value="SSS
  CNS Menu" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="23" startcol="2" endrow="23" endcol="40" value="Enter
  Command or Service name or LOGOFF" shouldAppear="true" acceptAny="false"
  isBlank="false"/>
  <identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
  shouldAppear="true" acceptAny="false" isBlank="false"/>
</screen>
<screen>
  <screenname value="screen3.3" base="AnyScreen" />
  <identification startrow="0" startcol="0" endrow="0" endcol="0" value=""
  shouldAppear="true" acceptAny="true" isBlank="false"/>
  <field name="recover" startrow="24" startcol="7" endrow="24" endcol="80"
  type="send" float="no"/>
</screen>

```

```
<screen>
  <screenname value="FirstScreen.2" base="FirstScreen" />
  <identification startrow="1" startcol="16" endrow="1" endcol="29" value="SSS
  Computing" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="2" startcol="21" endrow="2" endcol="32" value="&
  Networking" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="3" startcol="16" endrow="3" endcol="32" value="SSS
  Services" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="5" startcol="16" endrow="5" endcol="42" value="SSS
  CNS Menu" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="23" startcol="2" endrow="23" endcol="40" value="Enter
  Command or Service name or LOGOFF" shouldAppear="true" acceptAny="false"
  isBlank="false"/>
  <identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
  shouldAppear="true" acceptAny="false" isBlank="false"/>
</screen>
<screen>
  <screenname value="menu" base="menu" />
  <identification startrow="1" startcol="50" endrow="1" endcol="80" value="Press
  CLEAR&lt;Pause&gt; for CNS Menu" shouldAppear="true" acceptAny="false"
  isBlank="false"/>
  <identification startrow="2" startcol="21" endrow="2" endcol="56"
  value="COMMAND OR SERVICE NAME UNRECOGNIZED" shouldAppear="true"
  acceptAny="false" isBlank="false"/>
  <identification startrow="4" startcol="26" endrow="4" endcol="38" value="Services
  List" shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="6" startcol="2" endrow="6" endcol="8" value="Service"
  shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="23" startcol="2" endrow="23" endcol="63" value="Enter
  Command or Service name or LOGON APPLID=Applid or LOGOFF"
  shouldAppear="true" acceptAny="false" isBlank="false"/>
  <identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
  shouldAppear="true" acceptAny="false" isBlank="false"/>
  <field name="list1" startrow="7" startcol="2" endrow="7" endcol="20" type="rec"
  float="no"/>
  <field name="list2" startrow="8" startcol="2" endrow="8" endcol="20" type="rec"
  float="no"/>
  <field name="list3" startrow="9" startcol="2" endrow="9" endcol="20" type="rec"
  float="no"/>
  <field name="list4" startrow="10" startcol="2" endrow="10" endcol="20" type="rec"
  float="no"/>
  <field name="list5" startrow="11" startcol="2" endrow="11" endcol="20" type="rec"
  float="no"/>
</screen>
<screen>
  <screenname value="FirstScreen.3" base="FirstScreen" />
```

```
<identification startrow="1" startcol="16" endrow="1" endcol="29" value="SSS
Computing" shouldAppear="true" acceptAny="false" isBlank="false"/>
<identification startrow="2" startcol="21" endrow="2" endcol="32" value="&
Networking" shouldAppear="true" acceptAny="false" isBlank="false"/>
<identification startrow="3" startcol="16" endrow="3" endcol="32" value="SSS
Services" shouldAppear="true" acceptAny="false" isBlank="false"/>
<identification startrow="5" startcol="16" endrow="5" endcol="42" value="SSS
CNS Menu" shouldAppear="true" acceptAny="false" isBlank="false"/>
<identification startrow="23" startcol="2" endrow="23" endcol="40" value="Enter
Command or Service name or LOGOFF" shouldAppear="true" acceptAny="false"
isBlank="false"/>
<identification startrow="24" startcol="2" endrow="24" endcol="5" value="===&
;"
shouldAppear="true" acceptAny="false" isBlank="false"/>
<field name="keyword" startrow="24" startcol="7" endrow="24" endcol="80"
type="send" float="no"/>
</screen>
</screen-definitions>
```

Troubleshooting

It is very important to include some form of error handling within your application. The following sections provide the information needed to track down errors that may occur while an application is running using the Cleo Transaction Processor.

Troubleshooting Tips

It is worthwhile to leave at least one (1) host session unassigned. Leave it available to check on the status of the host connection, reproduce problems, ascertain the exact text and cursor location of host screen text, etc.

This circumvents the need to stop and start the Cleo Transaction Processor Service whenever a need to look at a host screen occurs. Use the Transaction Processor Session Viewer to view and manipulate the 3270 host screens.

Other tips:

- Set the Transaction Processor log threshold to DEBUG, temporarily, to analyze a problem. (Use the Web Admin configuration page to do this.)
- Change the configuration to only have one session assigned to the transaction set and use one voice channel to reduce the amount of logging information to analyze.
- Review the transaction.log file, which contains the Cleo Transaction Processor log messages.

It may be useful to turn on the low level and/or data stream tracing when trying to analyze a problem, especially if it is related to host communications. (See the *Cleo TP Administrator's Guide* for details).

File Location Information

The base path for all files used by the Transaction Processor is set to the directory path where the Cleo Transaction Processor software is installed (%TP_HOME% on Windows). The default is C:\Program Files\CleoTP on Windows, and /opt/cleotp on UNIX/Linux.

Location of Log Files

Cleo Transaction Processor

Windows: %TP_HOME%\logs\transaction.log

(Default is C:\Program Files\CleoTP\logs\transaction.log)

UNIX/Linux: /opt/cleotp/logs/transaction.log

When this log file reaches 1 megabyte in size, it is copied to transaction.log.<n>, where <n> is a number from 1 to 10, and a new transaction.log file is created. Thus, at most 11 log files will be created, and the most space these log files will use is 11 megabytes.

TP Client Applications

The TP client logging configuration log file is named
 log4j.properties
 and resides in the tomcat's "bin" directory.
 Windows: %TP_HOME%\tomcat\bin
 UNIX/Linux: /opt/cleotp/tomcat/bin

The TP client logging file itself is named
 clientlog.log
 and resides, by default, in the "tomcat" directory's "logs" directory. The location of the TP
 client logging file can be specified in the TP client logging configuration log file. Note that
 forward slashes are used to specify the location.
 The size and number of "clientlog.log" files to use are configurable.

TP Client Logging Configuration File

The format of the text file log4j.properties is:

```
log4j.rootLogger = DEBUG, FILE

log4j.appender.FILE=org.apache.log4j.RollingFileAppender
log4j.appender.FILE.File=/%TP_HOME%/tomcat/logs/clientlog.log
log4j.appender.FILE.MaxFileSize=10000KB
log4j.appender.FILE.MaxBackupIndex=10
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=%m%n
```

Where: The "log4j.rootLogger" parameter can have the values

DEBUG, FILE	(Same as DEBUG in previous TP Releases)
WARN, FILE	(Same as INFO in previous TP Releases)
ERROR, FILE	(Same as CRITICAL in previous TP Releases)
FATAL, FILE	(Same as FATAL in previous TP Releases)

NOTE: There is no long a log level parameter of "NONE".
 Use FATAL instead.

Location of Cleo Configuration Files on Windows**Cleo Transaction Processor**

%TP_HOME%\tomcat\bin\log4j.properties
 %TP_HOME%\resources\sessions
 %TP_HOME%\macros\keyboard[n].mac (where n is the dependant on the version of the TP)

Cleo Host Setting Files

%TP_HOME%\conf\hosts.cfg
 %TP_HOME%\conf*.zcc
 %TP_HOME%\conf*.hcc

Location of Cleo Configuration Files on UNIX/Linux

Cleo Transaction Processor

/opt/cleotp/tomcat/bin/log4j.properties
/opt/cleotp/conf/wrapper.conf
/opt/cleotp/resources/sessions
/opt/cleotp/macros/keyboard[n].mac (where n is the dependant on the version of the TP)

Cleo Host Setting and Log Files

/opt/cleotp/conf/hosts.cfg
/opt/cleotp/conf/*.tne
/opt/cleotp/conf/*.hcc
/opt/cleotp/conf/*.txt
/opt/cleotp/conf/*.cfg
/etc/opt/tn3270/com.cfg
/etc/opt/tn3270/*.stu
/opt/cleotp/bin/tnstart.tp
/var/opt/tn3270/com.err
/var/opt/tn3270/com.aud

Avoiding Common Problems

Application Startup

After the Cleo Transaction Processor service is started, it will attempt to get all the configured TN sessions to a **parked** state by running the **login** transaction of the transaction set(s) assigned to each configured session, followed by an optional **park** transaction.

After a reasonable amount of time, all of the configured sessions should get to a **parked** state. Viewing session status from the Transaction Processor Administration Page should show all the configured sessions in a **parked** state.

If all or some of the configured Sessions are in a **recovery** state, then there may be a problem in one of the following areas:

- **Transaction Set Problem** – the transaction set's **login** or **park** transactions may be incorrectly defined. Review the **login** and optional **park** transactions. Look at the transaction.log file for possible errors.
- **Host Server Problem** – the servers providing host communications may be having problems, or the host may not be available. On UNIX/Linux, check for messages in the tn3270 log files in /var/opt/tn3270. HLLAPI server tracing and tn3270 tracing can be turned on to help diagnose a problem. (See the *Administrator's Guide* for details).
- **Transaction Processor Problem** – Look at the transaction.log file for possible errors.

Application Call Processing

When a *runTransaction* function encounters a non-zero return code, there may be a basic problem with the TD transaction set transaction and field definitions, or the application's use of them.

OR

The host application may have encountered an error.

Try to use the *runTransaction* return code values to dynamically work around problems with the host connection.

Problems with Sessions getting Reparked

Use the Cleo Transaction Processor Web administration page to view the current status of sessions to see if the sessions are in a recovery state.

Look at the transaction.log file to attempt to determine the problem.

Error Handling

Different types of errors can be returned by the Transaction Processor functions in the return code value. The following chart lists possible return codes, identifies them by name, type, description and status of transaction.

Based on the type of error and the state of the transaction, you may decide to use different recovery strategies. Often, running a recovery transaction may suffice. Otherwise, it may be necessary to view the log for more information and/or contact technical support to determine what caused the error and how to correct the problem. (Refer to Log Messages section on page 75.)

Concerning return codes, consider the following:

- **Network/host errors** are potentially recoverable. Retry the transaction after resolving the network/host issues.
- **User application errors** usually result from incorrect information provided by the user application or defined in the transaction set. Check Transaction Designer (TD) definitions.
- **External environment errors** may be due to failures of one or more components and are generally fatal errors.

A session can be deactivated (removed from the session pool) using the Web Admin administration page. This allows the session viewer to be run to display the session's screen, which can be navigated back to the **parked** screen before re-activating the session. See Appendix A in the *Administration Guide* for details.

