Internal System Documentation (view as outline)

TAPE REJECT CLEARING (TRC)

1. Application Purpose and Overview

Some TAC insured companies send their group insurance premium payment with payment detail recorded on Tape, disc or cartridge electronic media to the Home Office. This payment information is formatted for entry into the LIPAS GL but in certain cases individual policy information may not be complete or does not match policies on file. Generally, these are policies that have been either terminated or do not yet exist in the LIPAS data structure. The ***Tape Reject Clearing*** application accepts rejected entries from this media, automatically bringing together the record which needs this information and the research data needed to complete the entry. A technician then compares the corrected entry from information supplied by TRC from the EDM databases and edits or accepts the changes. When the entry is complete, the information is re-submitted to the LIPAS GL through the IE3600 mainframe process.

1.1. Document approach

This document is Process oriented but certain diagrams and the Developer's section are object oriented.

1.2. Functionality (What is the software supposed to do?)

1.2.1. Summary of Major Functions

- Open Reject Files that contain data rejected by the LIPAS GL as incomplete.
- Create a data file of the rejected data.
- Locate and retrieve information contained in the EDM databases that relate to each rejected record.
- Display a screen showing the data retrieved with the rejected record.
- Allow a technician to accept or change information imported for the record.
- Create a file for inclusion in the daily IE3600 process for re-submitting the records.
- Create a History table of all TRC transactions.

1.2.2. Design constraints

1.2.2.1. Regulatory policies

N/A

1.2.2.2. Hardware limitations

N/A

1.2.2.3. External interfaces

While an external interface may be possible in the future it is not currently implemented nor contemplated.

1.2.2.4. Parallel operations

This Application is designed to be multi-user.

1.2.2.5. Audit functions

Completed transactions are held in a History table that is currently kept in perpetuity.

1.2.2.6. Control functions

N/A

1.2.2.7. Reliability

Application requires 100% accuracy. Consequently, the TRC Technician must always approve any changes made to the data suggested by the Application or edit the record manually.

1.2.2.8. Application criticality

This Application is not mission critical.

1.2.2.9. Safety and security considerations

Security is not currently implemented but is currently under development in the Phase II production version.

1.2.3. Integrity policies

The integrity of the data is in the hands of the Technician since the program can only suggest data that it has found in the EDM databases to fill in the missing information in a record.

1.2.4. Resource limits

N/A

1.2.5. Required Standards

TRC follows all established TAC Standards. It is a 3-tiered architecture.

1.2.6. Assumptions and dependencies

The Application assumes that the data that it receives in the Reject File is accurate and that the data from the EDM databases is accurate. It is dependent upon the TRC technician knowing what is correct.

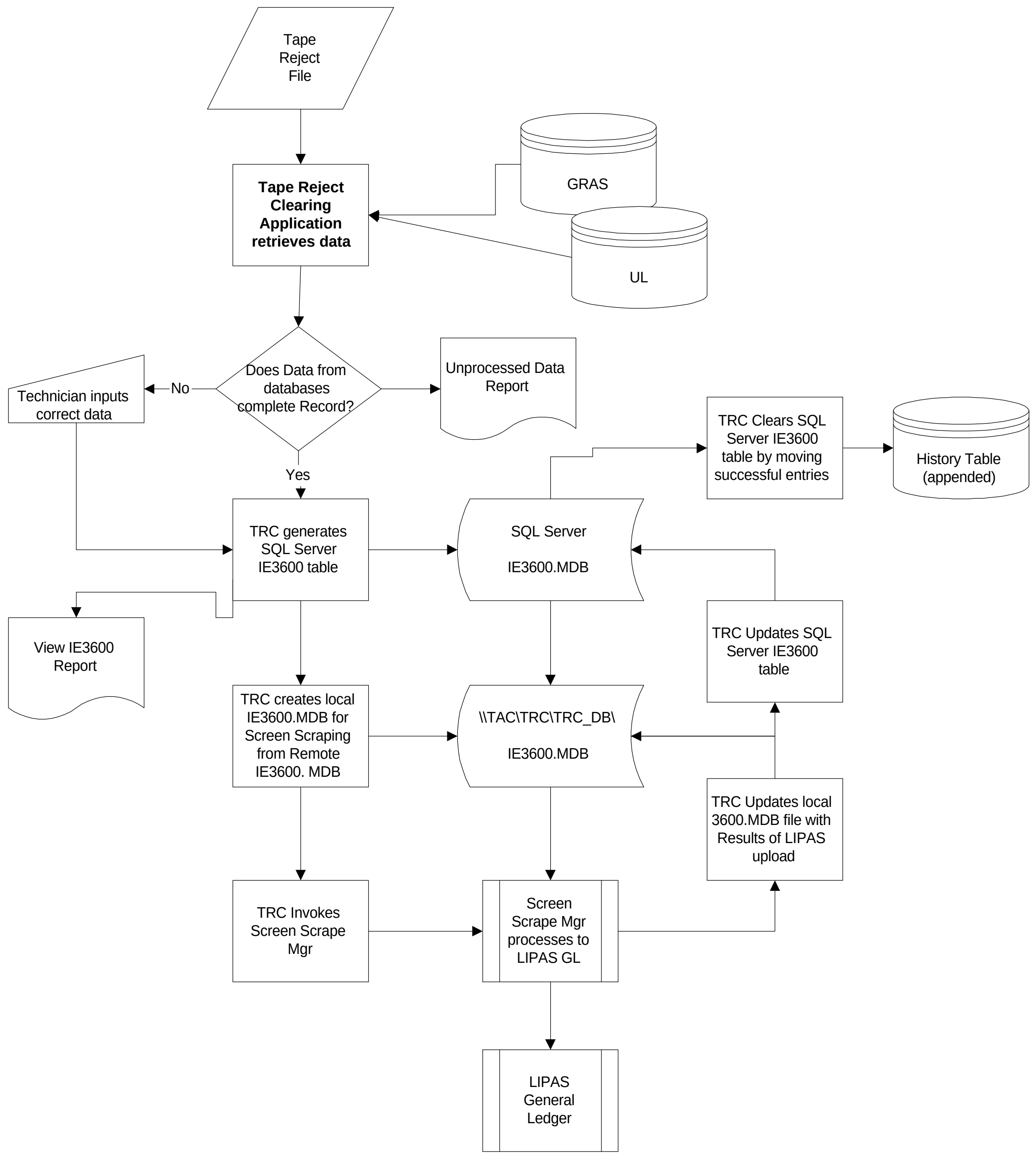
1.2.7. Other considerations

None

1.3. Diagrams

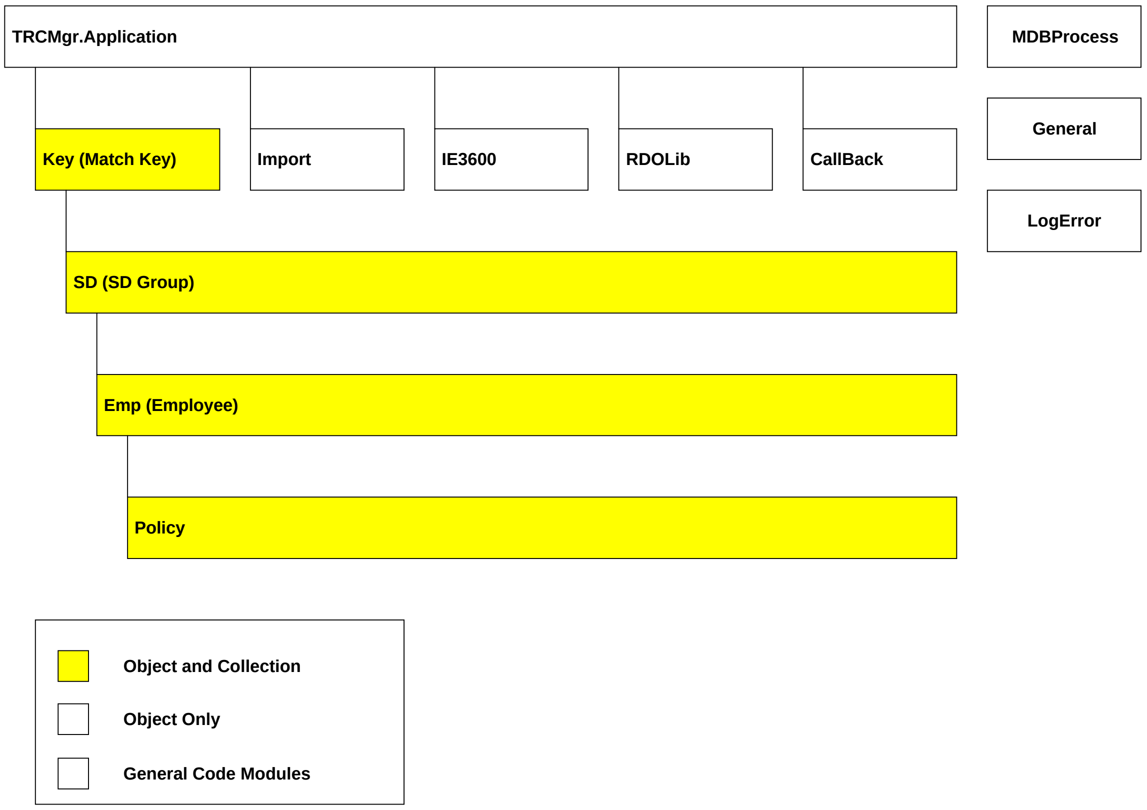
1.3.1. Data-flow

The following diagram illustrates the data flow into the Tape Reject Clearing Application.