Return Code Chart

<i>Return Code</i>	<i>Error or Meaning</i>	<i>Type error</i>	<i>Description</i>	<i>Status of transaction</i>
0 *	Success			Executed
100 *	Success		Failover occurred; connection to secondary Transaction Processor Service was successful.	Executed
101 *	Success		Failover occurred; connection to secondary Transaction Processor Service due to sessions being in a host down state was successful.	Executed
-1 *	IO Exception	Internal error	Executor component failed to respond to the Connector while processing a Transaction Processor function.	Transaction may or may not have started
-2 *	Socket Exception	External environment	The Connector lost its connection to the Executor, probably because the session went into recovery.	Transaction may or may not have started
-100	No Connection to Service	External environment	The Connector was unable to connect to the Service, perhaps because it hasn't started.	Transaction not executed
-102 *	No Session Reserved	User Application	Function attempted without first reserving a session	Transaction not executed
-103 *	Unexpected Screen	Network/host	Unexpected screen detected, not specified in transaction being run.	Transaction started but not completed
-104 *	Unrecognized Screen	Network/host	Screen received not defined in TD transaction set	Transaction started but not completed
-105 *	Unknown Field Name	User application	Application To Host field isn't defined in transaction (or any of its associated branch transactions). Return code can only appear when log level is set to DEBUG.	Transaction not executed

<i>Return Code</i>	<i>Error or Meaning</i>	<i>Type error</i>	<i>Description</i>	<i>Status of transaction</i>
-106 *	Invalid Field Value	User application	Application To Host field exceeds size defined in TD transaction set.	Transaction started but not completed
-107 *	Unknown Transaction	User application	Application transaction name not defined in TD transaction set	Transaction started but not completed
-108 *	System Error	Internal error	Usually, fatal error requiring restart of the Transaction Processor	Transaction may or may not have started
-109 *	Host Server Error	Internal error	Usually, fatal error requiring restart of the Transaction Processor	Transaction may or may not have started
-110 *	Wait Timeout	Network/host	No response from host within configured timeout. May need to increase timeout configuration parameter.	Transaction started but not completed
-111 *	Screen Not Defined	Internal error	Transaction looking for a screen that is not defined in TD transaction set	Transaction started but not completed
-112 *	Undefined Request	Internal error	Transaction looking for a request screen that is not defined in TD transaction set	Transaction started but not completed
-113 *	Sessions Not Initialized	External environment	Cleo Transaction Processor startup problem	Transaction not executed
-114 *	No Session Available	Network/host	Sessions not configured or all in use.	Transaction not executed
-116 *	Host Session Status Error	Network/host	Host session locked due to CommCheck, ProgCheck, Busy, or Wait condition.	Transaction may or may not have started
-118 *	Ignorable Transaction	User application	XML doesn't exist for optional transactions: park, logout, recovery, <transaction>_park.	Transaction not executed
-119 *	Missing Screen Identifiers	Internal error	Unable to recognize screen since it has no screen identifiers	Transaction may or may not have started

<i>Return Code</i>	<i>Error or Meaning</i>	<i>Type error</i>	<i>Description</i>	<i>Status of transaction</i>
-120	Session is deactivated	External environment	Session has been taken out of session pool, no longer available to the Transaction Processor.	Transaction not executed
-122 *	Invalid XML file	External environment	One or more XML files within the transaction set contain invalid XML format or version number.	Transaction not executed
-123 *	Session already reserved	User application	A second reserve was issued. Must release first.	Transaction not executed
-126 *	Sessions are HOST_DOWN	External environment	Sessions assigned to transaction set have lost host connectivity.	Transaction not executed.
-127 *	Failover unsuccessful	Internal error	Failover occurred; connection to secondary TP Service was unsuccessful.	Transaction not executed.
-128 *	Invalid pool	User application	Given pool does not exist.	Transaction not executed.
-130 *	Failover not initialized	User application	Obsolete. No Longer Returned. Used to be returned when the "InitializeFailover()" function was not called.	Obsolete. No Longer Returned.

* It is recommended that the impact of the return code on the application be given special consideration by the application designer and handled appropriately.

Log Messages

In certain cases, a return code may not give enough information to successfully track down the cause of a problem. You may wish to view the session log for more details. Use the Cleo Transaction Processor web administration page to view the transaction.log file. (See the *Administrator's Guide* for details).

Log Format and Configuration Levels

Format

The basic format of a log entry is:

<component>|<date/time> <log level> <message id> <message>

where:

- *component* identifies the originator of the message
- *log level* is the threshold as described in the Configuration section, following.
- *message id* is the key used to identify the message
- *message* is the actual text of the message.

Configuration Levels

Using the Cleo Transaction Processor administration utility, you can configure the extent of the detail of log messages. Choose from these six different log levels:

- **None** – Do not log any messages.
- **Debug** – Provide detailed information on what is occurring during a run of the Transaction Processor.
- **Info** – Provide basic information on what is occurring during a run of the Transaction Processor.
- **Warning** – Provide information on minor / recoverable / ignorable errors that are encountered during a run of the Transaction Processor.
- **Critical** – Provide information on errors that need to be addressed immediately before a run of the Transaction Processor can continue.
- **Fatal** – Provide information on errors that prevent the Transaction Processor from running and cannot be recovered.

Whether or not messages are logged depends on what log level is configured, as shown in the following table.

Log Level Setting	Log Message Level	Logged?
None	Debug	No
	Info	No
	Warning	No
	Critical	No
	Fatal	No
Debug	Debug	Yes
	Info	Yes
	Warning	Yes
	Critical	Yes
	Fatal	Yes
Info	Debug	No
	Info	Yes
	Warning	Yes
	Critical	Yes
	Fatal	Yes
Warning	Debug	No
	Info	No
	Warning	Yes
	Critical	Yes
	Fatal	Yes
Critical	Debug	No
	Info	No
	Warning	No
	Critical	Yes
	Fatal	Yes
Fatal	Debug	No
	Info	No
	Warning	No
	Critical	No
	Fatal	Yes

Message Descriptions

The following tables provide details on the possible warning, critical, and fatal level messages that may be logged to the transaction.log. They may not all relate to a programming error, but may at least help determine where the application failed.

For additional details on how to perform the suggested steps, please refer to the *Transaction Designer User's Guide* and/or the *Transaction Processor Administrator's Guide*.

Disclaimer: Much effort has been spent in compiling these tables and in making them as accurate as possible. However there may be cases where additional support will be necessary. In the event that the provided solution or workaround does not resolve the issue, please contact your Cleo service representative.

ID	Level	Message	Reason	Solution
1	Warning	Unable to connect to JtsHllapi socket (1st attempt) - will attempt again in 3 seconds.	Attempt was made to reserveLU. However, the Transaction Processor Service may not have successfully started the HLLAPI server providing host screen access.	Wait for automatic retry.
2	Warning	Unable to connect to JtsHllapi socket (2nd attempt) - will attempt again in 3 seconds.	Attempt was made to reserveLU. However, the Transaction Processor Service may not have successfully started the HLLAPI server providing host screen access. First and second tries failed.	Wait for automatic retry.
3	Critical	Third attempt failed to connect to JtsHllapi socket!	Attempt was made to reserveLU. However, the Transaction Processor Service may not have successfully started the HLLAPI server providing host screen access. Last attempt failed. Port may be wrong.	Last attempt made, no retries left. Check hserver.log for messages. Verify that the configured port is available using "netstat -a".
6	Fatal	<message>	A general Transaction Processor error occurred. Message will detail the reason.	N/A
7	Warning	Wait on thread terminated prematurely	Thread that has been waiting, sleeping, or otherwise paused for a long time has been interrupted by another thread. Usually occurs when the system is overloaded.	Reduce the number of configured sessions.
9	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.

10	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
11	Critical	N/A	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
12	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
14	Warning	Now removing session from pool. InvalidXMLException: Invalid version or content detected in an xml file for application <app name>.	XML (scdefs and transactions) is not the correct version for the Transaction Processor. This is likely due to manual editing of the files and/or using the wrong version of the TD.	If the log level supports CRITICAL messages, refer to message 176 in reference to XML versioning in the service.log or transaction.log for additional details. Stop the TP, redefine transactions with TD, restart the TP.
15	Critical	Version number in <xml file> not formatted properly.	A non-numeric version number is provided in the transaction or screen definitions file. This may be the case when files are hand edited.	Preferred method: Regenerate files in the most current version of the TD. Alternate method: Hand edit the XML to use the correct version number.
16	Debug	Session status updated - new status: <status>	Session has either been released or put into recovery.	N/A

17	Warning	InterruptedException during sleep in user-defined recovery.	Sleep between recovery attempts was interrupted.	This message can be ignored. The only affect this error will have is that recovery will be attempted more quickly than what was configured for each time the error occurs.
19	Critical	No sessions in session pool!	Sessions are either not configured/enabled or were not successfully initialized.	Check that sessions are configured and enabled in the Configurator. Check the transaction.log or service.log for any initialization errors.
27	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
28	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
29	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor currently uses jdom-b8.
37	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor currently uses jdom-b8.
38	Info	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor currently uses jdom-b8.
43	Debug	Transaction path: <path>	Provides debug path information for the transaction.	N/A

44	Warning	No Screens (Requests) defined in the Transaction file: <transaction file>. Transaction will be ignored.	At least one screen must be defined for the transaction to be valid.	Define screen(s) in the TD for the transaction. Copy transaction files to the Transaction Processor trans directory. Restart the Transaction Processor.
45	Critical	There was a problem reading in the Screens (Requests) for Transaction file: <transaction file>	An error occurred while parsing the transaction XML. This error could have been caused by using an incorrect version of the parser or an invalid XML format.	Recreate XML with a compatible version of the TD. Check that the version of the parser libraries are correct for the version of the Transaction Processor. The Transaction Processor is set up to use jdom-b8.
49	Critical	<message>	XML parsing error. Format of XML could be wrong or the wrong version of the parser libraries are being used. This error is more likely when the Transaction Processor is being used in a web based environment (ASP/JSP/etc).	Check that XML is in a valid format and uses the correct version of XML for the parser. The Transaction Processor is set up to use jdom-b8.
51	Debug	Repark set path: <repark file>	Provides debug path information for the park transaction.	N/A
52	Debug	SetSessionParms successful	Provides notification that the set session parameters was issued successfully.	N/A
53	Debug	NullHllapiException : Failed to reserve session. JtsHllapi could not be initialized.	HLLAPI server (JtsHllapi) for the requested session was never initialized.	Check hserver.log and transaction.log for errors that may have occurred during initialization.
54	Debug	Waiting for keyboard to unlock...	Provides notification that the TP is waiting for the keyboard to unlock before processing the transaction against the associated session any further.	N/A
55	Debug	Keyboard is unlocked.	Provides notification that the keyboard has become unlocked so that the TP will continue processing the transaction against the associated session.	N/A

56	Debug	Keyboard has not unlocked. Wait RC=<return code>	The host connection may have been lost, the host may be busy or in an error state, or the keyboard has become locked for other reasons (value sent to field may have been invalid).	Deactivate the session from the Administrator. View the session with the session viewer. If the keyboard is locked for non-fatal reasons, issue a reset, manually repark session, and activate session.
57	Debug	Throwing TimeoutException...	Provides notification that the TP timed out while waiting for the keyboard to unlock.	N/A
65	Debug	Expected screen <screen name> found after <n> times through pause with a total wait time of <m> seconds. Pause returned with rc=<return code>.	Provides debug response time information. The times will be high if the host is slow and pause may time out depending on what wait time is configured.	If the pause times out consistently, then the wait time may have to be increased in the Configurator. However, if the wait time has to be set above an acceptable value. Then check for problems with the network/host.
66	Warning	Request data was truncated	Input data sent in from the transaction was too long for the actual input field on the 3270 screen. This error can only occur if the field was defined incorrectly in the TD.	Redefine the field in the TD to match the actual field on the 3270 screen.
67	Warning	User input was truncated.	Input data sent in from the user app was too long for the actual input field on the 3270 screen.	The user app must be modified to verify that input sent in from the end user is valid.
70	Debug	There are <n> fieldDefs to process	Provides debug information about how many fields will be processed.	N/A
71	Debug	Processing RECEIVE field <field name>...	Provides notification of which from host field is about to be processed.	N/A
72	Debug	Processing SEND field <field name>...	Provides notification of which to host field is about to be processed.	N/A
73	Debug	User input supplied SEND fieldValue=<field value>	Provides debug information about the value that will be sent to the host screen. This value will override the field value given in the transaction.	N/A
74	Debug	User input triggered SEND macro fieldValue=<field value>	Provides debug information about the keyboard macro value that will be sent to the host screen. This value will override the field value given in the transaction.	N/A
75	Debug	User input length=<n>	Provides debug information about the length of the data sent in by the user.	N/A

76	Debug	Processing command "<command>"...	Provides notification that a command is about to be processed.	N/A
77	Debug	Transaction defined SEND fieldValue=<field value>	Provides debug information about the value that will be sent to the host screen. This value is taken from the transaction.	N/A
78	Debug	Transaction triggered SEND macro fieldValue=<field value>	Provides debug information about the keyboard macro value that will be used. This keyboard macro index is taken from the transaction.	N/A
79	Debug	Transaction field value length=<n>	Provides debug information about the length of the data defined in the transaction.	N/A
80	Debug	Returned data: <results>	Provides debug information about which data will be sent back to the Connector/user app.	N/A
81	Debug	Run of transaction successful	Provides notification that the transaction was run successfully.	N/A
82	Debug	Reserving session...	Provides notification that a session is about to be reserved.	N/A
83	Warning	Host connect returned BUSY/LOCKED (RC=<return code>), continuing forward.	The 3270 screen may be transitioning or previous activity on the screen caused the keyboard to become locked.	The reserve request will still be accepted and the session will be given to the Transaction Processor. Note: Unless the screen becomes unlocked or reaches a stable state, any subsequent runTransaction calls that are made may fail.
86	Debug	Powering off...	Provides notification that the session is getting powered off.	N/A

87	Debug	Powering on...	Provides notification that the session is getting powered on.	N/A
88	Warning	No Keyboard Macro Definitions defined for session, no substitution can be made for <fielddef value>	Keyboard macros were not previously configured for the session.	If keyboard macros are required, define the values in the Configurator following the instructions provided in the Admin Guide. Otherwise, the macro reference should be removed from the transaction using the TD.
89	Debug	Keyboard Macro substitution - <fielddef value> becomes <new value>	Provides debug information on which keyboard macro substitution is about to be made.	N/A
94	Fatal	Failed to reserve session. Could not obtain CommandMapper.	A CommandMapper is required to perform Transaction Processor tasks. This message should never be seen unless the TP or JVM have been corrupted.	Reinstall the TP and/or JVM.
96	Fatal	Failed to reserve session. No Session Pool defined.	The session pool was never initialized.	Check the service.log and transaction.log for any related errors. If the log level is set to INFO, then message 217 should have been entered.
97	Debug	Reserving session...	Provides notification that a reserve is about to be issued.	N/A
98	Critical	Failed to reserve session. No Host session available.	Sessions are either not configured in the sessions file, have not been successfully parked, or are in use or deactivated.	Check that sessions are configured, have been parked, and are not currently in use or deactivated. Try the reserve again later.
99	Warning	Failed to reserve session. All sessions are deactivated.	A reserve attempt was made when all sessions have been deactivated.	Activate session(s) in the Administrator and retry the reserve.

101	Fatal	Finished reserving session.	Provides notification that a successful reserve has been issued.	N/A
102	Fatal	HllapiException (RC=<return code>): <message>	The hllapi reserve returned a non success return code.	Check the return code provided in the message. See section in the Admin Guide on Hllapi return codes.
104	Fatal	Failed to run transaction <transaction>. Could not obtain a CommandMapper	Reserve not previously issued so command mapper never initialized.	Issue a reserve before runTransaction.
105	Fatal	Failed to run transaction <transaction>. No session was previously reserved	Reserve was not previously issued or reserve previously issued, but was not successful.	Try issuing reserve again, when successful, issue the runTransaction.
106	Debug	results=<result string>	Provides debug information on the results that were returned from the runTransaction.	N/A
107	Critical	HllapiException (RC=<return code>): <message>	An error occurred when the TP invoked a HLLAPI function.	Check the return code provided in the message. See section in the Admin Guide on Hllapi return codes.
108	Fatal	UnexpectedScreenException: saw <actual screens> when expected <expected screens>	A screen that did not match the screen defined in the transaction was encountered.	If the screen changed on the host and needs to be updated in the TD then do so. Verify that input from the user app or built into the transaction did not cause the wrong screen to appear.
109	Critical	UnrecognizedScreenException when expecting one of the following screens named: <expected screens>	A screen that was never defined in the TD appeared during a transaction.	Deactivate the session, view the screen (Note: this may be a screen in the recovery). If the screen is new then the transaction must be changed. Check that input from the user app and/or built into the transaction did not cause the screen to appear.

110	Critical	TimeoutException: <message>	A screen did not arrive in the configured amount of time. Either the host conditions have slowed response times, the Timeout configuration setting is set too low, or a wrong screen arrived and the Transaction Processor timed out waiting for the expected screen.	Deactivate the session, show the screen. If the screen is the expected screen, then reconfigure the Timeout configuration setting. Otherwise, something might be wrong with the transaction and/or input data.
111	Critical	NullDefinitionException: In transaction set <app name>, transaction <transaction> references an undefined screen definition named <screen name>	Transaction file may be out of sync with the screen definitions.	Verify that the screen definitions and transaction files match those defined in the TD.
112	Critical	NullRequestException at screen #<n> within transaction: "<transaction>" of transaction set "<app name>"	Request in transaction was either not parsed properly or no requests exist in the transaction.	Check for related errors in the transaction.log and service.log.
113	Critical	UndefinedTransactionException: <transaction> is undefined or invalid	The requested transaction either does not exist or is formatted incorrectly.	Check the trans directory for the transaction file. Check that the version of the XML matches the version required by the TP. Check for any versioning or parsing errors in the service.log and transaction.log.
114	Fatal	Screen does NOT have any Identifiers	Screen definitions are incomplete. Screen identifiers are required. If xml files are edited outside the TD then it is possible for this error to occur.	Use the TD to create the transaction and screen definition files. If this is not possible, refer to the section in the programmer's guide that documents the XML and lists the requirements.
115	Warning	Transaction <transaction> cannot be run because it does not exist (ignored)	An attempt to run a park, logout, recovery, or repark transaction that was not defined but is optional.	If the transaction is needed, then it should be defined in the TD and copied to the trans directory for use by the Transaction Processor.

116	Fatal	MissingReservedSessionException: User application attempted to run transaction <transaction> without previously issuing a reserve request.	Self-explanatory.	Issue a reserve before issuing a runTransaction.
117	Fatal	InvalidXMLException: Invalid version or content detected in an xml file for transaction set <transaction set name>.	XML (scdefs and transactions) is not the correct version for the TP.	If the log level supports ERROR messages, check for message 176 in reference to XML versioning in the transaction.log or service.log. Stop TP, validate the XML (recreate if necessary), restart the TP.
118	Fatal	OiaException: <message> Symbol - <oia symbol>	Keyboard is fatally locked for a non-time out reason. May never see this message. Normally, a timeout would occur first.	Check for problems with transaction first. Deactivate session, manually correct keyboard lock, return to first screen of the login transaction or to the parked screen, activate session.
121	Warning	getScreen Feature NOT implemented	This message will likely not appear.	This feature may have been superceded by deactivate, show3270.
124	Critical	Release failed. Could not obtain session	A session has not been successfully reserved.	Only issue a release when a successful reserve has been made.
125	Critical	Release failed. Could not obtain CommandMapper	A reserve has not been issued.	Issue a reserve before issuing a release.
127	Debug	Logging in session...	Provides notification that the session is about to be logged in.	N/A
128	Debug	Parking session...	Provides notification that the session is about to be parked.	N/A
130	Fatal	InvalidXMLException: Invalid version or content detected in an xml file for transaction <app name>.	XML (scdefs and transactions) is not the correct version for the TP.	If the log level supports ERROR messages, check for message 176 in reference to XML versioning in the transaction.log or service.log. Stop TP, validate the XML (recreate if necessary), restart the TP.
131	Warning	No PARK transaction to run after LOGIN, assuming LOGIN left session Parked	Provides notification that an attempt was made to run an undefined (optional) PARK transaction.	If a park transaction is required, then define it in the TD.

132	Critical	HllapiException (RC=<return code>): <message>	An error occurred when the TP invoked a HLLAPI function.	Check the return code provided in the message. See section in the Admin Guide on Hllapi return codes.
133	Fatal	UnexpectedScreenException: saw <actual screens> when expected <expected screens>	A screen that did not match the screen defined in the transaction was encountered.	If the screen changed on the host and needs to be updated in the TD then do so. Verify that input from the user app or built into the transaction did not cause the wrong screen to appear.
134	Critical	UnrecognizedScreenException when expecting one of the following screens named: <expected screens>	A screen that was never defined in the TD appeared during a transaction.	Deactivate the session, view the screen (Note: this may be a screen in the recovery). If the screen is new then the transaction must change. Check that input from the user app and/or built into the transaction did not cause the screen to appear.
135	Critical	TimeoutException: <message>	A screen did not arrive in the configured amount of time. Either the host conditions have slowed response times, the Timeout configuration setting is set too low, or a wrong screen arrived and the Transaction Processor timed out waiting for the expected screen.	Deactivate the session, view the screen. If the screen is the expected screen, then reconfigure the Timeout configuration setting. Otherwise, something might be wrong with the transaction and/or input data.
136	Critical	NullDefinitionException: In transaction set <transaction set name>, transaction <transaction> references an undefined screen definition named <screen name>	Transaction file may be out of sync with the screen definitions.	Verify that the screen definitions and transaction files match those defined in the TD.
137	Critical	NullRequestException at screen #<n> within transaction: <transaction> of transaction set <transaction set name>	Request in transaction was either not parsed properly or no requests exist in the transaction.	Check for related errors in the transaction.log.

138	Fatal	UndefinedTransactionException: <transaction> is undefined or invalid	The requested transaction either does not exist or is formatted incorrectly.	Check the trans directory for the transaction file. Check the version of the XML matches the version required by the TP and for any versioning or parsing errors in the logs. Check the transaction name against the name in the transaction map.
139	Fatal	Screen does NOT have any Identifiers	Screen definitions are incomplete. Screen identifiers are required. If xml files are edited outside the TD then it is possible for this error to occur.	Use the TD to create the transaction and screen definition files. If this is not possible, refer to the section in the programmer's guide that documents the XML and lists the requirements.
140	Warning	Transaction <transaction> cannot be run because it does not exist (ignored)	An attempt to run a park, logout, recovery, or _park transaction that was not defined but is optional.	If the transaction is needed, then it should be defined in the TD and copied to the appropriate trans directory for use by the Transaction Processor.
141	Fatal	MissingReservedLUException: User application attempted to run transaction <transaction> without previously issuing a reserve request		Issue a reserve before issuing a runTransaction.
142	Fatal	OiaException: <message> Symbol - <oia symbol>	Keyboard is fatally locked for a non-time out reason. May never see this message. Normally, a timeout would occur first.	Check for problems with transaction first. Deactivate session, manually correct keyboard lock, activate session.
143	Fatal	InvalidXMLException: Invalid version or content detected in an xml file for transaction set <transaction set name>.	XML (scdefs and transactions) is not the correct version for the TP.	If the log level supports ERROR messages, check for message 176 in reference to XML versioning in the transaction.log or service.log. Stop TP, validate the XML, restart the TP.
144	Fatal	<message>	A reset was requested on a non-numeric session id.	Try again with a valid session id.
152	Warning	StopService requested more than once	Notification that an attempt to stop the service has already been made.	Ignore.

153	Warning	Unable to logout session. Session is still in recovery.	A request to stop the service was made while the session was still in recovery.	Recovery will be cancelled. The screens must be manually returned to the starting screen of the login transaction before the service can be started again.
154	Warning	Unable to logout session. Session was assigned but not parked	The session is in an unacceptable state. The session may have been used previously in an unsuccessful transaction.	Wait for shutdown to complete. Manually return the screen to the first screen of the login transaction before attempting to start the service again.
155	Warning	Unable to logout session. Exception: <message>	Any number of exceptions could have occurred and will be described in the log message.	Wait for shutdown to complete. Manually return screen to first screen of the login transaction before attempting to start the service again.
156	Warning	Unable to logout session. Session was not assigned or parked.	Session may never have been used successfully in a transaction, released, but never reparked.	Wait for shutdown to complete. Manually return screen to the first screen of the login transaction.