Tape Reject File
GRAS
Tape Reject Clearing Application retrieves data
UL
Technician inputs correct data
No
Does Data from databases complete Record?
Unprocessed Data Report
Yes
TRC Clears SQL Server IE3600 table by moving successful entries
History Table (appended)
TRC generates SQL Server IE3600 table
SQL Server IE3600.MDB
View IE3600 Report
TRC Updates SQL Server IE3600 table
TRC creates local IE3600.MDB for Screen Scraping from Remote IE3600. MDB
\\TAC\TRC\TRC_DB\ IE3600.MDB
TRC Updates local 3600.MDB file with Results of LIPAS upload
TRC Invokes Screen Scrape Mgr
Screen Scrape Mgr processes to LIPAS GL
LIPAS General Ledger

1.3.2. Object Oriented

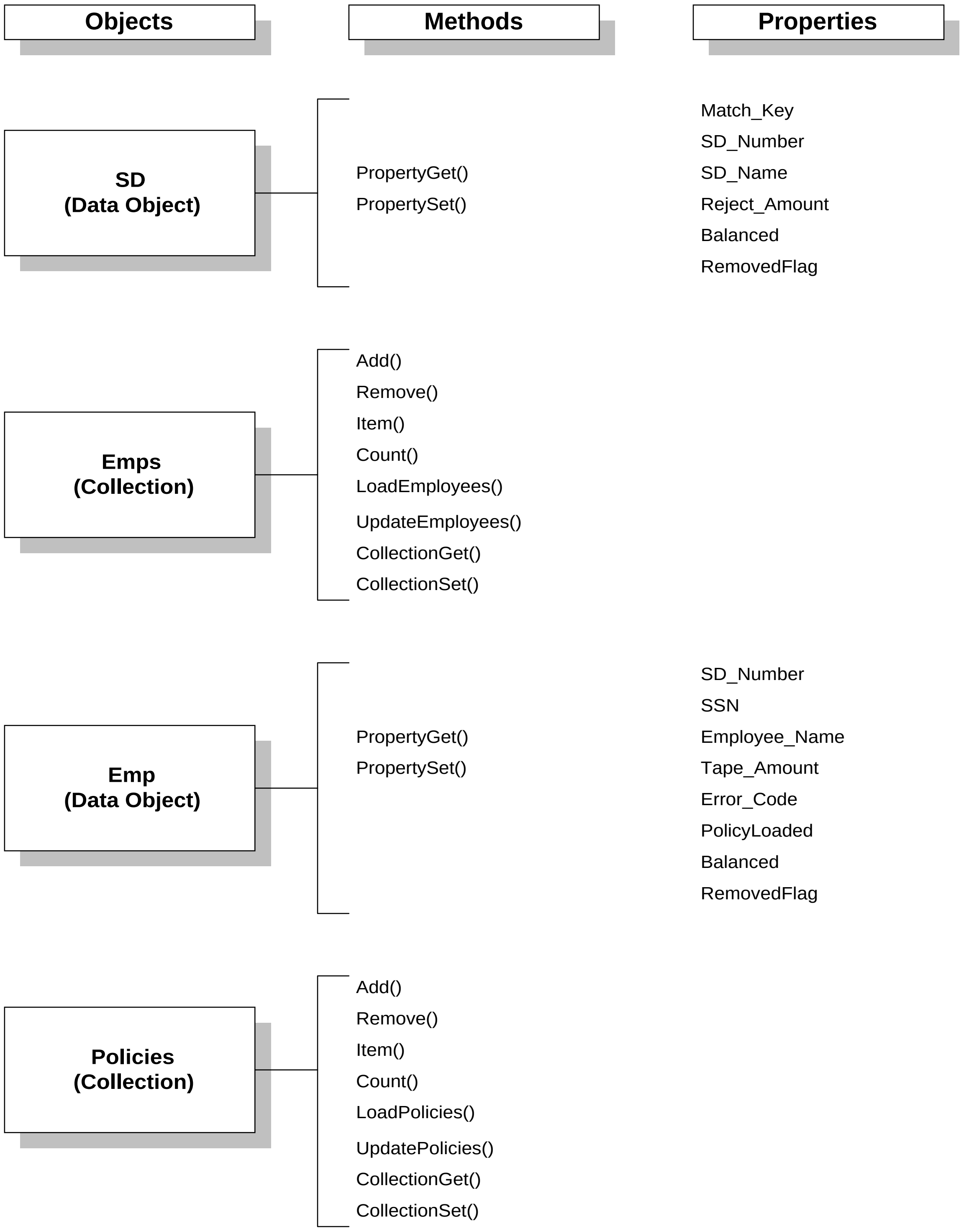
1.3.2.1. *Tape Reject Clearing* System Automation Manager (Tier-2) Object Model

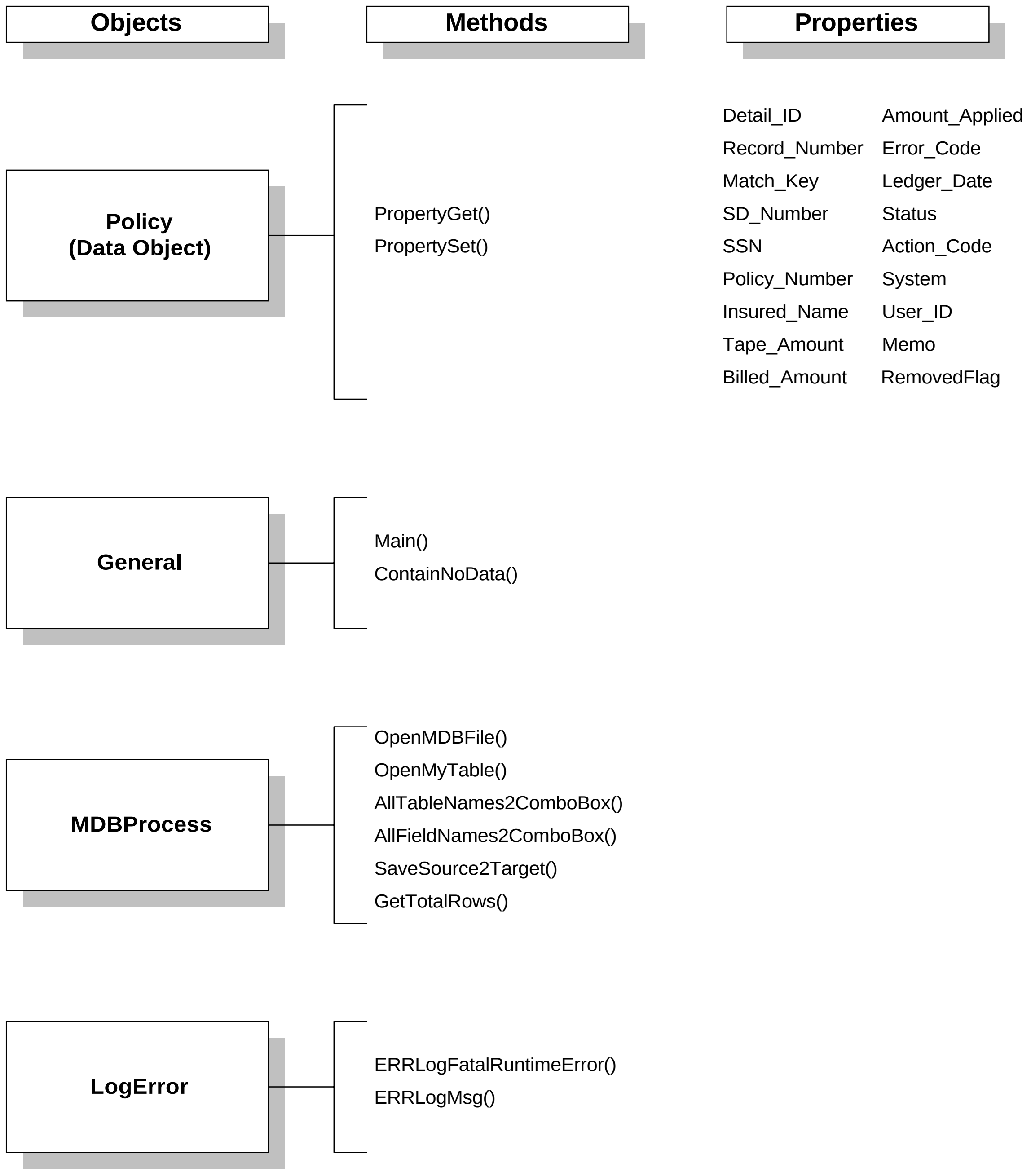


1.3.2.3. *Tape Reject Clearing* System Automation Manager (Tier-2) Object Methods and Properties.

Objects	Methods	Properties
TRCMgr.Application	Initialize()	ConnectString
	OpenAccessMDB()	IE3600_Table
	GetActionCode2Array()	Mainframe_ID
	GetErrorCode2Array()	Mainframe_PassWord
	GetPolicyStatus2Array()	UserID
	*GetUserSecurity()	Password
	*SaveUserSecurity()	*Security_ID
	ProRatePolicyList()	Session
	CloseAccessMDB()	TestMode
	Terminate()	
RDOLib	Initialize()	
	GetRDOResultset()	
	ExecuteRDOPStatement()	
	CloseQueryNResultset()	
	ObjectInCollection()	
	RegDSN()	
	SaveRDOSource2Target()	
CallBack	AddObjectReference()	
	DropObjectReference()	
	SetInterval()	
	PercentCompleted()	
Import	ImportTextFile()	ImportCompleted
	DuplicateMatchKey()	Match_Keys
	SaveMatchKey()	
	MMDDYYYY2Date()	
	GetMissingInfo()	
	GetColumnData()	
	RefreshMissingInfoFromDetail()	
	ProRateAmountApplied()	

* Not implemented yet


Objects
Methods
Properties
SD
(Data Object)
PropertyGet()
PropertySet()
Match_Key
SD_Number
SD_Name
Reject_Amount
Balanced
RemovedFlag
Emps
(Collection)
Add()
Remove()
Item()
Count()
LoadEmployees()
UpdateEmployees()
CollectionGet()
CollectionSet()
Emp
(Data Object)
PropertyGet()
PropertySet()
SD_Number
SSN
Employee_Name
Tape_Amount
Error_Code
PolicyLoaded
Balanced
RemovedFlag
Policies
(Collection)
Add()
Remove()
Item()
Count()
LoadPolicies()
UpdatePolicies()
CollectionGet()
CollectionSet()



1.3.2. Relational

None

1.4. References to other Documents

Document Name	Description
No other documents are referenced	

2. Administrative Information

2.1. Responsible Parties

2.1.1. Developer
Charles Luan

2.1.2. Maintenance
Charles Luan

2.1.3. Change Control
Maggie Lui, Charles Luan

2.1.4. Security
Security is not yet implemented.

2.2. Security

2.2.1. Physical Security

2.2.1.1. Servers
Company Production server

2.2.1.2. Media
N/A

2.2.1.3. Location

2.2.1.3.1. Who has location access
TAC Developers and TAC LAN Administrators have access to the Application.

2.2.2. Logical Security

2.2.2.1. User authorization

Currently, security in the form of User Authorization is not implemented. In the Phase II version of this Application, currently in development, access will be limited to Users who are on an Authorized User List that matches their Novell NetWare login name.