157	Critical	Failed to stop host notify and disconnect from session, Hllapi RC=<return code>	An error occurred while the TP attempted to invoke the HLLAPI StopHostNotify function.	See section in the Admin Guide on Hllapi return codes.
159	Debug	Wait on thread terminated prematurely.	Thread may have been interrupted by the JVM. It is possible that the system is overloaded.	Reduce the number of sessions that are configured and restart the TP service.
161	Debug	Initializing session....	Provides notification that the session is about to be initialized.	N/A
164	Debug	Parking session...	Provides notification that the session is about to be parked.	N/A
165	Debug	Finished parking session.	Provides notification that the session was successfully parked.	N/A
166	Info	No PARK transaction to run after LOGIN, assuming LOGIN left session Parked	Provides notification that an attempt was made to run an undefined (optional) PARK transaction.	If a park transaction is required, then define it in the TD.
167	Critical	<message>	A general TP error occurred. Message will detail the reason.	N/A

168	Fatal	<session> is being placed in RECOVERY due to above error that prevented it reaching a PARKED state	An error occurred before the session could be successfully parked.	Deactivate session, depending on the type of error - either correct the transaction files, verify that the port for the HLLAPI server is available.
169	None	Successfully logged-in & parked all <n> configured session(s).	Provides notification that all sessions were successfully added to the session pool and are now available to be reserved by the user app.	N/A
170	Fatal	Unable to login & park ANY of the <n> configured session(s).	An error occurred that prevented all of the sessions from getting parked.	Check that the host is still up, the screen at startup was the correct screen to start the login at, and that the login (and park) transactions are valid.
171	Warning	Unable to login & park <m> of the <n> configured session(s).	Errors prevented some sessions from being parked.	Wait for recovery. If recovery is unsuccessful, then the session may be unrecoverable without restarting the Transaction Processor or there may be problems with the login/park transactions. Alternatively, deactivate, manually recover, and activate session.
175	Critical	Unable to check for XML version information. Exception: <message>	Error occurred when parsing the xml.	Verify that the correct version of XML is being used and that the installed parser is compatible with the XML.
176	Critical	Version number <version> in <xml file> is not less than or equal to <accepted version>	Invalid version in transaction or screen definition file for the transaction set.	Check that the version in the files matches the accepted version. File may have to be republished in the most current version of the TD.
177	Critical	Recover or repark request made against an unspecified (null) session	Session is lost before the recover/repark could be called.	Recovery/Repark will never occur. May have to manually recover/repark session by deactivating session, returning screen to the first screen of the login or the parked screen, and activating session.
178	Debug	Attempt made to recover or repark session prevented since session is deactivated.	Once a session has been deactivated, any attempts to recover or repark the session by the TP Service will fail.	Manually repark/recover session. Activate session.
180	Info	Session successfully reparked by repark or within user defined transaction.	Provides notification that the session was either reparked by the repark transaction or by the last transaction run.	N/A

181	Critical	Reparking via transaction <transaction> failed. Exception: <message>	General TP error occurred while running the repark transaction. Message will provide details.	Depending on the type of error, manual repark/recovery of session may be possible by deactivating session, returning screen to the first screen of the login or parked screen, and activating session.
183	Info	RECOVER: Will attempt to run the user defined recovery transaction up to <n> times.	Provides notification that the user defined recovery transaction will be run up to the configured amount of time in an attempt to recover a lost session.	N/A
186	Critical	RECOVER: Recovery failed due to <message>	General Transaction Processor error occurred. Message will provide details.	Depending on type of error - check that HLLAPI Server is available for session and that the transaction files are valid. A manual recovery may also be possible by deactivating session, returning screen to start or parked screen, and activating session.
187	Warning	RECOVER: No user defined Recovery Transaction defined	Notification that recovery attempt was made, but a Recovery Transaction as defined in the TD does not exist.	If no recovery is needed, turn auto recovery off. Otherwise, if auto recovery will not work in this case, create a Recovery Transaction in the TD.
188	Warning	RECOVER: Maximum number of recovery tries has been reached.	Attempts to recover were made that exceeded the configured amount.	Check that the recovery transaction and/or auto recovery should work in this case. If one or the other should work, then there may have been other errors that prevented the recovery from succeeding. Manual recovery can be attempted.
189	Info	RECOVER: Now attempting auto recovery...	Provides notification that auto recovery (power off, power on, login) is about to be attempted. This message will only appear if auto recovery is turned on and the log level is set to INFO.	N/A
190	Debug	RECOVER: Logging in session...	Provides notification that a session is about to be logged in during the automatic recovery process.	N/A

191	Debug	RECOVER: Parking session...	Provides notification that the session is about to be parked during the automatic recovery process.	N/A
193	Info	RECOVER: No PARK transaction to run after LOGIN, assuming LOGIN left it Parked	Provides notification that an attempt was made to run an undefined (optional) PARK transaction during the automatic recovery process.	If a park transaction is required, then define it in the TD.
194	Critical	Automatic Recovery failed due to <message>	A general TP error occurred. Message will detail the reason.	Manual recovery may be possible by deactivating session, returning screen to the first screen of the login transaction or the parked screen, and activating session.
196	Info	License key: <key>	Provides information about the license key.	N/A
197	Fatal	LICENSE ERROR: The Cleo Transaction Processor is not licensed to run. Please contact salesEN@cleo.com with Subject: Cleo Transaction Processor License - Host ID = <host id>	License is either invalid or expired.	Follow provided instructions to obtain a new license file.
198	Info	Property applicationPath set to <app path>	Provides information about the configured path to the transaction sets.	N/A
199	Info	Property sessionsPath set to <session pool path>	Provides information about the configured path to resources (license file and sessions file)	N/A
200	Info	Property macrosPath set to <keyboard macros path>	Provides information about the configured path to keyboard macros.	N/A
201	Info	Property timeOut set to <time out>	Provides information about the configured amount of time (in seconds) to wait for a screen to arrive from the host.	N/A

202	Info	Property maxRecoveryTries set to <max recovery ct>	Provides information about the configured maximum number of times to attempt automatic/user defined recovery.	N/A
203	Info	Property interLoginInterval set to <inter login interval>	Provides information about the configured amount of time (in seconds) to wait before logging in the next session.	N/A
204	Info	Property recoveryRetryInterval set to <recovery retry interval>	Provides information about the time (in milliseconds) to wait before the next attempt is made to recover a session.	N/A
205	Info	Property recoveryWaitTime set to <recovery wait time>	Provides information about the time (in seconds) to wait before a session that has been stuck in a certain state is thrown into recovery.	N/A
206	Info	Property autoRecovery set to <auto recovery on off>	Provides information about whether the session should be auto recovered (powered off, powered on, logged in).	N/A
211	Critical	Comm check encountered for session.	Host session experienced a comm check. The link to the host is likely down.	Check for errors in low level trace or data stream trace if tracing is turned on. If link outage appears permanent: Stop TP. Contact host administrator to reset the link. Restart the TP.
212	Critical	Keyboard lock encountered for session, OIA state=<state>	Data previously sent to the host screen caused the keyboard to lock.	Check that data sent in from the user app and/or transaction are sent into an acceptable input field as defined by the transaction.
213	Debug	Busy host encountered for session, OIA state=<state>	Host may be in transition.	Can ignore. However, if the next transaction sends data to the screen before the host settles down, then the transaction may not run successfully.
214	Critical	BUG: Don't have the parkedScreenIDs to perform the PARKED check for the session.	Session never reached a parked state during startup.	If this error appears during a call, then the session will have to reactivated.

216	Debug	Building session pool....	Provides notification that the session pool is about to be read in.	N/A
217	Info	Session pool now contains <n> session(s)	Provides information about how many sessions are available to be reserved.	N/A
218	Warning	Session was stuck in state <state> for <age>. Initiating recovery.	Session maintained a certain state longer than what was configured (acceptable) by the recovery wait time.	If recovery fails, check that the transactions are valid and the communication to the host is still up before reactivating the session. If the host is slower than usual, reconfigure the recovery wait time.
219	Warning	Session is no longer Parked.	The host screen changed outside the TP - possibly by a line disconnect.	Deactivate the session, view the session. If there is a comm check or other host related conditions, correct them first before activating the session.
220	Fatal	Could not locate transaction file, <transaction file>.	An attempt was made to run a transaction that does not exist.	Copy the transaction file into the associated TP transaction set directory.
224	Warning	<service status>	Notification of whether the service successfully shutdown.	If sessions did not get logged out, manually return screens to the first screen of the login transaction.
225	Fatal	InvalidFieldNameException:Field sent in user data [<input>] is not defined in field definitions.	Field name in one or more field name/value pairs does not match a field name in the screen definitions and/or transaction files.	Use the transaction map generated by the TD to verify that the input field defined in the user app matches the to host field defined in the transaction and screen definitions.
226	Fatal	InvalidFieldValueException:value =<value>,fieldName=<field name>	An empty value was provided by the transaction or from the IVR app or the length of the value exceeded the defined length of the field.	Check that the field defined in the TD has the correct length. Modify the user app to only accept input that will be accepted by theTP.
227	Fatal	InvalidFieldNameException:Field sent in user data [<input>] is not defined in field definitions.	Field name in one or more field name/value pairs does not match a field name in the screen definitions and/or transaction files.	Use the transaction map generated by the TD to verify that the input field defined in the user app matches the to host field defined in the transaction and screen definitions.

228	Fatal	InvalidFieldValueException:value =<value>,fieldName=<field name>	An empty value was provided by the transaction or from the user app or the length of the value exceeded the defined length of the field.	Check that the field defined in the TD has the correct length. Modify the user app to only accept input that will be accepted by the TP.
229	Info	Transaction set, <transaction set name>, does not have any sessions assigned to it.	Sessions were not configured for <transaction set name>.	Check that the transaction set name is valid and that sessions have been assigned to use this transaction set using the Configuration page .
234	Warning	STOP_HOST_NOTIFY failed (RC=<return code>)	An attempt was made to stop host notification from the emulator that failed. Possibly, the start host notify was never issued or failed. Should not be seen by the TP.	Check the return code provided in the message. 1=An invalid session ID was specified, 8=Start Host Notification has not been requested for this presentation space, 33=Function cannot be called if the XNOTIFY parameter is set (not likely), 9=System error.
235	Debug	Wait on repark thread terminated prematurely.	Thread for repark that has been waiting, sleeping, or otherwise paused for a long time has been interrupted. Usually occurs when the system is overloaded.	Configure fewer sessions or stop other services that may be running on the system.
239	Critical	Recovery failed due to <message>	General error occurred. Message will provide details.	Depending on the error, manual recovery can be attempted. Deactivate session, manually return session back to the parked screen, activate session.
241	Info	Login thread is done.	Provides notification that the login of a session has completed.	N/A
242	Warning	InterruptedException during sleep in recovery thread.	Thread that has been waiting, sleeping, or otherwise paused for a long time has been interrupted by another thread. This may or may be reason for the recovery to fail.	If the session had not successfully been recovered, manual recovery may be possible by deactivating session, returning screen to the first screen of the login transaction or the parked screen, and activating session.
243	Warning	InterruptedException occurred. Did not wait for full interlogin gap time.	This error affects the interlogin interval. The gap may have been prematurely shortened. It may be possible for sessions to attempt to connect to the emulators too quickly and thus fail to connect.	If there is a problem with logging in any of the sessions, the TP may have to be restarted.

245	Critical	Property HllapiServer ipAddress set to <ip address>	Provides information about the configured ip address of the Hllapi Servers providing host access.	N/A
246	Critical	Property HllapiServer port set to <port>	Provides the configured port number of the Hllapi Server providing host access for the 1 st enabled session.	N/A
247	Warning	IOException occurred while shutting down host server.	Socket communication between TP and HLLAPI Server experienced an error.	Use "ps -ef grep Hll" to check if any HllapiServers are running. If so, kill each one using "kill <process id>". Wait for ports to become available again before restarting the TP (netstat -a).