Allowed Name	Department	Description of rights

2.2.2.2. Usage log

All transactions are currently kept in perpetuity in a History table.

2.2.2.3. Audit trails

All transactions are currently kept in perpetuity in a History table.

2.2.2.4. Encryption

N/A

2.2.2.5. Data integrity check

No Data Integrity check is made by the Application. All such checks are the responsibility of the TRC Technician.

2.2.2.6. Communications restrictions

2.2.2.6.1. Internal

Currently, there are no restrictions except that the Application be installed on the User’s workstation.

2.2.2.6.2. External

There are no methods established to enable external operation of the Application.

2.2.3. Backup Files/Programs

2.2.3.1. Frequency

The LAN Administrator is responsible for assuring that system and file backups are done on a regular basis. Currently the TAC Novell servers and NT/SQL servers are backed up, differentially, nightly from Monday through Thursday nights, using the program BACKUP EXEC (Version 7.0) by Arcada. Full back ups of these servers are performed nightly Friday and Saturday, also using Arcada backup software.

2.2.3.2. Medium
Standard TAC procedures using 8mm Dat tape.

2.2.3.3. Storage of backups

2.2.3.3.1. Logical
Back ups are made on Dat cartridges in the Data Center on the 6th floor of the Broadway Transamerica building.

2.2.3.3.2. Physical
Server Backups are trucked to Arcus every 24 hours so backups will never be on the premises longer than 24 hours and usually considerably less. In case of catastrophic loss of data, backups can be retrieved from Arcus.

2.2.3.4. Documentation Backup
Specific procedures documented here and elsewhere regarding this application should be stored conveniently for immediate access, but should also be on file with the Central Technology Line (CTL) and stored off-site (e.g., at Arcus) for disaster recovery purposes.

Insure that up-to-date copies of this document and all critical system files are located:

- in the TAC LAN Administrator’s office,
- at Arcus, with Business Recovery documents
- with the CTL Helpline
- with the CTL Network Services Team
- with the user Project Manager
- electronically in SourceSafe with other application documentation

2.2.4. Problem Management

2.2.4.1. Detection
See Problem Management document (To Be Completed)

2.2.4.2. Identification
See Problem Management document (To Be Completed)

2.2.4.3. Containment
See Problem Management document (To Be Completed)
(as in the case of a virus)

2.2.4.4. Correction
See Problem Management document (To Be Completed)

2.2.4.5. Tracking
See Problem Management document (To Be Completed)

2.2.4.6. Logging

See Problem Management document (To Be Completed)

2.2.4.7. Other issues,

Currently, none

2.2.5. Change Control & Management

Refer to TAC document, “Change Control Guidelines”

2.2.5.1. File copies

Company Policy, “Change Control Guidelines”

2.2.5.2. Test versions

Company Policy, “Change Control Guidelines”

2.2.5.3. Testing of changes and enhancements

Company Policy, “Change Control Guidelines”

2.2.5.4. Version identification

Company Policy, “Change Control Guidelines”

2.2.5.5. Change log, audit trail

Company Policy, “Change Control Guidelines”

2.2.5.6. Change authority

Those individuals listed in section **2.1.3. Change Control** on page **8** must approve all changes that will effect users.

2.2.5.7. Change process

2.2.5.7.1. Communication of Changes

Company Policy, “Change Control Guidelines”

Business Recovery

For procedures during catastrophic business interruptions of TAC services, refer to the **BusRec.doc** document in the SourceSafe documentation directory. A copy of this plan is located in the LAN Administrator’s office and in any other area specified by our Business Recovery coordinator. Recovery for this application will follow procedures in this document. Where there is specific deviations from that plan, they will be written below.

Data protection

Company Policy, “BusRec.doc” above

Critical equipment

Off-site storage
Company Policy, "BusRec.doc" above

Recovery plan
Company Policy, "BusRec.doc" above

Process
Company Policy, "BusRec.doc" above

Location
Company Policy, "BusRec.doc" above

System Documentation
This document is essential to problem resolution, and can be helpful in identifying and isolating recurring problems. Specific procedures documented here and elsewhere regarding this application should be stored conveniently for immediate access, but should also be on file with the Central Technology Line (CTL) and stored off-site (e.g., at Arcus) for disaster recovery purposes.

Up to date copies of this document and all critical system files should be located:

- In a binder in the TAC LAN Administrator's office
- At Arcus
- With the CTL Helpline
- With the CTL Network Services Team
- Data Systems Project Manager's Office

System Environment

Hardware Requirements (supported)

Workstation
The Application assumes the TRC installation will be on a standard Workstation.

Special system configurations
The Terminal Program **EXTRA!,** the **ScreenScrape Manager,** and **Crystal Reports** must be installed on the Workstation.

2.3. Mainframe relationship

2.3.1. Mainframe Screens Used (Screen names)

2.3.1.1. System

Name	Function
None	

2.3.1.2. Transactions

Name	Function
IE3600	Send Corrected Bill Payment detail to LIPAS

2.4. Network Considerations

2.4.1. LAN protocols

IPX/SPX or TCP IP. No Name Pipes allowed.

2.4.2. Intranet protocols

None

2.4.3. Internet protocols

None

2.5. Operating System(s)

2.5.1. OS's which support application

Windows 95

2.5.2. Support needed

Standard Company Policy

2.6. Source Code

2.6.1. Language(s)

Visual Basic 4.0

2.6.2. Security

TAC uses a Microsoft software management tool called SourceSafe. Documents may only be checked out and changed by authorized users and information regarding the last user who changed the document is stored in the system. All Source Code is stored in designated SourceSafe Directories that are in turn back up during the standard TAC Systems back-up procedures.

2.6.3. Location

Source Code for Tape Reject Clearing source code is stored in the SourceSafe Production directory.

2.6.4 Source Code

Located in the SourceSafe production directory.

2.6.3.2. Shared Components

Application uses the Resize.ocx and Standard Developer controls.

2.6.3.3. Executable

The executable is stored in the TAC/TRC folder on the Standard Workstation desktop and uses Local OLE.

2.6.3.4. INI

The .ini file is kept in the same directory as the Executable in the TAC/TRC folder on the Standard Workstation.

2.6.3.5 History Table

The History table is stored on the SQL Server in the TapeReject database.

2.6.3.5. Reports

An EDM report and a TAC report are produced by the Application.

2.6.3.6. Documentation

This is the current Internal System Documentation for this Application and a User Guide is under development.

2.7. External Interfaces

2.7.1. People who use Application

Departments

Premium Accounting

2.7.2. System impact

2.7.2.1. Systems changed by Application

Name
LIPAS General Ledger

2.7.2.2. Application changed by other programs

Name
None

2.7.3. External communications

None

3. Applications System

3.1. Input/Output

3.1.1. Input

The Tape Reject Clearing Application receives its data input from a Tape Reject File which is compiled from transactions which did clear in the normal General Ledger upload process. TRC allows the technician to Browse to any file containing data in the proper format. File format must be in plain ASCII text with Fixed Length fields.

3.1.2. Output

The Tape Reject Clearing Application outputs the corrected data into a Microsoft Access database, the IE3600 table stored on the appropriate SQL Server. After TRC invokes the Screen Scrape Manager, this database creates a mirror of the IE3600.MDB file on the local workstation specifically for upload to the LIPAS GL through the IE3600 process. After the Screen Scrape process is complete, the local IE3600.MBD file is updated with the success or failure of the upload, which in turn updates the SQL Server version of this database. Records, which have been successfully uploaded, are moved from the SQL Server IE3600.MDB to History.MDB, where they are appended onto the existing data.

3.2. Developer Documentation

```
*************************************************************************************
 *                                                                                 *
*    TRCMgr.EXE - Tape Reject Clearing System Automation Manager (OLE Automation Server)    *
 *                                                                                 *
*************************************************************************************
```

TAC
System Date: 10/28/96
Document Date: 7/24/97
Written by Chuck Luan
****

+++++++++++++++++++++++
Purpose
+++++++++++++++++++++++

This ***Tape Reject Clearing*** Automation Manager (TRCMgr.EXE) is an OLE Server that serves TRC Related requests from client applications as a so called "Black Box" where client applications do not need to know HOW these black boxes do the task. The client applications only need to know WHAT functionality this OLE Server provides (i.e. what OLE Servers can do for them) and HOW it is accessed.

The purpose of this OLE Server is to define all of the business rules and logic related to ***Tape Reject Clearing*** processing to be used by the client applications. In this case, business rules and logic to be used by ***Tape Reject Clearing*** User Interface System, TRC.EXE.

+++++++++++++++++++++++
Theory
+++++++++++++++++++++++

The Architecture of this project is based on a coding technique called 3-Tier Implementation. It splits the Business Logic (anything related to the company's business rules such as premium calculations, billing process and data validations etc.) out from both the User Interface (Client screen or Tier-1) and the Data service (Database Server or Tier-3). This level is called the Business Service or Tier-2. This is the first 3-Tier project written for TAC.

3-Tier architecture provides a better way to organize and maintain our in-house applications. It makes all of our applications as Non-Software Tool Dependent as possible. For example, we can design our screens using visual tools such as Visual Basic or Powerbuilder while our logic may be partially or completely written in C++ or other tools that can be compiled into DLLs or OLE Server. We can also change our backend database from Access to SQL Server or to Oracle easily without affecting our User screens and Business logic. We can even put all of our Business Logic modules on the network as Remote OLE Automation Servers. This means more flexibility, efficiency, and control.

+++++++++++++++++++++++
3-Tier Architecture
+++++++++++++++++++++++

Tier-1: The User Interface

This service contains all graphical screens or code modules that relate to the user; front end interaction only. All forms, intrinsic controls, third party controls, display methods, user application option settings, general routines etc. should be put on this tier. No data logic process allowed. This service can be represented by applications written in any tools such as Visual Basic, MS Access, Powerbuilder or MS Excel etc. that provide a user interface to access standard OLE objects.

Tier-2: The Business Service

This service, theoretically, should contain no graphical objects such as forms, controls or message boxes etc. An exception may be something like a CallBack Form that contains a Timer to check the progress of a requested task. This tier is used only for objects and collections that contains all the company's business rules, calculations, Logic etc. and for the methods and properties that interact with the Tier-1 User interface and the Tier-3 backend Data Service. This 2nd Tier gets data from, and saves data to, the Tier-3 backend database, and sends and gets data stored in array structures to and from the Tier-1 User front end application.

Tier-3: The Data Service
This data service is actually the implementation of backend database's GET DATA and SAVE DATA processes by using SQL database's Stored Procedures. The Stored Procedures should only contain SQL Statements that perform retrieving, adding and updating data such as SELECT, INSERT and UPDATE etc. In order to maintain the integrity of 3-Tier architecture, the calculations and business logic built in the Stored Procedures are NOT allowed. The TRC Automation System use MS SQL Server as its backend database engine.

++++++++++++++++++++++++
Interface with OLE Server
++++++++++++++++++++++++

To use or, get connected to the TRC Automation Manager (OLE Server), the client application must first create and initialize the TRC Manager object.

To create an instance of the TRC Manager object, use the following lines:

```
    'Declare OLE object first
    Global TRCMgr As Object

    'Create an instance
Set TRCMgr = CreateObject("TRCMgr. Application")
```

After the OLE Server object is created, it must be initialized before client application can request the services. At this point, client application may pass additional information to the Server. This additional information is primarily used for security:

UserID:	The user ID of current calling client application.
Password:	The user password of current calling client application.
MainframeUserID:	The mainframe user ID of current calling client application user.
MainframePassword:	The mainframe user password of current calling client application user.

TestMode:	The mainframe's IMS processing mode. i.e. True (= -1) for "TESTIMS" and False (= 0) for "IMS"
Session:	Mainframe session to be used for screen scrapping.

To initialize TRC Manager, use:

```
        TRCMgr.Initialize UserID:=MyUserID, Password:=MyPassword, _
                MainframeUserID:=MyMainframeUserID,  _
                MainframePassword:=MyMainframePassword, _
                TestMode:=MyTestMode, Session:=MySession
```

Once the OLE Server object is initialized, you can start using it’s methods and properties, and the methods and properties of the objects under it.

A client application can use some of these methods to get Tape Reject data from the OLE Server:

```
  'Store Action Code data to array for reference in detail screen.
  aActionCode = TRCMgr.GetActionCode2Array

  'Store Policy Status codes and descriptions to array for reference in detail screen.
  aPolicyStatus = TRCMgr.GetPolicyStatus2Array

  'Call TRC OLE Server to start Importing reject data from a given reject file.
  TRCMgr.Import.ImportTextFile MyTextFile

  'Use the collection get method to obtain the Match key rows
  aMatchKey = TRCMgr.Keys.CollectionGet

  'Loads Match Key to array from OLE Server
  aSrchMatchKey = TRCMgr.Keys.LoadKeys2Array(Match_Key:=MyKey2Search)
```

```
******************************************************************************************
        ********************************************************************************
      *                                                                                   *
    *        TRCMgr OLE Server's General Modules, Objects and their Methods and Properties        *
      *                                                                                   *
    *************************************************************************************
    *************************************************************************************
```

```
****************************************
++++++++++++++++++++++++++++++++++++++++
                General.BAS
++++++++++++++++++++++++++++++++++++++++
****************************************
```

This module is the main module to be referenced when TRCMgr.exe is activated. It contains all general functions or methods that do not need to be instantiated as an object.

```
****************
*      Methods      *
****************
```

Main()

This is the first main module to be called in order to activate the TRCMgr.Exe

----

ContainNoData(ItemString As Variant) As Boolean

This function checks the passed parameter for valid data content. A NULL or empty string is invalid, and therefore a TRUE value will be returned to caller.

```
***************************************
++++++++++++++++++++++++++++++++++++
                    LogError.BAS
++++++++++++++++++++++++++++++++++++
***************************************
```

This module contains functions to track and log all errors that occur during TRCMgr OLE server processes.

```
****************
*      Methods      *
****************
```

ERRLogFatalRuntimeError(ByVal vsProcName As String, ByVal viErrCd As Integer)

This procedure will log any runtime errors to the log file that will be located in the user Windows directory, display a message to the user and end the application.

Parameters:

VsProcName:	Name of the procedure where the error occurred
viErrCd:	The Err value when the runtime error occurred

ERRLogMsg(ByVal vsProcName As String, Optional vsMsgText1 As Variant, Optional vsMsgText2 As Variant)

This procedure will write a message to the error log that is located in the users temp directory. If the file exists, the message will be appended to the current file, otherwise the file will be created.

Parameters:

vsProcName:	Name of the procedure where message is being written from
vsMsgText1:	The text that is to appear in the error log.
vsMsgText2:	The text that is to appear in the error log.

```
***************************************
++++++++++++++++++++++++++++++++++++
```

MDBProcess.BAS
++++++++++++++++++++++++++++++++++++
****
This module acts as a DAO and Jet Engine's wrapper to open, close a MS Access MDB database, open and close DAO's Workspace, Queries and Recordsets etc.

```
****************
*   Methods    *
****************
```

OpenMDBFile(sFullFileName As Variant)

This sub routine opens a user specified MDB file. The file name includes the Drive and Path to where the MDB is located.

OpenMyTable(sTableName As String, nOpenType As Integer) As Recordset

This function opens a table *sTableName* of the type *nOpenType* from current *MyDB* database. (e.g.
table = "table_name" type = dbOpenTable)

GetTotalRows(sTableName As String) As Long

This function retrieves the total number of rows in a table by firing a SQLPassThrugh query statement.

AllTableNames2ComboBox(cboTableName As ComboBox)

This Sub routine gets all table names from the current database and stores them in a ComboBox.

AllFieldNames2ComboBox(sMyTableName As String, cboFieldName As ComboBox)

This Sub gets all Column Names from current Table and stores them in a ComboBox.

SaveSource2Target(rsSource As Recordset, rsTarget As Recordset, Optional SkipColumns As Variant)

This method copies one row of data from Source recordset to Target recordset.

Parameter:
SkipColumns - Lets user specify columns not to be copied (skipped) into a string
separated with a ',' (comma).

CloseMDBFile()

This method closes the MS Access MDB file that caller specified when MDB file was opened.

****
++++++++++++++++++++++++++++++++++++
TRCMgr. Application

```
++++++++++++++++++++++++++++++++++++
****************************************
```

This object is the main interface and entry point to the OLE Automation Server. This object must be instantiated and activated before a client application can get access to its functional methods and properties. To get connected to this server, method Initialized() must be called.

```
 ****************
 *    Methods    *
 ****************
--------------------------------------------------------------------------------------------------------
  Initialized(Optional UserID As Variant, Optional Password As Variant, Optional Id As Variant, _
              Optional Parent As Variant, Optional MainFrameUserID As Variant, _
              Optional MainFramePassword As Variant, Optional TestMode As Variant, _
              Optional Session As Variant)
```

This procedure is called once when the server is instantiated. If it is instantiated by an Instance Manager, the Instance Manager will pass, the object Id and Parent data. If it is instantiated by a
client application, the client probably has no need to pass an Id or Parent. This demonstrates how this generic object shell can be used in an Instance Manager environment or in an environment where the client application is allowed to create an instance of the object directly.
Note: Currently, there is no Instance Manager being used, Because this TRCMgr.exe is a local OLE Server and therefore no need to use IM.

```
--------------------------------------------------------------------------------------------------------
  OpenAccessMDB(sFullFileName As Variant)
```

This Sub open TRC's MDB database file.

```
--------------------------------------------------------------------------------------------------------
  CloseAccessMDB()
```

This Sub close TRC's MDB database file.

```
--------------------------------------------------------------------------------------------------------
 * GetUserSecurity() As Variant     ===> (Not Implemented yet!)
```

This function gets current login user's security level, mainframe Id, and password. then stores to array and returns to caller.

```
--------------------------------------------------------------------------------------------------------
 * SaveUserSecurity(DataArray As Variant) As Variant     ===> (Not Implemented yet!)
```

This function saves current login user's security level, mainframe Id, and password to backend database.

```
--------------------------------------------------------------------------------------------------------
 GetActionCode2Array() As Variant
```

This function Retrieves Action codes from DB, stores them in an array, and returns them to caller.

```
--------------------------------------------------------------------------------------------------------
 GetErrorCode2Array() As Variant
```

This function Retrieves Error Codes from DB and stores them into array and return it to caller.

```
--------------------------------------------------------------------------------------------------------
```

GetPolicyStatus2Array() As Variant

This function Retrieves Policy Status codes and descriptions from DB and stores them in an array and returns them to caller.

ProRatePolicyList(PolicyArray As Variant, TotalPayment As Variant) As Variant

This function re-calculates all the policies' Amount Applied for current Employee. In most cases, this happens when user tried to zero out amount applied to one policy record, and we need to re-prorate all the other policies of the same Employee due to that change.

```
****************
*   Properties   *
****************
```

UserID:	Set or Get User ID who is currently running TRC client application and calling the OLE Manager.
Password:	Set or Get user Password of user who is currently running TRC client application and calling the OLE Manager.
Security_ID:	Set or Get the security ID in order to access to the OLE Manager.
ConnectionString:	Set or Get the ConnectionString that will be used to access TapeReject Database in SQL Server.
IE3600_Table:	Set or Get the name of IE3600 Table to be used to screen scrape LIPAS' IE3600 transaction.
Mainframe_ID:	Set or Get the LIPAS's mainframe User ID.
Mainframe_Password:	Set or Get the LIPAS's mainframe User Password.
Session:	Set or Get the LIPAS's mainframe Session to be used for screen scraping.
TestMode:	Set or Get the LIPAS's mainframe session's processing mode. For real data or for Test only.

```
****************************************
++++++++++++++++++++++++++++++++++++
                    RDOLib
++++++++++++++++++++++++++++++++++++
****************************************
```

This object is the wrapper for Remote Data Object (RDO). The wrapper simplifies interaction with the RDO interface. When methods and objects in RDO are modified or added, no RDO function changes need to be made throughout the code modules. All that needs to change are the RDO functions in the wrapper. This will keep the application very flexible, clean, and easy to maintain.

```
****************
*    Methods   *
****************
```

Initialize(Optional Default_User As Variant, Optional Default_Password As Variant, Optional Connect _
As Variant, Optional ServerName As Variant)

This method sets up the RDO Engine and stores default user Id and password data, then creating a connection to the backend database server. Note that in the connection process, we use a so-called *DSN-Less* connection method, which uses the Database Server's driver to connect the backend directly rather than using the traditional Data Source Name(DSN).

RegDSN(Server As Variant) As Boolean

Registers a datasource if it does not already exist. Sends it the Server name.