248	Warning	StopService requested in an offline state.	Service is already stopped.	Ignore.
249	Warning	IOException: Failed to connect to JtsHllapi socket - <message> - will attempt again in 3 seconds.	An I/O error occurred that prevented the socket from being created. Should not be seen by the TP.	Wait for automatic retry.
250	Debug	Unable to spawn hllapi server for session <id>. Now removing session from pool. Exception: <message>	An exception occurred that prevented the HllapiServer process from being successfully spawned. Should not be seen by the TP.	Check the HServer.log for errors.
252	Info	IOException: Second attempt failed to connect to JtsHllapi socket - <message>	An I/O error occurred that prevented the socket from being created. First and second attempts failed. Should not be seen by the TP.	No additional attempts will be made. Check the HServer.log for errors.
254	Fatal	Failed to reserve session. JtsHllapi could not be initialized.	Connection to the OHIO Service could not be established.	Stop TP. Stop any OHIO Service(s) that were not shut down by the TP. Reboot the system. Check the log for any OHIO Service related errors. Correct any problems with the port, etc. Restart the TP.
255	Warning	Connect timed out - will attempt again in 3 seconds.	Emulator did not respond to Hllapi library in configured time. Should not be seen by TP. It is possible that the TP was incorrectly configured to use HllapiServers instead of OHIO Service(s).	Use "ps -ef grep Hll" to check if any HllapiServers are running. If so, kill each one using "kill <process id>". Wait for ports to become available again before restarting the TP (netstat -a).

256	Warning	Connect timed out twice - will attempt again in 3 seconds.	Emulator did not respond to Hllapi library in configured time. First and second attempts failed. Should not be seen by TP. It is possible that the TP was configured incorrectly to use HllapiServers instead of OHIO Service(s).	Use "ps -ef grep Hll" to check if any HllapiServers are running. If so, kill each one using "kill <process id>". Wait for ports to become available again before restarting the TP (netstat -a).
257	Critical	Exception occurred during connect - <message>	Socket connection may have been lost or the network may be having problems. Should not be seen by TP. TP is unable to access Hllapi Server for host access.	Stop the TP service. Use "ps -ef grep Hll" to check if any HllapiServers are running. If so, kill each one using "kill <process id>". Wait for ports to become available again before restarting the TP (netstat -a).
258	Critical	Screenname could not be parsed!	ScreenDefinition XML was not created properly with appropriate <screenname...> tags.	Regenerate the XML with a compatible version of the TD.
259	Debug	Releasing session...	Provides notification that the session is about to be released.	N/A
261	Critical	Failed to initialize session - no instance of JtsHllapi could be created!	Connection to Hllapi Server for session was not successfully established.	Stop the TP. Stop any Hllapi Servers that were not stopped by the TP. Check that configured ports are available (netstat -a). Restart the TP.
262	Info	Failed to initialize session - SetSessionParms failed. RC=<return code>	The HLLAPI SetSessionParms function returned a non -success return code. Should not be seen by the TP. It is possible that the TP was incorrectly set up to use HllapiServers instead of OHIO Service(s).	Replace jar with correct build.
263	Critical	Recovery failed. Session is now removed from pool.	Recovery will not occur if a session has been deactivated.	If recovery is required the session will have to be manually recovered by returning screen to the first screen of the login or parked screen and then activating the session.

264	Warning	Failed to complete reset command due to <message>	An error occurred while trying to execute a power on and run the login transaction, as a result of a reset command configured for a screen by the Transaction Designer.	Check that there aren't any problems with the login transaction and that a Hllapi Server is available for the session. Manually recover the session by deactivating session, getting the screen back to the first screen of the login, and activating session.
265	Info	Spawning hllapi server for session...	Provides notification that a hllapi server is about to be started for the session.	N/A.
266	Info	License file (<path>cleotp.lic) does not exist.	License file was never copied into the <path> as required or was never acquired from Cleo Sales.	Once license is obtained, copy file into the <path>.
267	Fatal	Logging in session...	Provides notification that the session is about to be logged in as part of the activate.	N/A
268	Fatal	Parking session...	Provides notification that the session is about to be parked as part of the activate.	N/A
269	Info	No PARK transacton to run after LOGIN for session <id>, assuming LOGIN left it Parked	Provides notification that an attempt was made to run an undefined (optional) PARK transaction.	If a park transaction is required, then define it in the TD.
270	Info	Exception occurred while restoring session: <message>	Including but not limited to invalid xml, missing screen definitions, and any of the errors that could occur during a runTransaction, reserveLU, or releaseLU.	Check that the login and park transactions are valid, a Hllapi Server is available for the session, and that the screen was left after the deactivate at the first screen of the login transaction or at the parked screen.
271	Debug	Exception while reading from Connector: <message>	An error occurred that prevented the successful read from the socket.	N/A
272	Critical	BUG: CommandMapper no longer available.	The CommandMapper went away for unknown reasons.	N/A
273	Critical	Session no longer available.	The session went away for unknown reasons.	N/A

274	Critical	IOException occurred while closing connection to Connector.	An error occurred on the network that prevented the connection from being closed successfully.	N/A
275	Debug	Building Host Settings...	Provides notification that the settings for the host are about to be read in.	N/A
276	Warning	Unable to start logger at port <port>. Exception: <message>	The configured port for the logger may not be available.	Ignore. Log messages will be diverted to the service.log.
277	Critical	Unable to stop OHIOService<n>.	An error occurred that prevented the OHIO Service from successfully being shutdown by the TP.	Stop the OHIO Service from the Windows NT Services manager.
278	Fatal	Unable to start ohio-service<n>.	An error occurred that prevented the OHIO Service from being successfully started by the TP.	If none of the configured OHIO Services can be started successfully, then the TP should be stopped. Check that the configured base port (range) is available.
279	Critical	Unable to start session.	Possible that the OHIO Service for the session was never started. It is also possible that the wrong host setting was configured.	Check that the OHIO Service is running for the session. Check the Event Viewer for any related messages. Verify from the Configurator that the correct host setting was configured for the session.
280	Fatal	Unable to connect to ohio server. Now removing sessions from pool. Exception: <message>	An error prevented the session from successfully connecting to OHIO Service.	Check that configured ports are available and that OHIO Service(s) were started.
282	Info	Property servicePort set to <port>	Provides information about the configure port of the TP Service.	N/A
283	Info	Property inittime set to <time>	Provides information about the allowable time (in milliseconds) for the TP Service to start up.	N/A
284	Info	Property stoptime set to <time>	Provides information about the allowable time (in milliseconds) for the TP Service to shutdown.	N/A
285	Info	Property enableTrace set to [off low level data stream both]	Provides information about what level of tracing should be turned on.	N/A
286	None	Prog check encountered for session.	Host session experienced a prog check. The host is no longer responding.	Contact the host administrator to reestablish the host connection.
287	None	Released session was not PARKED.	The check for whether the session was parked failed normally or due to error.	Manually recover session from the Administrator.

289	Warning	Number of sessions configured are greater than the number licensed. Will now truncate the session pool.	The sessions file contains more sessions than what the license allows.	If more sessions are needed, contact Cleo sales to purchase a new license.
290	Critical	Exception occurred while logging in session: <message>	One of the following could cause this log message: invalid XML version number, undefined screen, missing screen identifiers, missing request screen. These problems are most likely a result of hand editing XML files.	The session is put into DEACTIVATED state. Correct the XML files using the Publish function in the TD for the erroneous transaction set, transfer the files into the TP transaction set directory. Restart the TP. (Refer to the Administrator's Guide)
291	Fatal	Exception reading int from Executor: <message>	Executor either closed down the socket or failed to send back data or conditions on the network prevented data from arriving.	May not be a recoverable state. Could try restarting the Connector.
292	Fatal	Exception reading bytes from Executor: <message>	Executor either closed down the socket or failed to send back data or conditions on the network prevented data from arriving.	May not be a recoverable state. Could try restarting the Connector.
293	Warning	Reserve failed, returned rc=<return code>	Sessions may not be available to be reserved due to being used or being in recovery/deactivated state.	Optionally, try the reserve again since a session may have been recovered, activated, or released.
294	Fatal	Lost access to session due to <message>	Executor closed socket.	May not be a recoverable state. Could try reinitializing the Connector.
295	Critical	Recovery of unspecified session failed.	Session was never instantiated (NULL).	N/A
296	Warning	Attempting to recover from a host outage, rc=<return code>	Host is unavailable due to host outage, network outage, or link outage.	In most cases, the TP will attempt to reestablish session automatically when the host becomes available again. However if an invalid host configuration (invalid port, invalid ip address) is given, will need to correct settings and then restart the TP. Return Codes: COMM_CHECK = 101 NVT = 105 MACHINE_CHECK = 106 DISCONNECTED = 107 UNOWNED = 109
297	Info	Host is now up. Logging in session.	When the host becomes available again after an outage, the TP attempts to relogin (park) the session.	N/A

298	Warning	Unable to login session after host outage. Initiating RECOVERY.	The login failed.	If recovery is configured, the TP will attempt to recover the session.
300	Warning	Session is not licensed for use by the OHIO Service.	The license for HIO contains fewer sessions that what is configured by the TP. This is normally the case when either the TP license was generated with the wrong HIO license key or the TP license has not been properly installed.	Obtain a valid TP license where the HIO license matches the TP license count and then install using the Configurator.
302	Info	Now running the associated branch transaction - <transaction name>.	The transaction that was previously running was defined to branch. That is, a subsequent transaction is running based on what screen was returned by the host.	N/A
303	Fatal	Unable to start emulators. Exception: <stack trace>	Java exception occurred executing a command to start the tn3270 emulators on Unix. TP service is set to OFFLINE and must be restarted.	Contact Cleo Tech. Support, providing log file containing this log message.
304	Fatal	Some or all emulators did not start on Unix.	tn3270 emulators did not start running on Unix. TP service is set to OFFLINE and must be restarted.	Check that begin3270 and tstart.tp exist in /opt/cleotp/bin. Check the com.aud and com.err files in /var/opt/tn3270 for error messages related to starting the emulators. Contact Cleo Tech. Support, providing log files.
305	Warning	Unable to check if emulators are running. Exception: <stack trace>	Java exception occurred executing a "grep" command to check that the tn3270 emulators are running on Unix. The service continues to attempt to process host communications for enabled sessions.	Contact Cleo Tech. Support, providing log file containing this log message.
306	Critical	Unable to stop emulators. Exception: <stack trace>	Java exception occurred executing a command to stop the tn3270 emulators on Unix. This may affect the ability of the TP service to perform host communications if it is restarted.	On the Unix system, issue "tnstop3270" to manually stop the tn3270 emulators. Contact Cleo Tech. Support, providing log file containing this log message.

307	Info	Property assignedFactor set to <number>	Provides information about the factor used to multiply the recovery wait time (in seconds), resulting in the time to wait before a session that has been stuck in the assigned state is thrown into recovery.	N/A
308	Warning	Failed to initialize session - failed to connect to host in <timeOut> seconds!	There was no response from the host in the configured timeOut amount of time.	Wait for the recover host process to complete.
309	Warning	keepSession failed, returned rc=<return code>	Unsuccessful attempt to execute keepSession which allows application to keep control of session even when it goes into HOST_DOWN state.	Should never happen - bug.
311	Warning	Unable to connect to <url>, RC=<return code>.	TP could not communicate with site specified in <url>. Could be due to invalid URL, certificate not accepted, or the site is not available.	Use the Web Config. Page to check the host setting parameters for this connection. Use the Test Connection button to verify communications work.
312	Warning	Attempting to recover from an HTTP host outage.	Host is unavailable due to host outage, network outage, or link outage.	In most cases, the TP will attempt to reestablish session automatically when the host becomes available again. However if an invalid host configuration (invalid port, invalid ip address) is given, will need to correct settings and then restart the TP.
315	Critical	Invalid transaction format. There must be at least one block in the transaction.	XML Transaction xml file is not well-formed.	Use the XML TD to publish the transaction in the correct format.