GetRDOResultSet(QueryName As String, SQLString As String, Optional OpenType As Variant, _
Optional LockType As Variant, Optional ParameterArray As Variant) As _
rdoResultset

This method fires a SQL query (Stored Procedure) to retrieve resultset using RDO's PreparedStatement.

++
TCL 970211:
Modified to accept sp call that has no parameters.
++
Definitions:

QueryName:	Current PreparedStatement's name
SQLString:	Stored Procedure string, e.g. "{call sp_Find_TapeMatchKey (?)}"
OpenType:	Type of Resultset to be returned. e.g. rdOpenForwardOnly (default), rdOpenStatic, rdOpenKeyset/Dynamic etc.
LockType:	Type of concurrency control. If you don't specify a locktype, rdConcurReadOnly is assumed. e.g. rdConcurLock, rdConcurReadOnly, rdConcurRowver or rdConcurValues.
ParameterArray:	Array that stores all parameters needed for stored procedure passed to this function call

----
ExecuteRDOPStatement(QueryName As String, SQLString As String, Optional ParameterArray As _
Variant) As Variant

This method fires a SQL query (Stored Procedure) to execute a query using RDO's PreparedStatement. This method returns number of rows affected by the SQL execute operation (i.e. Insert, Update or Delete statements etc.)

++++++++++++++++++++++++++++++++++++
TCL 970211:
Modified to accept sp call that has no parameters.
++++++++++++++++++++++++++++++++++++
Definitions:

QueryName:	Current PreparedStatement's name
SQLString:	Stored Procedure string, e.g. "{call sp_Add_Tape (?,?,?,?,?)}"
ParameterArray:	Array that stores all parameters needed for stored procedure passed to this function call

----

SaveRDOSource2Target(rsSource As rdoResultset, QueryName As String, SQLString As String, _
Optional SkipColumns As Variant)

This method copies one row of data from the Source RDO resultset to the Target table.

Parameter:
SkipColumns - Lets user specify columns not to be copied (skipped) into a string
separated by a ',' (comma)

Note: the column name in SkipColumn will be compared with Source's columns, not Target's!!
Therefore, there is no need to specify Target's unique Id column (e.g. IE3600_h_id) because there is NO such Id in the Source table (e.g. Source table IE3600 has "IE3600_id", not "IE3600_h_id" from target table IE3600_h.)

ObjectInCollection(ObjectToSearch As Object, ObjectName As String) As Boolean

This function search through the built-in data object collection to find out if a given object existed in this collection

----

CloseQueryNResultset(Optional QueryName As Variant, Optional SQLString As Variant) As Variant

This function search through built-in RDO data object collection to find out if a given object such as a PreparedStatement or Resultset object exists in their collections. If one exists, then close it and return a TRUE flag to confirm close.

****
++++++++++++++++++++++++++++++++++++
CallBack
++++++++++++++++++++++++++++++++++++
****

This object allows caller application to add a reference to this caller object so that, by calling back caller application's UpdateProgressBar() method, it can call back to caller application to indicate the percentage completion of a task being processed in the OLE Automation Server, In this case, the TRCMgr.EXE

****
*   Methods   *

```
 ****************
-----------------------------------------------------------------------------------------------------------------------
 AddObjectReference(Caller As Object) As Boolean

   This procedure add a reference to the caller object so after callback object activates a timer, it can call
   back to caller to indicate the percentage completion of current task process of an OLE Server by calling
   caller object's UpdateProgressBar() method.

-----------------------------------------------------------------------------------------------------------------------
 DropObjectReference(Caller As Object) As Boolean

   This procedure terminates the connection link of current OLE Server's processing status between
   Callback object and Caller object.
-----------------------------------------------------------------------------------------------------------------------
 SetInterval(iInterval As Integer) As Boolean

   This procedure sets callback object's timer Interval property to a given value that caller object
   specified.
-----------------------------------------------------------------------------------------------------------------------

 ****************
 *    Properties    *
 ****************

 PercentCompleted:
   This property tells caller the percentage completion of current process loop (based 0-100%)
   Note!  This property is NOT USED by TRCMgr.EXE Currently, the value of nPercentCompleted will
   be passed back to the caller application by using parameter:PercentCompleted in caller application's
   object method, UpdateProgressBar().
-----------------------------------------------------------------------------------------------------------------------

****************************************
++++++++++++++++++++++++++++++++++++
                    Import
++++++++++++++++++++++++++++++++++++
****************************************
```

This object downloads Tape Reject Text file and based on these Employee SSN level records, retrieve missing information such as Policy number, insured name etc. from EDM and create Policy level Reject detail records for each Employee SSN found in tape reject so later can be used to adjust the Action Codes (PAY,SUSP , etc.) by the technician and screen scrape back to the LIPAS mainframe system and therefore clear out these suspense reject records.

```
 ****************
 *     Methods     *
 ****************
-----------------------------------------------------------------------------------------------------------------------
 ImportTextFile(TextFileName As Variant) As Boolean

   This function reads a text file specified by user, and stores data record onto a given table.
```

DuplicateMatchKey(Match_Key As String) As Boolean

This function find if there is any duplicate Match Key in Tape table. If found, this means current Tape file run that contain this Match Key had already been run and processed. If not found, go check Tape History file to make sure that this match key had not been processed and archived to the history table.

SaveMatchKey(sMatch_Key As String, dProcess_date As Date, nTapeTotal As Currency)

This method saves current Match Key to Tape level table in SQL Server. Note that this is a private method, used only by Import object.

RefreshMissingInfoFromDetail(Optional Match_Key As Variant)

This method Re-Retrieves missing information such as Policy Number, SSN, and Insured Name from EDM backend without re-building Policy-Level detail records.

Logic:
Retrieves all detail records that contain no valid data such as empty Status, missing Policy Number and Amount Applied etc. then go to lookup EDM database for these missing info, save them to detail record if found in EDM. If the Status column of current detail record contain no data (which means can not find any data from EDM or other system), and more than one rows are being retrieved from EDM, then we will need to build new detail record for each extra rows just retrieved.

GetColumnData(ColumnName As Integer, rsGetData As rdoResultset, rsReject As rdoResultset) As Variant

This is a private function to get data we wanted from either TapeReject (TapeRejectDetail) or EDM's resultset.

GetMissingInfo(Optional Match_Key As Variant)

This method Builds the Reject Detail records from Tape Reject table and Retrieves missing information such as Policy Number, SSN, and Insured Name from EDM backend. This function also does the calculation to prorate the Payment received from tape file to Amount Applied column on all policies detail records.

Error Code definitions:
"006" -No matching Due Date (TCL 970118 added. No action on this yet!)
"007" (Old "001") - Missing Policy Number
"009" (Old "002") - No matching Payment
"011" (Old "003") - Policy terminated
"014" (Old "004") - Policy not on LIPAS
"013" (Old "005") - Negative Payment

Constant variables to be used:
```
Const NO_POLICY_NUMBER = "007"      '    "007" (Old "001") - Missing Policy Number
Const NO_MATCH_PAYMENT = "009"      '    "009" (Old "002") - No matching Payment
Const POLICY_TERMINATED = "011"     '    "011" (Old "003") - Policy terminated
```

```
    Const POLICY_NOT_ON_LIPAS = "014"    '       "014" (Old "004") - Policy not on LIPAS
    Const NEGATIVE_PAYMENT = "013"       '       "013" (Old "005") - Negative Payment

--------------------------------------------------------------------------------------------------------------------
 ProRateAmountApplied(Optional Match_Key As Variant, Optional Process_Status As Variant)

   This function allocates Payment received from tape file to Amount Applied column on all policies
   detail records

   Amount Applied Allocation logic:
      If Payment received = Sum of all Billed Amount on Policies found,
         then for each policy record, Amount applied = Billed Amount (PPP Amount)
      If Payment received <> Sum of all Billed Amount on Policies found,
         then for each policy record, Amount applied = ( Billed <PPP> Amount / Total Billed <PPP>
         Amount ) * Reject Tape Amount

   TCL 970114:
     Changed to use RDO to retrieve newly created detail records by calling stored procedure other than
     using recordset
     as a parameter.
--------------------------------------------------------------------------------------------------------------------

 ****************
 *    Properties     *
 ****************
 Match_Keys:
   This property returns an array of Match Keys that were input from current Tape file run.
 ImportCompleted:
   This property tells caller if current Tape Import process is completed successfully or not.

--------------------------------------------------------------------------------------------------------------------

****************************************
+++++++++++++++++++++++++++++++++++
                    IE3600
+++++++++++++++++++++++++++++++++++
****************************************
```

This object allows user to do all the preparations related to the IE3600 screen scrape processes in LIPAS system. After setting Action Codes in Reject Detail processing screen, user can call the following methods to prepare or update the IE3600 result data so that the Screen Scrape Manager (SSMgr.EXE, another OLE Automation Server) can be called to do the actual IE3600 screen scrape through LIPAS system:

Use BuildIE3600Table() method to build the IE3600 result data ready for screen scraping IE3600 through LIPAS.

Use LoadIE36002Array() method to load and return the IE3600 result data into an array so user can display on the screen or edit the data.

Use IE3600Data2MDB() method to download IE3600 Result data to a given MS Access MDB so it can be used by Screen Scrape Manager (SSMgr.Exe) to do the screen scrape IE3600.

Use UpdateIE3600Data() method to upload the IE3600 result data that have been screen scraped in the given MDB to TRC database in SQL Server.

Use SynchronizeProcessStatus() to synchronize Process Status column used to flag whether current IE3600 screen scrape process is succeeded (status = 1) or failed (status = -2).

Use GetIE3600ColumnNames() method to obtain all column names to be used in IE3600 screen scraping.

Use ClearRejectData() method to archive, clear and update all level reject items that have been cleared through IE3600 screen scraping. Note that the only exception is the Action Code setting to "SUSP" (Suspension), which even though is processed through LIPAS IE3600 screen scrape, WILL NOT be cleared from the Reject Database, because it means to suspend the reject item and KEEP it in the bulk balance (TRC database).

```
****************
*    Methods    *
****************
```

BuildIE3600Table(Optional Match_Key As Variant)

This method generates IE3600 table ready for screen scraping IE3600 transaction. If the Match Key is not passed, ALL SD Group records of ALL Match Keys will be processed.

+++++ Important ! ! ! +++++
Note that for each Match Key + SD Group to be processed, if there is any NULL or Empty Action Code encountered in the detail record, then the IE3600 table for this Key+Group will NOT be built since this method would not be able to know what "action" must be taken to screen scrape IE3600 for the record in this Key+SDGroup.