316	Critical	Invalid transaction format. Each request requires a response (which could be empty).	XML Transaction xml file is not well-formed.	Use the XML TD to publish the transaction in the correct format.
317	Info	Connecting to <resourceName>= <resourcePath>.	Informational message indicating configured name of resource and the resource path being used for the connection.	N/A
320	Fatal	Failed to reserve XML session. Could not obtain CommandMapper.	A CommandMapper is required to perform TP communication tasks. This message should never be seen unless the TP or JVM have been corrupted.	Re-install the TP and/or JVM.

322	Fatal	Failed to run transaction <transName>. No XML session was previously reserved.	Reserve was not previously issued or reserve previously issued, but was not successful.	Try issuing reserve again, when successful, issue the runTransaction.
324	Warning	HTTP Connection Error - <message>	Unable to connect to the web application using HTTP. The URL may be invalid or unavailable.	Check the message to determine how to resolve the connection problem. Session is put into recovery unless keepSession was called.
325	Critical	<IO Exception message>	Error sending XML request on HTTP connection.	Check the host setting for the HTTP connection, including the address, port, resource path, and any headers configured for the resource.
326	Fatal	<JDOM Exception message>	There was an error parsing the xml in a request or response document.	Check the format of the XML response or request documents. Publish the transaction set using the XML TD and retry. Note that default namespaces (set to "") are not supported.
327	Warning	UnexpectedDocumentException when expecting one of the following documents: <expectedDocNames>	An XML document was received from the host that was not configured in the TD to be an expected response in the transaction being run.	Use the TD recorder to determine what the expected response from the host should be as a result of sending the particular XML request. Check that the input field value sent in the request was correct.
328	Warning	Could not locate XML transaction request < transName>	The transaction requested by the user does not exist, or a request referred to in the transaction does not exist.	Check that the name of the transaction was specified correctly (case sensitive), and that any requests referred to inside the transaction exist in the trans folder for the transaction set.
329	Critical	<illegal argument exception message>	There was an XML parser error when processing an XML request or response received.	Use the XML TD recorder to record possible response documents that can be received from the host, and publish the transaction set using them.
330	Warning	Cannot activate session <sessionId> - <reason>.	The session cannot be activated due to one of the following: host is down, in use or not configured, not in a deactivated state, in use by Session Viewer.	Take action based on <reason> provided.

332	Warning	Cannot deactivate session <sessionId> - <reason>.	The session could not be deactivated, i.e. taken out of the session pool, due to one of the following: host is down, in use or not configured, being held by an application (issued keepSession).	Take action based on <reason> provided.
335	Critical	Activate failed - <reason>.	Activating TN session <id> failed since it could not be navigated to the parked screen, due to <reason>.	The session is put into RECOVERY in an attempt to restore it to the parked state.
336	Info	Property useSSL set to true false	States whether or not the TP client is configured to run securely or not.	N/A
337	Warning	Processor timed out waiting for incoming request - will keep waiting.	Keep session issued, therefore even though a timeout occurred on the socket, the TP will continue processing incoming requests from the same client and not close the connection.	N/A
338	Warning	Processor timed out waiting for incoming request - will stop waiting.	No keep session issued, therefore the TP will stop processing incoming requests for the client and close the connection.	If more time is needed between each subsequent call into the TP from the client, then either increase the recovery wait time or issue a keep session.
339	Critical	Received improperly chained certificates.	The checkServerTrusted method was unable to sort a chain of certificates received from an https server.	Communication will not be possible to the particular https connection.
340	Warning	Received untrusted certificate from <subject>	A certificate was received that has not been accepted and put into the TP trust store file. The session has status UNAVAILABLE.	Use the "Test Connection" button on the host setting page for the particular host to receive, view, and accept the certificate. Then restart the TP service, and the session should go to PARKED status.
341	Warning	Received expired Certificate from <server>	A certificate that has expired was received when connecting to <server>.	No action is necessary - communications with <server> will continue.
342	Critical	Error creating SSL Context: <exception msg>	Initialization has failed for https communications.	Contact Cleo Technical Support.
346	Warning	getHostStatus returned <rc>	Non-zero return code indicates host is not available, checked every 5 secs. when attempting to recover a session from a host outage.	Check that communications to the host are available.

347	Warning	Notification attempt (trigger 4) failed.	When starting TP service, a notification email could not be sent.	Contact Cleo Technical Support, providing transaction.log file with this message.
348	Warning	Notification attempt (trigger 5) failed.	When stopping TP service, a notification email could not be sent.	Contact Cleo Technical Support, providing transaction.log file with this message.
349	Warning	Notification attempt (trigger 3) failed.	When a session went HOST_DOWN, a notification email could not be sent.	Contact Cleo Technical Support, providing transaction.log file with this message.
351	Warning	getHostStatus returned <rc>	When processing a reserve or runTransaction request, the TP detected a loss of host communications for the session, which is set HOST_DOWN.	Check that communications to the host are available for the host setting assigned to this session.
353	Critical	Transaction set <name> sessions in HOST_DOWN state.	When processing a reserve request, TP detected that sessions in transaction set are HOST_DOWN.	Check that communications to the host are available for the host setting assigned to this session.
355	Warning	Exception thrown in Processor.	An exception other than EOF or SSL exception occurred when waiting for information on the socket from the TP client. A stack trace follows identifying the exception and where it occurred.	No action required
356	Critical	Exception sending command <command> - close connection.	Error occurred returning results of a command to the TP Client. Results in connection to client being closed. A stack trace follows identifying the exception.	Contact Cleo Technical Support, providing transaction.log file with this message.
357	Warning	Received unidentified command <command> - close connection.	TP did not recognize command received on socket from TP Client. Causes connection to client to be closed.	Contact Cleo Technical Support, providing transaction.log file with this message.
358	Warning	Failed to send results to client - close connection.	Sending results to TP Client was interrupted due to the socket connection from TP service to client being broken. Could result in loss of output data sent to application.	Contact Cleo Technical Support, providing transaction.log file with this message. Check that application did not terminate abnormally.
359	Warning	Failed to send results to client - not connected	TP could not send results to TP Client due to the socket connection from TP service to client being broken. Could result in loss of output data sent to application.	Contact Cleo Technical Support, providing transaction.log file with this message. Check that application did not terminate abnormally.

361	Critical	Cannot send results to connector.	TP service was unable to send all results to TP client (application) due to failure of the socket connection. Could result in loss of data sent to application.	Contact Cleo Technical Support, providing transaction.log file with this message. Check that application did not terminate abnormally.
364	Fatal	RESET command failed due to: <message>	The RESET command which executes power off, power on followed by running the login transaction, failed to complete. Could be due to host not returning a screen within the configured timeout period. Seen with log message #264. Session is put into Recovery.	Check that there aren't any problems with the login transaction. Manually recover the session by deactivating session, getting the screen back to the first screen of the login, and activating session.
366	Critical	powerOn failed to connect to host session, rc= <rc>	Disconnecting and reconnecting to host session failed, most likely due to rc=1, host connection no longer available.	Check that communications to the host are available for the host setting assigned to this session.
367	Warning	Repark transaction <transactionName_park> is invalid or does not exist.	The TP has determined that a session released by an application after running <transactionName> is not at the PARKED screen, and there is no valid <transactionName_park> to run. Session will be put into Recovery.	Define a <transactionName_park> using the Transaction Designer, or add screens to <transactionName> to put the session at the PARKED screen before releasing session.
369	Critical	checkHostDown/reserve: Deactivating session due to HLLAPI system error (rc=9)	Problem has occurred in tn3270/tn5250 emulator, most likely on Solaris/Linux. Session will be DEACTIVATED.	TP Service must be restarted to recover this session.
371	Warning	Unable to update running TP since cannot locate config. file	An administrator has attempted to change configuration parameters using the TP web configuration page while the TP is running. The configuration file could not be located by the TP.	Check for the existence of cleotp.cfg in the conf directory where the TP was installed.
373	Critical	TP did not receive host screen within configured timeout period of <timeOut> secs. Followed by log message #107.	While running a transaction, the TP gave up waiting for a screen to arrive from the host after <timeOut> seconds. The session is put into RECOVERY unless the application issued keepSession for this session.	Check the screen TP did not receive for correct identifiers. Use the TD to import and view unexpected screens received by the TP from the scr_dump directory where the TP was installed.

Client Log Message Descriptions

The following tables provide details on the possible info, warning, critical, and fatal level messages that may be logged to the clientlog.log. They may not all relate to a programming error, but may at least help determine where the application failed.

For additional details on how to perform the suggested steps, please refer to the *Transaction Designer User's Guide* and/or the *Transaction Processor Administrator's Guide*.

Disclaimer: Much effort has been spent in compiling these tables and in making them as accurate as possible. However there may be cases where additional support will be necessary. In the event that the provided solution or workaround does not resolve the issue, please contact your Cleo service representative.

ID	Level	Message	Reason	Solution
600	Warning	Reserve <name> failed - no connection to TP Service (errorMessage).	Applicaton request to reserve a session assigned to transaction set <name> failed since the TP client could not contact the TP service due to errorMessage.	Check that the TP service is running.
601	Critical	Lost access to session due to <errorMessage>.	TP client used by application has lost its connection to the TP service.	Check that the TP service is running.
602	Critical	Reserve <name> failed - no connection to TP Service.	Applicaton request to reserve a session assigned to transaction set <name> failed since the TP client had no connection to the TP service.	Check that the TP Service is still running.
605	Critical	Could not create TPConnector using pool <poolName>.	TP client was unable to establish a connection to the pool named <poolName> providing access to one or more TP Servers.	Verify that the pool named <poolName> has been configured using the Connection Manager on the system where the application is running.
607	Critical	Error (errorMessage) creating TPConnector to TP Service at <IPAddress>:<port>.	TP client was unable to establish a connection to the TP server running at the given IP address and port.	Verify that there is a TP server running on the system at the given IP address and port.
612	Critical	keepSession failed: no session reserved.	Application called the TP keepSession function without first reserving a session.	Make sure a successful reserve is done before calling the keepSession function.
613	Critical	runTransaction <name>.rc=<rc>	Transaction with the given name failed to run due to return code <rc>.	Check the Programmer's Guide for the meaning of the return code to determine what action is necessary.

618	Critical	addInput failed: <errorMessage>	Application called the TP addInput function which could not be processed due to one of the following: Session not reserved, null value provided for a field.	Make sure a successful reserve is done before calling the addInput function. Verify that the application is not passing a null field value to the addInput function.
619	Critical	getOutput:no session reserved	Application called the TP getOutput function when no session was reserved.	Make sure a successful reserve is done before calling the getOutput function.
620	Critical	Unable to change failover retry interval to <num> minute(s). Failover must be initialized first.	Application attempted to call SetFailoverRetryInterval before calling InitializeFailover, or the number of minutes passed in was negative.	Make sure the application calls InitializeFailover before SetFailoverRetryInterval, and that the number of minutes passed in is positive.
621	Critical	Pool not specified, or not configured using the Connection Manager.	Application attempted to reserve a session using a pool when no pool is configured.	Use the Connection Manager on the system where the application is running to configure a pool containing 1 or more connections to TP servers.
622	Info	SSL in use – enabled cipher suite:	Socket connection between TP Client and TP Server is using SSL.	N/A
623	Fatal	VXML Client >API Request Feature< Error, No/Failed Prior Reserve! session name = >sessionName< transaction = >transaction<	VXML Application request cannot find Connector for the sessionName given(eg. No RESERVE API Request made prior to this API Request).	Make sure VXML Application does a RESERVER API Request before any other API Request.
624	Fatal	VXML Client >MISSING FEATURE NAME< Error! session name = >sessionName<	VXML Application request didn't specify a featureName parameter or the parameter "featureName" itself was not specified correctly(eg. spelling or case error).	Make sure VXML Application specifies a featureName and that the featureName parameter itself is correctly spelled and has correct case(lower case all except N)
625	Fatal	VXML Client >MISSING SESSION NAME< Error! Feature = >featureName<	VXML Application request didn't specify a sessionName parameter or the parameter "sessionName" itself was not specified correctly(eg. spelling or case error).	Make sure VXML Application specifies a sessionName and that the sessionName parameter itself is correctly spelled and has correct case(lower case all except N)
626	Fatal	VXML Client >MISSING TRANSACTION SET NAME< Error! Feature = >reserve< session name = >sessionName<	VXML Application reserve request didn't specify a transSetName parameter or the parameter "transSetName" itself was not specified correctly.	Make sure VXML Application specifies a transSetName in the reserve API Request and that the transSetName parameter itself is correctly spelled and has correct case
627	Fatal	VXML Client >MISSING TRANSACTION NAME< Error! feature = >runTransaction< session name = >sessionName<	VXML Application runTransaction request didn't specify a transaction to run or the parameter "transaction" itself was not specified correctly.	Make sure VXML Application specifies a transaction in the runTransaction API Request and that the transaction parameter itself is correctly spelled and has correct case

628	Fatal	VXML Client >INVALID FEATURENAME SPECIFIED< Error! feature = >featureName< session name = >sessionName<	VXML Application was not one of the 7: runTransaction reserve, resetInput, release getSessionId, getSessionStatus, getCurrentScreen	Make sure VXML Application specifies one of the 7 supported API featureNames. Make sure featureName is spelled correctly and has case specified correctly.
629	Fatal	VXML Client RESERVE Error RC = -nnn session name = >sessionName<	VXML Application reserve request failed with a negative Return Code.	Check RESERVE documentation for meaning of negative Return Code value.
630	Fatal	Client getCurrentScreen Error NO SESSION RESERVED (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getCurrentScreen failed due to the session not being reserved or no longer being reserved.	Check transaction.log file for more information about the failure.
631	Fatal	Client getCurrentScreen Error TP CONNECTION LOST (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getCurrentScreen failed because the Socket Connection between the TP Client and TP Server no longer exists.	Check transaction.log file for more information about the failure.
632	Fatal	Client getSessionStatus Error NO SESSION RESERVED (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getSessionStatus failed due to the session not being reserved or no longer being reserved.	Check transaction.log file for more information about the failure.
633	Fatal	Client getSessionStatus Error TP CONNECTION LOST (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getSessionStatus failed because the Socket Connection between the TP Client and TP Server no longer exists.	Check transaction.log file for more information about the failure.
634	Fatal	Client getSessionId Error NO SESSION RESERVED (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getSessionId failed due to the session not being reserved or no longer being reserved.	Check transaction.log file for more information about the failure.
635	Fatal	Client getSessionId Error TP CONNECTION LOST (NOTE: Cleo TP Server Session ID is contained in the error message.)	Application request getSessionId failed because the Socket Connection between the TP Client and TP Server no longer exists.	Check transaction.log file for more information about the failure.
636	Warning	Client RESERVE Error findConnection could not write connPool.xml file when InService state of pool connection was found to be changed.	RESERVE worked OK, but connPool.xml file could not be updated when an InService state of a pool connection was found to be changed.	Check clientlog.log file for more information about the failure.
637	Critical	Client RESERVE Error establishConnectionP passed a NULL value for the ConnectionPool CPOOL argument.	RESERVE will fail with RC_INVALID_POOL value of -128.	Check clientlog.log file for more information about the failure.

638	Critical	Client RESERVE Error establishConnectionP finds value of pool to be NULL.	RESERVE will fail with RC_INVALID_POOL value of -128.	Check clientlog.log file for more information about the failure.
639	Critical	Client RESERVE Error reserve ConnectionPool connpool is Null.	RESERVE will fail with RC_INVALID_POOL value of -128.	Check clientlog.log file for more information about the failure.
640	Critical	Client RESERVE Error reserve Array List cons is NULL.	RESERVE will fail with RC_INVALID_POOL value of -128.	Check clientlog.log file for more information about the failure.
641	Critical	Client RESERVE Error reserve Connection Pool connpool is NULL just before establishConnectionP call.	RESERVE will fail with RC_INVALID_POOL value of -128.	Check clientlog.log file for more information about the failure.
642	Warning	Client RESERVE Error findConnection Exception trying to lock connPool.xml file while writing(saving) it.	RESERVE may work OK, but connPool.xml file could not be updated when an InService state of a pool connection was found to be changed.	Check clientlog.log file for more information about the failure.
643	Debug	Client RESERVE Error findConnection failed to find working TP Server Connection.	RESERVE will fail with RC_FAILOVER_ERROR value of -127.	Check clientlog.log file for more information about the failure.
644	Warning	Client RESERVE Error reserve Exception trying to lock connPool.xml file while reading it.	RESERVE may fail with RC_LOCKERROR value of -129, however, read of connPool.xml file will be attempted anyway, and if OK RESERVE will continue.	Check clientlog.log file for more information about the failure.
645	Warning	Client RESERVE Error Reserve Exception trying to lock connPool.xml file before reading it.	RESERVE may fail with RC_LOCKERROR value of -129, however, read of connPool.xml file will be attempted anyway, and if OK RESERVE will continue.	Check clientlog.log file for more information about the failure.

Dynamic Failover

The Transaction Processor (TP) supports dynamic fail over to alternate Transaction Processor(s) when a TP server fails or host connectivity is interrupted. This feature is **ONLY SUPPORTED** when a TP Connector is created using the “pool” option. This section describes this feature in detail.

Description

Figure 3 shows an example of a configuration where the application is running on a separate application (IVR) server, having the ability to communicate with one of four stand-alone TP servers.

The TP Client manages the dynamic fail over capability from the IVR server, when, and **ONLY** when, the application creates a TP Connector using the “pool” option. When the application issues a *Reserve Session*, after creating a TP Connector using the “pool” option, the TP Client determines which TP Server is available and directs subsequent transaction requests to the active TP Server. Once a connection is established, it remains in effect throughout the customer's call.

If the TP Server goes out of service during a call, then that call's connection to the Host is lost. The remainder of the document will describe the failover capabilities in more detail, including the administration functions required to enable the Cleo Fail Over Monitor Service, set up, and manage the alternate failover servers.

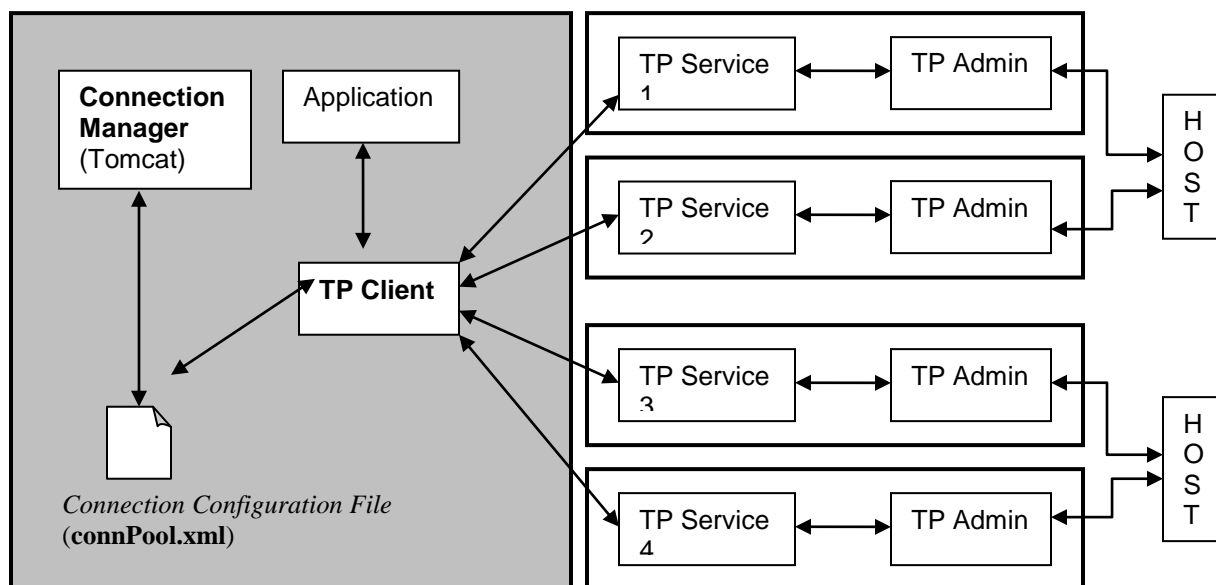
System Overview

In the dynamic failover mode, the TP Client provides access to one or more Transaction Processor services running on remote server(s). To support dynamic failover, a Connection Manager component is provided to enable/disable the Fail Over Monitor Service, and administer the connections to the various Transaction Processor Servers.

To provide a consistent web based interface, and leverage the built-in logging mechanism, the Connection Manager runs in an instance of Tomcat installed within the TP Client.

Component Diagram

Figure 3 Component Diagram



Component Descriptions

1. **Application:** any program that invokes the functions of the TP Client.
2. **TP Client:** provides access to any TP Service, associated with a “pool”, running locally or remotely through several API functions (reserve, addInput, runTransaction, getOutput, release, etc.)
3. **Failover Monitor Service:** When enabled by the TP Admin, the Failover Monitor Service runs periodically and maintains a list of pools and their connections to their TP Services and the status of each. Each pool connection has either a status of “In Service” or “Unavailable(Not In Service)”.
4. **Connection Manager:** provides web-based configuration of the TP connections for each pool and access to the associated configuration and administration functions for each configured TP Service connection in each pool. (See “Using the Connection Manager” in the Administration Guide for a more detailed discussion.)
5. **Connection Configuration File:** contains the list of TP Service pool connections (IP address, port, connection status) that are in the TP connection pool. This file is configured by the Connection Manager. The file is “connPool.xml” in the “resources” directory.

6. **Connection Pools:** for each “pool” a list of available TP Services assigned to the pool is kept. The TP Client will connect to the first available TP Service in a pools connection list, when a TP Connecotr using the “pool” option is created by an application.
7. **TP Admin:** provides configuration and administration functions for the TP Service.
8. **TP Service:** processes transactions at runtime.
9. **Host:** system running a TN 3270/5250 application with which the TP Service can exchange data.

Service Pool Management

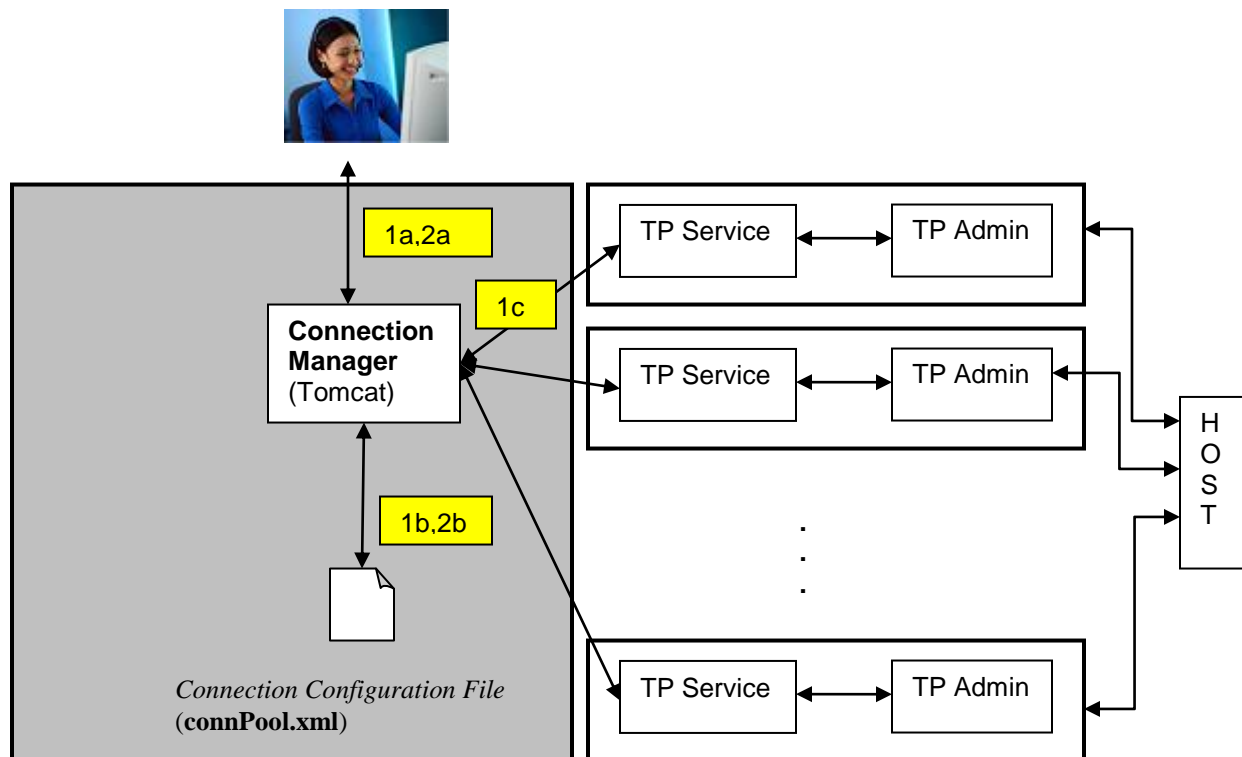


Figure 4 Configuration Flow Diagram

1. Status and configuration information is requested:
 - a. Administrator accesses the Connection Manager.