LoadIE36002Array(Optional Match_Key As Variant) As Variant

This function Retrieves IE3600 table (which is ready for screen scraping) from DB and stores them into array and return it to caller.

IE3600Data2MDB(Optional Match_Key As Variant) As Variant

This function Retrieves IE3600 table (which is ready for screen scraping) from SQL Server and stores them into IE3600 table in TRC.MDB so later can be used to actually do the screen scraping task.

UpdateIE3600Data(Optional Match_Key As Variant) As Variant

This function Retrieves IE3600 table (which is ready for screen scraping) from TRC.MDB and store the changed columns back to SQL Server 's IE3600 table.

GetIE3600ColumnNames() As Variant

This function Retrieves IE3600 table's column names for screen scraping and stores them into an array.

ClearRejectData(Optional Match_Key As Variant)

This sub clears the Tape Reject data from both Tape and Tape Reject Detail tables based on the IE3600 records that have been screen scraped completely and successfully. This includes generating history records in ALL History tables (Tape, Tape Reject, Reject Detail and IE3600 transaction etc.)

The steps to clear the reject data are as the follows:

1. Before start getting clearing IE3600 records, Make sure you have call "Synchronize_ProcessStatus" method to synchronize those records that failed and did not get Process Status field updated to "Failed" (-2). =====> See Modifications
2. Open all related tables
3. Start checking the process status of IE3600 transaction table. If succeeded for current SD Group (by checking the process status of the last row of this SD Group), copy these records to history tables and update the $ Amount remained in the Tape table and clear (delete) Reject Detail records (the rows that have gone through the IE3600 transaction successfully) from its table.

++++++++++++
Modifications:
++++++++++++
TCL 970611: Synchronize function -- TRCMgr.IE3600.SynchronizeProcessStatus is now called from ScreenScrapeIE3600() instead of ClearRejectProc in the User Interface program --- TRC.ExE.

TCL 970710: Modified to filter out IE3600 rows that are marked for suspension (Action Code = 'SUSP') i.e. After IE3600 screen scrape through LIPAS, the suspended rows should be kept in the Bulk Balance in the TRC database until they are set to other action codes such as 'PAY','REF' etc. and re-written to LIPAS again.

SynchronizeProcessStatus()

This sub synchronizes the process status that were flagged during IE3600 Screen Scraping based on the same Match key and SD Number. Because the IE3600 Translation is a batch process that update many IE3600 transaction records at one time, and therefore there would be only one record (the LAST one that has VoucherYN = "Y") getting updated to "failed" (-2) IF this Match Key+SD Number's batch process Failed (process status=-2) This sub flags all records back to Failed (-2) if one of their sibling (the one with VouchYN='Y') is flagged failed.

++++++++++++
Modifications:
++++++++++++
TCL 970611: Changed to update Match Key level only. No more SD Number level synchronization.

****
+++++++++++++++++++++++++++++++++++
Keys

```
++++++++++++++++++++++++++++++++++++
****************************************
```

This object does the Match Key-Level data processing. This includes adding, updating and deleting new Keys (Match Keys) from Key collection, Loading keys into Key collection and array etc. This object also creates its related new SD object and collections when loading Key items or updates related SD objects when updating Key items back to Key object and collections. Notes that this object is a collection, not a data object!

```
 ****************
 *    Methods   *
 ****************
```

Add(Optional Match_Key As Variant, Optional Vol_SerialNumber As Variant, _
Optional Process_Date As Variant, Optional Tape_Amount As Variant, _
Optional RemovedFlag As Variant) As Object

With the data passed to it, this function adds a new Key data object to the Key collection.

Remove(Key As Variant)

This sub does not actually remove the Key, just marks it to be removed. This allows the client-application to "undo", and enables the collection to remember which Key to delete.

Count() As Variant

This function counts number of Key objects in the Key collection and returns total count to the caller

---

LoadKeys(Optional Match_Key As Variant)

This sub retrieves Match Keys from database based on the Key parameter passed to it and stores these keys into Key collection. The related SD Groups for each Key retrieved are also retrieved from database and stored into SD Group collection by activating Key's SD object and calling SD object's LoadSDs() method.

LoadKeys2Array(Optional Match_Key As Variant) As Variant

This function is similar to LoadKeys() method but instead of loading Match Key to collection, it retrieves Match Key from database based on the Match Key parameter passed to it and stores these Match Keys into Match Key array and return this array back to the caller.

Item(Key As Variant) As Object

This function returns a given Key data object currently stored in the Key collection to the caller.

CollectionGet() As Variant

This function stores all Match Key in the Match Key collection into Match Key Array and their related SD Groups in the SD Group collection into SD Group Array.

CollectionSet(KeyArray As Variant)

This function saves all Match Keys in the Match Key array into Match Key collection and their related SD Groups in SD Group array into SD Group collection.

UpdateKeys()

This sub updates Match Keys from Key collection to backend database. It also triggers the series nested update from its related SD Numbers, Employees and Policies collections to backend.

+++++
Note:
+++++
For now, ONLY Policy data objects in Policy collection will be truly updated back to database. Since the Match Key-Level, SD Group-Level and the Employee-Level data objects are stored in the collections just for the purpose of being displayed on detail editing screen for reference, and they are NOT "true entity" in backend database (i.e. the TapeReject database is not normalized enough and even it is, there is no need to construct tables for these entity levels. Because we do not allow users to edit, update Match Key, SD Group, and Employee data in this system. In other words, there is no "Editing Screen" for these data level!

```
***************************************
+++++++++++++++++++++++++++++++++++
                        Key
+++++++++++++++++++++++++++++++++++
***************************************
```

This object is used as a data object for storing Match Key level information. This Key object can be instantiated by setting a new object reference to it. For example, use the codes below, you can create a new Key data object (an instance) ready for saving Match Key data to it:

```
Dim NewKey As Key
Set NewKey = New Key
```

After Key data object is created and data is stored in it, you can add this Key data object into KeyS object's Key Collection (Note! it is NOT the Key data object of this discussion! Please see the KeyS object above this Key object.) by calling Add() method in Keys object:

```
e.g.  TRCMgr.Keys.Add NewKey, CStr(NewKey.Match_Key)
```

Since this object is a data object for storing Key data, it contains only two methods, PropertyGet() and PropertySet() to send or update Key information (roughly corresponding to the columns of Match Key's table, "Tapes" in the TRC database) stored in it in one single call.

```
****************
*      Methods     *
****************
```

```
PropertyGet(Optional Match_Key As Variant, Optional Vol_SerialNumber As Variant, _
            Optional Process_Date As Variant, Optional Tape_Amount As Variant, _
```

Optional RemovedFlag As Variant)

This method retrieves a given Match Key information (table columns) from its Key data object in the Key collection.

```
PropertySet(Optional Match_Key As Variant, Optional Vol_SerialNumber As Variant, _
            Optional Process_Date As Variant, Optional Tape_Amount As Variant, _
            Optional RemovedFlag As Variant)
```

This method updates a given Match Key information (table columns) back to its Key data object in the Key collection.

```
****************
*    Properties     *
****************
```

Match_Key: Get or set Match Key value of a given Key data object
Vol_SerialNumber: Get or set Vol_SerialNumber value of a given Key data object
Process_Date: Get or set Process_Date value of a given Key data object
Tape_Amount: Get or set Tape_Amount value of a given Key data object
RemovedFlag: Get or set RemovedFlag value of a given Key data object

----

```
****************************************
++++++++++++++++++++++++++++++++++++
                    SDs
++++++++++++++++++++++++++++++++++++
****************************************
```

This object does the SD Group (Employer) Level data processing. This includes adding, updating and deleting new SD Group from SD Group collection, Loading SD Group into SD Group collection and array etc. This object also creates its related Employee object and collections when loading SD Group items; or updates related Employee objects when updating SD Group items back to SD Group object and collections. This "SDs" object is a collection, and "SD" is a data object!

```
****************
*     Methods     *
****************
```

```
Add(Optional Match_Key As Variant, Optional SD_Number As Variant, _
      Optional SD_Name As Variant, Optional Reject_Amount As Variant, _
      Optional RemovedFlag As Variant, Optional Balanced As Variant) As Object
```

With the data passed to it, this function adds a new SD Group data object of a given Match Key into the SD Group collection.

Remove(Key As Variant)

This method does not actually remove the SD Group, just marks it to be removed. This allows the client-application to "undo" the changes, and enables the collection to remember which SD Group to delete.

Count() As Variant

This function counts number of SD Group data objects in the SD Group collection and returns total count to the caller

----

LoadSDs(Optional Match_Key As Variant)

This method retrieves a given Match Key's SD Groups from database based on the Match Key parameter passed to it and stores these SD Groups into SD Group collection. The related Employees for each of these SD Groups are also retrieved from database and stored into Employee collection by activating SD Group's Employee object (Emps) and calling Employee object's LoadEmployees() method.

Item(Key As Variant) As Object

This function returns a given SD Group data object currently stored in the SD Group collection to the caller.

CollectionGet() As Variant

This function stores all SD Groups in the SD Group collection into SD Group Array and their related Employees in the Employee collection into Employee Array.

CollectionSet(SDArray As Variant)

This function saves all SD Groups in the SD Group array into SD Group collection and their related Employees in the Employee array into Employee collection.

UpdateSDs()

This method updates SD Groups from the collection back to database. It also triggers the series of nested update from its related Employees and Policies collections to database.
[Note!!!!]
For now, ONLY Policy data objects in Policy collection will be truly updated back to database. Since the Match Key-Level, SD Group-Level and the Employee-Level data objects are stored in the collections just for the purpose of being displayed on detail editing screen for reference, and they are NOT "true entity" in backend database (i.e. the TapeReject database is not normalized enough and even it is, there is no need to construct tables for these entity levels. Because we do not allow users to edit, update Match Key, SD Group, and Employee data in this system. In other words, there is no "Editing Screen" for these data level!

```
***************************************
+++++++++++++++++++++++++++++++++++++++
                    SD
+++++++++++++++++++++++++++++++++++++++
***************************************
```

'This object is used as a data object for storing SD Group level information. This SD Group data object (SD) can be instantiated by setting a new object reference to it. For example, use the codes below, you can create a new SD data object (an instance) ready for saving SD Group data to it:

```
Dim NewSD As SD
Set NewSD = New SD
```

After SD data object is created and data is stored in it, you can add this SD data object into SDs object's SD Group Collection (Note!! It is NOT the SD data object! Please see the "SDs" object above this "SD" Group object.) by calling Add() method in SDs object:

```
TRCMgr.Keys.Item(KeyIndex).SDs.Add NewSD, CStr(NewSD.SD_Number)
```

Note: The object index ID can be changed to CStr(NewSD.Match_Key+NewSD.SD_Number) to avoid SD Group-Level duplication in the future improvement.

Since this object is a data object for storing SD Group data, it contains only two methods, PropertyGet() and PropertySet() to get or update all SD Group data items in one single call to these methods. User can also use SD Group data item's properties to retrieve or update each of these SD Group data items separately. To get all the properties in SD Group data object in one single call, use lines below:

```
TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).PropertyGet
                    Match_Key:=Match_Key,  SD_Number:=SD_Number, SD_Name:=SD_Name, _
                    Reject_Amount:=Reject_Amount, RemovedFlag:=RemovedFlag,, _
                    Balanced:=Balanced
```

To update all the properties in SD Group data object in one single call:

```
TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).PropertySet
                    Match_Key:=Match_Key,  SD_Number:=SD_Number, SD_Name:=SD_Name, _
                    Reject_Amount:=Reject_Amount, RemovedFlag:=RemovedFlag,, _
                    Balanced:=Balanced
```

To get or update individual SD Group data property:

```
MySD_Number = TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).SD_Number     (to Get)
TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).SD_Number = MySD_Number     (to Update)
```

```
****************
*    Methods   *
****************
```

```
PropertyGet(Optional Match_Key As Variant, Optional SD_Number As Variant,
            Optional SD_Name As Variant, Optional Reject_Amount As Variant, _
            Optional RemovedFlag As Variant, Optional Balanced As Variant)
```

This method retrieves a given SD Group information from its SD Group data object in the SD Group collection.

```
PropertySet(Optional Match_Key As Variant, Optional SD_Number As Variant, _
            Optional SD_Name As Variant, Optional Reject_Amount As Variant, _
            Optional RemovedFlag As Variant, Optional Balanced As Variant)
```

This method updates a given SD Group information back to its SD Group data object in the SD Group collection.

```
****************
*   Properties   *
****************
```

Match_Key - Get or set Match Key value of a given SD Group data object
SD_Number - Get or set SD Group Number value of a given SD Group data object
SD_Name - Get or set SD Group Name value of a given SD Group data object
Reject_Amount - Get or set Reject_Amount value of a given SD Group data object
Balanced - Get or set Amount Applied Balanced flag of a given SD Group data object
RemovedFlag - Get or set RemovedFlag value of a given SD Group data object

----

```
****************************************
++++++++++++++++++++++++++++++++++++
                 EMPs
++++++++++++++++++++++++++++++++++++
****************************************
```

This object does the Employee-Level data processing. This includes adding, updating and deleting new Employee from Employee collection, Loading Employee into Employee collection and array etc. This object also creates its related Policy data object and collections when loading Employee data; and saves related Policy data objects when updating Employee data back to Employee object and collections.
This "EMPs" object is a collection, and "EMP" is a data object!

```
 ****************
 *    Methods    *
 ****************
```

```
Add(Optional SD_Number As Variant, Optional SSN As Variant, _
    Optional Employee_Name As Variant, Optional Tape_Amount As Variant, _
    Optional Error_Code As Variant, Optional RemovedFlag As Variant, _
    Optional Balanced As Variant) As Object
```

With the data passed to it, this function adds a new Employee data object of a given SD Group into the Employee collection.

Remove(Key As Variant)

This method does not actually remove the Employee, just flags it to be removed. This allows the client-application to "undo" the changes, and enables the collection to remember which Employee to delete.

Count() As Variant

This function counts number of Employee data objects in the Employee collection and returns total count to the caller

----

LoadEmployees(Optional Match_Key As Variant, Optional SD_Number As Variant)

This method retrieves Employees of a given Match Key+SD Group from database based on the Match Key and SD Group parameter passed to it and stores these Employees into Employee collection. The related Policies for each of these Employees are also retrieved from database and stored into Policy collection by activating Employee's Policy object (Policies) and calling Policy object's LoadPolicies() method.

Item(Key As Variant) As Object

This function returns a given Employee data object currently stored in the Employee collection to the caller.

CollectionGet() As Variant

This function stores all Employees in the Employee collection into Employee Array and their related Policies in the Policy Collection into Policy Array.

CollectionSet(EmpArray As Variant)

This function saves all Employees in the Employee array into Employee collection and their related Policies in Policy array into Employee collection.

UpdateEmployees()

This method updates Employees from the collection back to database. It also triggers the nested update to Employee's Policies from Policy collections to database.
[Note!!!!]
For now, ONLY Policy data objects in Policy collection will be truly updated back to database. Since the Match Key-Level, SD Group-Level and the Employee-Level data objects are stored in the collections just for the purpose of being displayed on detail editing screen for reference, and they are NOT "true entity" in backend database (i.e. the TapeReject database is not normalized enough and even it is, there is no need to construct tables for these entity levels. Because we do not allow users to edit, update Match Key, SD Group, and Employee data in this system. In other words, there is no "Editing Screen" for these data level!

```
****************************************
++++++++++++++++++++++++++++++++++++
                    EMP
++++++++++++++++++++++++++++++++++++
****************************************
```

This object is used as a data object for storing Employee level information. This Employee data object (EMP) can be instantiated by setting a new object reference to it. For example, use the codes below, you can create a new EMP data object (an instance) ready for saving Employee data to it:

```
    Dim NewEMP As EMP
    Set NewEMP = New EMP
```

After EMP (Employee) data object is created and data is stored in it, you can add this EMP data object into EMPs object's Collection (Note!! This is EMP --- a data object, not EMPs object! Please see the "EMPs" object above this "EMP" object.) by calling Add() method in EMPs object:

```
  TRCMgr.Keys.Item(KeyIndex).SDs.Item(SDIdx).Emps.Add NewEmp, CStr(NewEmp.SSN) &
                                                          NewEmp.Error_Code
```

Since this object is a data object for storing Employee data, it contains only two methods, PropertyGet() and PropertySet() to get or update all Employee data properties in one single call to these methods. The purpose of these methods is to allow caller to retrieve or update all of the data object's values (or called properties) in one single call. By doing so, caller can improve the network traffic speed (because there is only one single call instead of many calls for each property values, and each call from caller application to OLE Server is counted for 2-way network traffic!) and therefore it's very useful for implementation of Remote OLE Automation Server.

'To get all the properties in Employee data object in one single call, use lines below:

```
  TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).PropertyGet  _
             SD_Number:=SD_Number, SSN:=SSN, Employee_Name:=Employee_Name, _
             Tape_Amount:=Tape_Amount,  Error_Code:=Error_Code, Balanced:=Balanced
```

To update all the properties in Employee data object in one single call:

```
  TRCMgr.Keys.Item(KeyIdx).SDs.Item(KeyIdx).Emps.Item(EmpIdx).PropertySet  _
             SD_Number:=SD_Number, SSN:=SSN, Employee_Name:=Employee_Name, _
             Tape_Amount:=Tape_Amount,  Error_Code:=Error_Code, Balanced:=Balanced
```

User can also use Employee data properties to retrieve or update each of these Employee data values individually. But this will take more network traffic than one single call methods mentioned above.

To get or update individual Employee data property:

```
  MyEmpSSN = TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).SSN    (to Get)
  TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).SSN = MySSN        (to Update)
```

++
  Notes for PropertyGet() and PropertySet() methods
++
Although caller application can use the property methods mentioned above, they are somewhat difficult to use because of their long reference notation. Caller can not avoid this simply because this system is based on a hierarchical object model --- When you need to refer to a data object that is under its parent object, you will need to refer to its parent object first before you can refer to it. Fortunately, in most cases, caller application do not need to use these property methods directly to retrieve or update data. There are methods in the collection object (e.g. EMPs) that are able to call these property methods and handle data update for caller application

automatically. For example, when caller application call Match Key level methods, TRCMgr.Keys.CollectionGet and TRCMgr.Keys.CollectionSet to retrieve or update Match Keys data, these methods will actually trigger a series of PropertyGet() and PropertySet() calls to get or update the data objects, such as SD Group, Employee and Policy data objects under this Match Key data object.

```
 ****************
 *      Methods      *
 ****************
--------------------------------------------------------------------------------------------------------------------------
  PropertyGet(Optional SD_Number As Variant, Optional SSN As Variant, _
                 Optional Employee_Name As Variant, Optional Tape_Amount As Variant, _
                 Optional Error_Code As Variant, Optional RemovedFlag As Variant, _
                 Optional PolicyLoaded As Variant, Optional Balanced As Variant)

    This method retrieves a given Employee information (i.e. data properties) from its Employee data
    object in the Employee collection.
--------------------------------------------------------------------------------------------------------------------------
  PropertySet(Optional SD_Number As Variant, Optional SSN As Variant, _
                 Optional Employee_Name As Variant, Optional Tape_Amount As Variant, _
                 Optional Error_Code As Variant, Optional RemovedFlag As Variant, _
                 Optional PolicyLoaded As Variant, Optional Balanced As Variant)

    This method updates a given Employee information (i.e. data properties) back to its Employee data
    object in the Employee collection.
--------------------------------------------------------------------------------------------------------------------------


 ****************
 *    Properties     *
 ****************
  Although these properties are supposed to be read-only, they are designed to allow updating the values in
  case we need to do so in the future improvement!

  SD_Number - Get or set SD Group Number value of a given Employee data object
  SSN - Get or set Employee's SSN value of a given Employee data object
  Employee_Name - Get or set Employee Name value of a given Employee data object
  Tape_Amount - Get or set total Tape Reject Amount for a given Employee data object
  Error_Code - Get or set Error Code for each Employee data object.
  Balanced - Get or set Amount Applied's Balanced flag of a given Employee data object
  PolicyLoaded - Get or set flag to indicate whether the Policies of current Employee have been loaded or
    not.
  RemovedFlag - Get or set RemovedFlag value of a given Employee data object

--------------------------------------------------------------------------------------------------------------------------


****************************************
+++++++++++++++++++++++++++++++++++++++
                           Policies
+++++++++++++++++++++++++++++++++++++++
****************************************
```

'This object does the Policy-Level data processing. This includes adding, updating and deleting new Policies from Policy collection, loading Policies into Policy collection and array etc. This Policy collection object is the bottom-most object in the TRCMgr's hierarchical object model. This "Policies" object is a collection, and "Policy" is a data object!

```
*********************
*     Methods      *
*********************
```

```
Add(Optional Detail_ID As Variant, Optional Record_Number As Variant, _
      Optional SD_Number As Variant, Optional SSN As Variant, Optional Tape_Amount As Variant,
      Optional Policy_Number As Variant, Optional Billed_Amount As Variant, _
      Optional Match_Key As Variant, Optional Ledger_Date As Variant, _
      Optional Error_Code As Variant, Optional Action_Code As Variant, _
      Optional Amount_Applied As Variant, Optional Memo As Variant, _
      Optional User_ID As Variant, Optional Insured_Name As Variant, _
      Optional System As Variant, Optional Status As Variant, Optional RemovedFlag As Variant)
```

With the data passed to it, this function adds a new Policy data object of a given Employee into the Policy collection.

Remove(Key As Variant)

This method does not actually remove the Policy, just flags it to be removed. This allows the client-application to "undo" the changes, and enables the collection to remember which Policy data object to delete.

Count() As Variant

This function counts number of Policy data objects in the Policies collection and returns total count to the caller

---

```
LoadPolicies(Optional Match_Key As Variant, Optional SD_Number As Variant, _
             Optional SSN As Variant)
```

This method, based on the Match Key, SD Group and Employee data parameters passed to it, retrieves all policies of a given Match Key+SD Group+Employee (SSN) from database and stores these Policies into Policies collection.

Item(Key As Variant) As Object

This function returns a given Policy data object currently stored in the Policies collection to the caller.

CollectionGet() As Variant

This function stores all Policies in the Policies collection into Policy Array.

CollectionSet(PolicyArray As Variant)

This function saves all Policies in the Policy array into Policies collection.

UpdatePolicies()

This method saves the data changed in Policies collection back to TRC database.
[Note!!!!]
For now, ONLY Policy data objects in Policy collection will be truly updated back to database. Since the Match Key-Level, SD Group-Level and the Employee-Level data objects are stored in the collections just for the purpose of being displayed on detail editing screen for reference, and they are NOT "true entity" in backend database (i.e. the TapeReject database is not normalized enough and even it is, there is no need to construct tables for these entity levels. Because we do not allow users to edit, and update Match Key, SD Group, and Employee data in this system. In other words, there is no "Editing Screen" for these data levels!

```
****************************************
++++++++++++++++++++++++++++++++++++
                    Policy
++++++++++++++++++++++++++++++++++++
****************************************
```

This object is used as a data object for storing Policy level information. This Policy data object can be instantiated by setting a new object reference to it. For example, use the codes below, you can create a new Policy data object (an instance) ready for saving Policy data to it:

```
Dim MyPolicy As Policy
Set MyPolicy = New Policy
```

After Policy data object is created and data is stored in it, you can add this Policy data object into Policy Collection by calling Add() method in Policies object:

```
MyEmployees.Add MyPolicy, CStr(MyPolicy.Detail_ID) & CStr(MyPolicy.Policy_Number)
```

Since this object is a data object for storing Policy data, it contains only two methods, PropertyGet() and PropertySet() to get or update all Policy data properties in one single call. The purpose of these methods is to allow caller to retrieve or update all of the data object's values (properties) in one single call. By doing so, caller can improve the network traffic speed (because there is only one single call instead of many calls for each property values, and each call from caller application to OLE Server is counted for 2-way network traffic!) and therefore it's very useful for implementation of Remote OLE Automation Server.

To get all the properties in Policy data object in one single call, use lines below:

```
TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).Policies.PropertyGet _
                Detail_ID:=Detail_ID, Record_Number:=Record_Number,  _
                SD_Number:=SD_Number, SSN:=SSN, Tape_Amount:=Tape_Amount, _
                Policy_Number:=Policy_Number, Billed_Amount:=Billed_Amount, _
                Match_Key:=Match_Key, Ledger_Date:=Ledger_Date, Error_Code:=Error_Code,  _
                Action_Code:=Action_Code, Amount_Applied:=Amount_Applied, _
                Memo:=Memo, User_ID:=User_ID, Insured_Name:=Insured_Name,   _
                System:=System, Status:=Status, RemovedFlag:=RemovedFlag
```

To update all the properties in Policy data object in one single call:

```
TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).Policies.PropertySet _
                Detail_ID:=Detail_ID, Record_Number:=Record_Number,  _
                SD_Number:=SD_Number, SSN:=SSN, Tape_Amount:=Tape_Amount, _
                Policy_Number:=Policy_Number, Billed_Amount:=Billed_Amount, _
                Match_Key:=Match_Key, Ledger_Date:=Ledger_Date, Error_Code:=Error_Code,  _
                Action_Code:=Action_Code, Amount_Applied:=Amount_Applied, _
                Memo:=Memo, User_ID:=User_ID, Insured_Name:=Insured_Name,   _
                System:=System, Status:=Status, RemovedFlag:=RemovedFlag
```

User can also use Policy data properties to retrieve or update each of these Policy data values individually. But this will take more network traffic than one single call methods mentioned above.

To get or update individual Policy data property:

```
MyInsuredName =
 TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).Policies(PolicyIdx).InsuredName
 (to Get)

TRCMgr.Keys.Item(KeyIdx).SDs.Item(SDIdx).Emps.Item(EmpIdx).Policies(PolicyIdx).InsuredName
  = InsuredName       (to Update)
```

++
Notes for PropertyGet() and PropertySet() methods
++

Although caller application can use the property methods mentioned above, they are somewhat difficult to use because of their long reference notation. Caller can not avoid this simply because this system is based on a hierarchical object model --- When you need to refer to a data object that is under its parent object, you will need to refer to its parent object first before you can refer to it. Fortunately, in most cases, caller application do not need to use these property methods directly to retrieve or update data. There are methods in the collection object (e.g. Policies) that are able to call these property methods and handle data update for caller application automatically. For example, when caller application call Match Key level methods, TRCMgr.Keys.CollectionGet and TRCMgr.Keys.CollectionSet to retrieve or update Match Keys data, these methods will actually trigger a series of PropertyGet() and PropertySet() calls to get or update the data objects, like SD Group, Employee and Policy data objects etc. under this Match Key data object.

```
*****************
*     Methods     *
*****************
```

```
 PropertyGet(Optional Detail_ID As Variant, Optional Record_Number As Variant, _
             Optional SD_Number As Variant, Optional SSN As Variant, _
             Optional Tape_Amount As Variant, Optional Policy_Number As Variant, _
             Optional Billed_Amount As Variant, Optional Match_Key As Variant, _
             Optional Ledger_Date As Variant, Optional Error_Code As Variant, _
             Optional Action_Code As Variant, Optional Amount_Applied As Variant, _
             Optional Memo As Variant, Optional User_ID As Variant, _
```

```
                    Optional Insured_Name As Variant, Optional System As Variant, _
                    Optional Status As Variant, Optional RemovedFlag As Variant)
```

This method retrieves a given Policy data properties from Policy data object in the Policies collection.

```
  PropertySet(Optional Detail_ID As Variant, Optional Record_Number As Variant, _
                    Optional SD_Number As Variant, Optional SSN As Variant, _
                    Optional Tape_Amount As Variant, Optional Policy_Number As Variant, _
                    Optional Billed_Amount As Variant, Optional Match_Key As Variant, _
                    Optional Ledger_Date As Variant, Optional Error_Code As Variant, _
                    Optional Action_Code As Variant, Optional Amount_Applied As Variant, _
                    Optional Memo As Variant, Optional User_ID As Variant, _
                    Optional Insured_Name As Variant, Optional System As Variant, _
                    Optional Status As Variant, Optional RemovedFlag As Variant)
```

This method updates a given Policy data properties back to Policy data object in the Policies collection.

```
 ****************
 *    Properties     *
 ****************
```

Although these properties are supposed to be read-only, they are designed to allow updating the values in case we need to do so in the future improvement.

Detail_ID - Get or set Detail ID value of a given Policy data object.
Record_Number - Get or set Record_Number value of a given Policy data object
Match_Key - Get or set Match_Key value of a given Policy data object
SD_Number - Get or set SD Group Number value of a given Policy data object
SSN - Get or set Employee's SSN value of a given Policy data object
Tape_Amount - Get or set Tape Reject Amount for a given Policy data object
Policy_Number - Get or set Policy Number for each Policy data object.
Billed_Amount - Get or set Billed Amount for each Policy data object.
Ledger_Date - Get or set Ledger Date for each Policy data object.
Error_Code - Get or set Error Code for each Policy data object.
Action_Code - Get or set Action Code for each Policy data object.
Amount_Applied - Get or set Amount Applied for each Policy data object.
Memo - Get or set Memo for each Policy data object.
User_ID - Get or set User ID for each Policy data object.
Insured_Name - Get or set Insured Name value of a given Policy data object
System - Get or set System for each Policy data object.
Status - Get or set Status for each Policy data object.
RemovedFlag - Get or set RemovedFlag value of a given Policy data object

```
-------------------------------------------------------------------------------------------------------------------------
*************************************************************************************
*************************************************************************************
*                         END OF TRCMgr.EXE SYSTEM DOCUMENTATION                                  *
*************************************************************************************
```

3.5. Data structures

3.5.1. Map of Directory Structure (Local Workstation)

Relative Directory	Files (Contents)
Tac\Trc	Trc.exe
	Trc.ini
Tac\Trc\Tr_db\	Tr_db.mdb
Shared Components	**Version and use**
Standard Developer Controls	
Reports	Path
Printed or saved at Users discretion	

3.5.3.1. Application changed by Systems

3.5.3.1.1. Databases accessed by TRC

The following Databases and their Tables are accessed by as Read Only by TRC.

3.5.3.1.1.1. GRAS Database

TABLE	FIELD
Family	FamilyID
	PayorSSN
	PayorLastName
	PayorFirstName

TABLE	FIELD
Insured	InsuredType
	SSN
	LastName
	FirstName

3.5.3.1.2. UL Database

TABLE	FIELD
ULPolicy	ULPolicyNo
	Premium
	PolicyStatusID

TABLE	FIELD
PolicyStatus	PolicyStatusID
	Description

4. Glossary, (Definitions, Acronyms, and abbreviations)

5. Appendix 1 - Databases accessed by Application