 - b. Connection Manager reads in the Connection Configuration File to determine which TP Services have been configured.
 - c. Connection Manager attempts to connect to each TP Service configured in the connection pool to determine and display its status
2. Configuration information is updated:
 - a. Administrator invokes Add/Modify/Delete connection option from the Connection Manager.
 - b. Connection Manager updates associated configuration entry in the connPool.xml file.

Dynamic Fail Over On Reserve Session

A. TP Server Failure

A TP Server failure is detected and marked by the Failover Monitor Service, when/if the Failover Monitor Service is ENABLED. Once a TP Server has been detected as offline, then the Failover Monitor Service marks that Server as UNAVAILABLE. For efficiency reasons, the Failover Monitor Service will not retry an offline TP Server until a specified number of minutes (failoverRetryInterval) have elapsed. Once a failed TP Server is detected to be active, it is returned to the pool and marked AVAILABLE.

B. Host Connection or Host Failure

In addition to the connection to the TP Server failing, the connection between the TP Server and the host can fail or the host can go down.

1. If a TP Server has no available sessions because they are all in use, then a return code of -114 is returned and the TP Server remains AVAILABLE.
2. When no sessions are available due to losing connectivity to a host, then an error code of -126 (HOST_DOWN) is returned when reserving a session.
3. If a reserve is done using a TP Client Connector created for a specific pool, and the return code for HOST_DOWN is detected by the TP Client, an attempt is made to fail over to the next available TP Service configured in the pool. The TP Service is marked offline (UNAVAILABLE) and will remain so until the Failover Monitor Service checks that it is AVAILABLE.
4. A return code of 101 indicates to the application that the reserve was successful, but only after Failover occurred to a different TP Server.

Exception and Error Handling

1. **Invalid IP Address and Port.** The status will indicate that a server is not available. It is up to the user to determine whether to remove the connection from the pool.
2. **Invalid pool name.** If an attempt is made to connect to a TP Client using a pool that is not configured, then an error return code of -128 is returned to the calling application. This type of error could likely result from a typo in the name that is entered for the pool.
3. **Failover attempted.** If a reserve is attempted and no TP Server is available, an error return code of -127 is returned to the calling application.

4. **Sessions are HOST_DOWN.** An error return code of -126 is returned to the application when all sessions assigned to the transaction set requested by the Reserve call have lost host connectivity.
5. **Failover not initialized(return code -130).** No longer used.

Programming Considerations

If an application wants to use Failover, it must use the “pool” option when creating a TP Connector before doing a Reserve API Request. There is an internal table kept (copy of last updated connPool.xml file) by the TP Client containing which connections are AVAILABLE and which are UNAVAILABLE. A connection is marked AVAILABLE if a connect on the socket is successful. The Failover Monitor Service periodically checks the list of connections for which are AVAILABLE and which are not. The default time interval is 1 hour.

The time interval can be configured using the “Connection Manager”, which is part of the TP Admin.

It is recommended that the Fail Over Monitor Service be ENABLED, if an Application uses TP Connectors specifying the “pool” option, in order that the real time status of TP Servers is updated in a timely manner.

When the application issues a reserve when using a pool, the Connector within the TP Client looks at the primary connection (TP Service) in the connPool.xml file and checks the status (In Service or Unavailable). If the status is not In Service, the TP Client will check each secondary connection, in priority order, until an In Service connection is found. The reserve is then done using the first In Service connector found, for the pool being used. If no connection is available, then return code -127 is returned.

In addition to displaying the status of each connection, the Connection Manager web page is used to enable/disable the Fail Over Monitor Service, configure the Fail Over Retry Interval (seconds), and to add, modify, or remove a connection from a pool.

The Failover Monitor Service updates its internal list of connections (and the connPool.xml file whenever it needs to be changed) when it is initialized and during each periodic check for connection availability.

Following are recommendations:

- The basic TP Client (TPConnector) API functions (reserve, keepSession, addInput, getSessionId, getSessionStatus, runTransaction, getOutput, and release) have not changed. When a “new TPConnector(*poolName*)” is issued in the application, a request to the Connection Manager is made to obtain the IP address and port information for the first available TP Service connection configured in the connection pool.
- The application should create a new TP Connector(*poolName*) to process each customer call before doing the Reserve and other API requests to process the caller's request.
- Cleo recommends that Applications do NOT use multiple threads to access the same TP Connector with API requests. Applications using multiple threads to access the same TP Connector could result in the TP Server experiencing OutOfMemory errors or IndexOutOfBoundsExceptions.

Therefore, a new TPConnector should be created each time the application processes data using a Host Session. After the TPConnector is created, the first API Request must be a “reserve”. When processing of one or more runTransaction API requests is completed, the “release” API request should be done specifying

release()

which automatically releases the Host Session and disconnects the TP Connector Socket from the TP Server, or by specifying TRUE for the “disconnect” parameter.

```
release(disconnect);
```

Glossary

AID key: 3270 Attention Identification key, when pressed, causes data to be sent to the host. Examples include ENTER, PA1, PA2, PA3, PF1-PF24, CLEAR.

API: Application Program Interface. A set of protocols, routines and tools for building a software application.

Application: User application, an example being a voice response application which handles phone calls.

Attribute: In an XML document, a sub element defined within an element. In the following example, GENDER and AGE are attributes within the PERSON element:

```
<person gender="male" age="36">
  <firstName>Bob</firstName>
  <lastName>White</lastName>
</person>
```

Base Screen: Transaction screen containing text identifiers which can be used by other transaction screens for the purpose of screen identification.

Cleo TP: Transaction Processor. A java API and runtime component providing the core function that runs transaction sequences against a host application.

Definition Files (and Transaction Files): text files created in XML format by the Transaction Designer (TD). They contain screen definitions and transaction sequences defined using the TD. They are used by the Transaction Processor to navigate screens through a particular mainframe application.

Element: A logical structure in an XML document that is delimited by a start and an end tag. Also called an XML Object.

Emulator: 3270 terminal emulation for accessing mainframe applications.

Field Definition: Describes a location on the host screen that accepts input or returns output. Fields are defined in the Screen Definitions as *send* (input) fields and *receive* (output) fields.

HLLAPI (High Level Language Application Program Interface): an API defined by IBM that allows a server-based application to communicate with a mainframe computer. HLLAPI requires a server to run 3270 emulation software and then defines an interface between the application and the emulation software. Use of this API is referred to as *screen-scraping* because the approach uses characters that would otherwise be displayed on a terminal screen.

Host Screen: A 3270 or 5250 host application screen

Host transaction sequence: A representation of a series of host application screens. The following are pre-defined transaction names:

Login: used to log on to the host

Park: used to navigate to a specific host application screen

Logout: used to log off the host

Recovery: used to navigate from an unknown or erroneous application screen

Identifier: text on a 3270/5250 screen used to identify a particular screen sent from the Mainframe Host.

Input field: area on a 3270/5250 screen containing data sent to the host. Also referred to as toHost or send field.

IR: Interactive Response system, which automatically responds to telephone requests for information.

IR Application: Any application developed using IVR Designer or another IR development tool.

Keyboard Macro: variable text stored for each session in a table to be used as input in a transaction sequence. A column number references each entry when defining input field definitions. The value given in the keyboard macro will override any value that was defined in the transaction. Commonly used to specify a different logon id and/or passwords for each host session.

Keyboard Macro File: encrypted file containing values for macros that can be specified in the TD, providing for variable input data being sent to the host (such as logon ids and passwords)

Output field: area on a 3270/5250 screen containing data received from the host, to be sent to the IVR application. Also referred to as receive or fromHost field.

Recording clip: information describing a contiguous series of host screen images with associated keyboard input used to progress from one screen to the next. The last screen image in the clip does not have an associated AID key.

Re-Park transaction: a user-defined transaction name with a suffix of “_park”, such as “transaction1_park”. The re-park transaction is run by the plug-in after its companion user-defined transaction has been successfully run. See the “Cleo 3270 Plug-in Programmer’s Guide” or “Transaction Processor Programmer’s Guide” for a more detailed discussion.

Screen Definition: characteristics identifying a 3270 host application screen including locations on the screen to be used for identifying the screen (screen identifiers), input fields, and output fields. Screen definitions are defined in the TD.

Screen identifier: an area on the host screen image to be used in identifying the screen, specified by its start row and column, and text.

Sessions File: contains mappings for session id to transaction set name and a flag indicating if it is configured to be used (enabled).

Session / LU: A communication channel that allows data to be sent to and received from a host computer. Transaction sets must be assigned to sessions before any transactions between application scripts and a host computer can take place.

TD: Transaction Designer. A standalone GUI based utility used to create recording clips that are then used to define transaction sequences run by the Transaction Processor.

%TP_HOME%: The install path of the TP that can be determined by issuing “echo %TP_HOME%” from the Windows command line.

Transaction Designer (TD): A standalone GUI based utility used to create recording clips that are then used to define transaction sequences run by the Transaction Processor.

Transaction Files: see Definition Files.

Transaction Processor: see Cleo TP.

Transaction Sequence: a transaction sequence is defined in the Transaction Designer (TD), to be run later by the Transaction Processor. (Refer to the *TD User's Guide* for more information on creating transaction sequences, also referred to as transactions.) It represents a series of 3270 host application screens. Either a user-defined transaction or a pre-defined transaction can be created. The following are pre-defined transaction names:

- Login – used to log on to the host
- Park – used to navigate to a 3270 screen that the next transaction to run requires
- Logout – used to log off the host
- Recovery – used to navigate from an unknown or erroneous application screen back to a parked state

Transaction Set: defined and created by the TD. It consists of a directory containing transaction files and screen definitions that will be used during Transaction Processor execution. It can be associated with an existing or newly created user application and therefore have the same name.

Transaction Set directory: this directory is the top-level directory where all TD applications will be accessed and stored. Each transaction set is stored in a subdirectory, named according to the transaction set name, below the transaction set directory.

User-defined transaction: contains host transaction sequence that is run by the Transaction Processor when requested by the IR application to retrieve information from the host. An empty user-defined transaction named “transaction1” is created by the Transaction Designer.

VoiceXML: markup language for creating voice user interfaces. It uses speech recognition and/or touchtone (DTMF keypad) for input, and pre-recorded audio and text-to-speech synthesis (TTS) for output. Callers interact with VoiceXML applications via a VoiceXML “interpreter” (also known as a “browser”) running on a telephony server.

XML: short for *Extensible Markup Language*, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents.