The present invention is a method and apparatus which is used to enable application developers to generate workflow enabled applications which request services from the workflow server component of the workflow system, remotely and indirectly using messaging, shared databases or inter-process communications. The present invention provides a standard transaction format (STF) for accessing such a workflow system through STF processors via messaging, updates to the shared databases or inter-process communications. Workflow enabled applications are used by users to act and participate in business processes and enable users and managers to observe and query the status of workflows and business processes.

11 Claims, 7 Drawing Sheets
These links indicate that one of two conditional workflows is initiated.

Fig. 1
INCOMING

TRANSPORTER

TRANSACTION PARSER

TRANSACTION FORMATTER

TRANSACTION PROCESSOR

WORKFLOW SERVER APIS

OUTGOING

Fig. 4
Fig. 5
Legend:

- **A → B** Class A is derived from Class B (A is-a B)
- **A ↔ B** Object of Class A has a Pointer to an object of Class B (A has-a B)
- **A ↔ B** Class A instantiates Class B

![Diagram with classes and objects](image)

**Note:** Type of Transactions
- **Init:** InitBP, InitWF
- **Act:** Act
- **Get:** GetWFStatus, GetAvaHBPs, GetPendingActions
- **Return:** ReturnWFStatus

**Fig. 6**
Fig. 7

ACTION BASED TRANSACTIONS

WORKFLOW ENABLED APPLICATION

ERROR (RESPONSE)

ACT INITIATE BOUND DATA (WFSTATUS)

STF PROCESSOR

WS TxAPI (UPDATE TxDB)

TRANSACTIONS DATABASE

QUERY BASED TRANSACTIONS

WORKFLOW ENABLED APPLICATION

ERROR (REQUEST)

WF STATUS PENDING ACTIONS AVAILABLE BPs (RESPONSE)

STF PROCESSOR

WS TxAPI (READ TxDB)

TRANSACTIONS DATABASE

RETURN WORKFLOW STATUS BASED TRANSACTIONS

WORKFLOW ENABLED APPLICATION

ERROR (RESPONSE)

NOTIFY FOLLOW-UP REMINDER ERROR

STF PROCESSOR

WS STFQAPI (READ STFQ)

STF QUEUE

TRANSACTIONS DATABASE

Fig. 7
A fundamental concept of a workflow system is that any business process can be interpreted as a sequence of basic transactions called workflows. Every workflow has a customer, a performer, and conditions of satisfaction. The customer and performer are roles that participants can take in workflows. In addition, each workflow can have observers.

In a workflow, the customer is the person for the sake of whom the work is done, either because they made a request or accepted an offer. It is customers who are responsible for evaluating performed work and determining whether this work meets their conditions of satisfaction.

The performer is the person who is responsible for completing the work and for declaring to the customer when the work is done.

Requests and Offers are the two basic types of workflows. There are other workflow types such as Question, Inform and Note that are simplified derivations of Request and Offer. The conditions of satisfaction specify the work to be performed by the performer. In a request, the customer specifies the conditions of satisfaction, and in an offer the performer specifies them. (Then, of course, the two can enter into negotiation about the work to be done.) For example, given the sentence:

"John asked Frank to prepare the report and deliver it by noon on Friday."

Given the sentence:

"John proposed to prepare the report and deliver it by noon on Friday for Frank."

Business process maps display the workflows as loops, and display the relevant information about each workflow—the customer, the performer, the conditions of satisfaction and the cycle time. FIG. 1 is a business process map having a primary workflow 11, conditional workflows 13 and 15, a conditional link 17, parallel workflows 19 and 21, serial workflows 23 and 25. In a workflow system of the type used in conjunction with the present invention, associated with each workflow are various parameters such as roles, cycle time, conditions of satisfaction or associated semantics to the links that imply automated action or provide the framework for application building, all of which are necessary to create a useful business process representation.

Each workflow has four phases as shown in FIG. 2. The first phase is called the proposal phase during which a request is made of the prospective performer by a customer or an offer to a customer is made by a prospective performer. The second phase is called the agreement phase during which the offer is accepted by the customer or the request is agreed to by the performer and conditions of satisfaction are identified. Of course, during the agreement phase the original conditions of satisfaction can be negotiated by the customer and performer until an agreement is reached. The third phase is called the performance phase during which the performer undertakes to meet the agreed to or accepted conditions of satisfaction. When the performer believes that the conditions of satisfaction have been met, the performer declares completion. The last phase is the satisfaction phase during which the customer determines whether or not the conditions of satisfaction have been met by the performer, and if so, declares satisfaction.
A workflow system incorporates the following components which are shown in FIG. 3, a workflow server and databases, application program interfaces (APIs) and workflow server manager. In addition, a complete workflow system of the type in which the standard transaction format (STF) processors of the present invention may be utilized includes an application builder, analyst, workflow enabled applications and reporter components. The application builder, analyst, workflow enabled applications and reporter components, while useful components of a complete workflow system, do not form part of the present invention and details concerning such components are set forth herein only as needed for an understanding of the invention. The present invention is concerned mainly with STF processors used in combination with a complete workflow system.

A workflow system provides certain services as follows:

transactions services which are those related to initiating and acting in workflows by users and agents;
definition services which are those related to defining the elements of a business process and its workflows and workflow links;
names and routing services which are those related to defining organizational roles and identities;
configuration services which are provided to the system administrator through a specific configuration database;
scheduling services which allow an authorized user to create, modify and delete records of scheduled business processes; and
STF processing services which are provided by the server to STF processors (which are the subject of the present invention as described below) through an STF queue database.

Further details concerning the definition services, names and routing services, configuration services and scheduling services are set forth in co-pending U.S. Ser. No. 08/014,796 filed Feb. 8, 1993. The present invention is directed to the STF processing services provided by a workflow system as well as STF processors.

In addition to the foregoing services provided by a workflow system, external interfaces to the system provide services that are used by end-user applications, the workflow application builder, the workflow reporter and the STF processors.

A workflow system utilizes a workflow server which concentrates workflow operations in the server rather than in the end user applications.

All work done by the server is performed by one of three processes which are referred to as the transaction manager, follow-up manager and date/time schedule manager. Processes are software components or tasks that arearchitected to run as separate entities from each other. The workflow server controls the three basic processes based upon workflow server administration data in a configuration database in the following manner. First, it determines when to run the transaction manager and spawns that process. Second, it determines when to run the follow-up manager and spawns that process. Third, it determines when to run the date/time schedule manager and spawns that process.

These processes may be separate executables or simply separate tasks within the body of the workflow system server.

A workflow system also utilizes:

a definitions database which contains records of the definitions of the organizations, business processes, workflows, roles, and acts;
a transactions database which contains records of the enactment of workflows;
enabled applications which request services from the workflow server component of the workflow system, remotely and indirectly using messaging, shared data, and inter-process communications. The present invention provides a standard transaction format (STF) for accessing such a workflow system through STF processors via messaging, updates to the shared data, or inter-process communications. Workflow enabled applications are used by users to act and participate in business processes and enable users and managers to observe and query the status of workflow and business processes.

In describing the invention, the following terms with their indicated definitions are used:

**Act**
Basic linguistic occurrence by which people intervene in moving a workflow towards completion.

**Agreement**
The outcome of negotiation phase, in which two parties come to a common agreement of the conditions of satisfaction.

**Business Process**
A network of workflows linked together that represent the recurring process by which an organization performs and completes work, delivers products and services, and satisfies customers.

**Business Process Map**
This is a graphical representation of business process, which shows its workflows and relationships.

**Primary Workflow**
This is the first workflow which is initiated when a business process is initiated. Its condition of satisfaction is satisfied by the condition of satisfaction of the business process.

**Conditional Link**
A link that indicates that only one of a group of workflows will be triggered based on some condition.

**Conditions of Satisfaction**
Conditions declared by or agreed to by a customer. The fulfillment of which is the purpose of a workflow.

**Customer**
The role in a workflow who makes a request or accepts and offers.

**Customer Satisfaction**
The objective of a workflow, the accomplishment of which is declared by the customer when the conditions of satisfaction in the workflow have been fulfilled.

**Cycle Time**
A measure of time from initiation to successful completion of a workflow phase, a complete workflow, or a business process.

**Exception Flow**
The path in the business process workflow map which is followed if a customer cancels or a performer revokes or declines.

**Link**
A defined dependency between two workflows and the mechanism by which dependencies between workflows are established.

**Loops (Workflow)**
A workflow is represented graphically by an elliptical loop with arrows shown in a clockwise direction wherein each quadrant of the ellipse signifies different phases of the workflow.

**Normal Flow**
This is the path followed in a business process map when workflows complete with customer satisfaction.

**Observer**
A role in a workflow who cannot perform acts in the workflow, but is informed of acts in the workflow, and has access to the information and data associated with the workflow.

**Offer**
The act by which the performer can initiate a workflow, specifying conditions of satisfaction that he is willing to satisfy for a customer.

**Organization Roles**
Named positions in an organization who are authorized to make certain requests, agreements, take certain actions, set certain policies, and make certain decisions. The kind of roles will be accountant, office manager, etc.

**Performer**
One of the principal roles in a workflow: the role that commits to complete the conditions of satisfaction.

**Phase**
A characterization of the status of a workflow based on the acts that have happened and the acts that are permitted. Each workflow has four phases namely, the proposal phase, the agreement phase, the performance phase, and the satisfaction phase.

**Request**
A customer does this act to initiate a workflow and declare conditions of satisfaction.

**Trigger**
An action in a workflow which causes an action in some other workflow.

**Triggered**
Action in a workflow based on certain conditions/status in some other workflow.

**Workflow**
A structured set of acts between customers and performers organized to satisfy a customers conditions of satisfaction.

**Workflow Activation**
A triggered action that enables the customer or performer of the workflow to take the initial act of the workflow.

**Workflow Initiation**
An act of request or offer initiates a workflow.

**Workflow Roles**
The association of participants in the workflows that take the acts in workflows; three roles are distinguished in workflows: customer, performer, and observer.

**Workflow Type**
This indicates whether the workflow is of request, offer or note type.

**BRIEF DESCRIPTION OF THE DRAWINGS**

FIG. 1 is pictorial representation of a business process, i.e., a set of linked workflows.

FIG. 2 shows the phases of a workflow.

FIG. 3 is a block overview diagram of a complete workflow system including STF processors.

FIG. 4 is a block overview diagram showing the major components of an STF processor.

FIG. 5 shows the exchange of STF transactions between a workflow enabled application, the STF processor and server.

FIG. 6 is a generic class hierarchy diagram of a STF processor showing how the classes in STF processors are linked.

FIG. 7 shows the interaction of workflow enabled application, STF processor and workflow server for different kinds of transactions.
Overview

The present invention is directed to a Standard Transaction Format (STF) specification to address the requirements of applications, platform and medium independent representation and transfer of data related to business processes of a workflow system. The present invention is also directed to STF processors which are the modules that provide the server-side connection point for client/server interactions using any of several STF specification variants. These STF specification variants are messaging, updates to shared databases and inter-process communications. By accessing STF processors workflow enabled applications, using a standard transaction format, are able to request services from the workflow server component of the workflow system, remotely and indirectly using messaging, shared databases or inter-process communications. In other words, the STF specification defines the semantics for accessing the workflow services.

For each one of these types of interfaces there is a syntactic definition that gives the specific format for the representation of the workflow data and the process specific data in that medium. This syntax definition constitutes an STF specification that a particular workflow enabled application will then use.

The communication and interface between workflow enabled applications and the server is provided by STF processors. These STF processors map and translate between a workflow-enabled application’s data format and the data elements of the workflow system APIs.

STF processors provide a layer for integration of many different protocols and technologies. STF processors can be constructed for any message transport database technology, and inter-process communication protocol.

The interface from STF processors to the server is accomplished through the workflow system APIs. From the point of view of workflow services, the STF processors appear to the server as additional applications.

Since an STF processor is an application whose job is to interface external systems to the workflow system, there is one STF processor for each different type of system that interfaces to the workflow system. Business processes can be managed across platforms in diverse locations with diverse interconnections through the management of a class of transaction called STF transactions. Transactions include taking acts, getting reports of status, and notifications of acts taken by other workflow participants, among others.

Three classes of STF processors are described, which are characterized by the method of connection between client and server, which, as previously noted are: messaging, shared database, and inter-process communication. Within these interfaces, STF processor variants are required to conform to the various syntactic requirements of the communication technology. The particulars of such variants depend upon such syntactic requirements, however, the implementation details of such STF processor variants should be apparent to persons skilled in the art having knowledge of the syntactic requirements and the descriptions contained herein.

The client/server communications medium determines whether a client application can run synchronously with the server—making server requests and receiving server responses during the duration of a connection, or asynchronously—making server requests, disconnecting, and reconnecting later for responses. Inter-process communications media typically allow synchronous connections between client and server, message based media typically allow only asynchronous communication between client and server, and clients and servers which communicate through a shared database can operate either synchronously or asynchronously.

Thus, while STF processors may be provided for several STF specification variants, an STF processor with generic functional requirements is described. Further, this description focuses particular attention to the requirements of an STF processor for a messaging platform known as MHS (message handling system) available from Novell Corporation which will serve as an example case.

FIG. 4 shows the major components of an STF processor, namely, a transporter module, which includes a transaction parser and a transaction formatter, and a transaction processor module. The transaction processor module processes STF transactions received from workflow enabled applications through the transporter module and sent to workflow server via calls to the workflow server APIs. Similarly, it processes transactions queued by the workflow server and passes them to the transporter module to be sent to the WEA. The transactions the transaction processor processes are environment independent. The transporter module is adapted to the STF environment (i.e., messaging, shared database or inter-process communication) and receives incoming transactions from a messaging application, shared database or inter-process communication and sends outgoing transactions to a messaging application, shared database or inter-process communication.

The workflow transactions API provides an interface to the workflow server. The workflow server responds to transactions it finds in the transactions database and updates the workflow and places status information in the STF queue database to be processed by STF processors.

STF Transactions

An STF transaction is a workflow transaction defined in a specific format called Standard Transaction Format (STF). This standard enables any application to interface to the workflow server. An application is said to be a Workflow Enabled Application (WEA) when it is able to send/receive the workflow transactions in STF. STF transactions are passed from the STF processor to the workflow server via calls to the transactions API.

An STF transaction is composed of an envelope and workflow data. The envelope provides connection and addressing information translated by STF processors to formats appropriate for the particular medium supported by the STF processor. Workflow data includes workflow specific data and bound process data. STF transactions are exchanged between client and server as shown in FIG. 5. The figure shows some types of transactions as well as the role of the STF processor.

There are five STF transaction types as follows:

- Initiating a workflow
- Acting in a workflow
- Requesting the status of a workflow
- Requesting list of declared business processes
- Requesting list of workflows with pending actions

Components of STF Transactions

STF Envelope

The STF envelope is entirely platform and medium dependent. The envelope contains addressing information. In a messaging environment, the STF envelope would typically specify user and STF processor email addresses or equivalent. In an IPC environment (connection oriented), the
STF envelope would typically specify a logical unit identifier or equivalent. In a database environment, the STF envelope would typically specify user and STF processor identifications or equivalent.

It contains STF Processor ID for identification of a particular STF processor. Essentially this is the address with which a WEA sends an STF transaction.

For example, to address an STF processor STFPROC1 in work group ATL, a WEA in a messaging environment might specify the STF processor address as follows:

```
To: STFPROC1@ATI
```

**Workflow Data**

As previously noted, workflow data includes workflow specific data and bound process data.

**Workflow Specific Data**

All workflow transactions include a set of required workflow specific data elements that are defined by the STF specification.

The workflow specific data component of an STF transaction contains workflow attributes required to do each of the five types of transactions. Each type of STF transaction has a different set of workflow information. The STF transaction sent by a WEA must contain a Transaction ID (STFID) keyword. This aids the WEA and STF processor in identifying a transaction uniquely and helps the STF processor in sending status transactions to the WEA. Each STF transaction also has a set of mandatory parameters called minimal transaction parameters. Each STF transaction may also contain other optional parameters which are called extended set parameters.

An example of a minimal set of parameters for the workflow specific data component of an STF transaction is as follows:

- **STF type** (transaction type)
- **STF transaction identification**

An example of an extended set of parameters for the workflow specific data component of an STF transaction is as follows:

- **workflow participants or users**
- **workflow type**
- **transaction type**
- **expected and/or requested workflow completion dates**
- **workflow status**

**Bound Process Data**

Bound process data are data elements which are managed by the workflow server for purposes of management and as values in assignment, calculation, and flow control statements. Bound data elements are application-specific extensions to the STF specification. Examples include sales price, image data, and quantity ordered. These additional data elements can be used in processing and display definitions of client applications and the processing and control of business process definitions at the workflow server. For example, a time sheet submission workflow includes bound process data in the form of project names and hours worked. The business process definition might specify that the sum of hours worked on a particular day is used to calculate an employee’s paycheck amount which is used in a recurrent automated workflow to cut a pay check.

The STF interchanges shown in FIG. 5 are realized via client applications, STF processors, and workflow server interfaces. For example, a workflow enabled application may use MHS messages to send STF transactions to the workflow server. STF transactions are sent as attachments to MHS messages. The STF processor receives these messages, identifies the STF transaction, parses the transaction and passes information as required to the workflow server.

**Mapping Between STF Transactions And Workflow Server APIs**

The workflow server APIs provide the following functions:

- **Initiate a workflow**
- **Act in a workflow**
- **Bind process data**
- **Get field attributes**
- **Get Workflow Status**
- **Get Available acts**
- **Get Available Business Processes**

Get Workflows in progress and pending actions

The STF transaction set is designed to facilitate the invocation of workflow server API functions and to return status reports. Workflow server API functions to STF transaction mapping is shown in Table 2. Descriptions of the specified workflow server APIs may be found in co-pending U.S. application Ser. No. 08/014,796 filed Feb. 8, 1993.

<table>
<thead>
<tr>
<th>STF Transaction (Transaction Keyword)</th>
<th>AWST APIs called by STF Processor and their purposes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialising a Business Process (InitBP)</td>
<td>BeginTransaction (InitWF)</td>
</tr>
<tr>
<td>Initialise a Workflow (InitWF)</td>
<td>BeginTransaction (InitWF)</td>
</tr>
<tr>
<td>Take an Act in a WF (Act)</td>
<td>BeginTransaction (InitWF)</td>
</tr>
<tr>
<td>Bind Application data to a Workflow (Bind Data)</td>
<td>BeginTransaction (InitWF)</td>
</tr>
<tr>
<td>Get the status of a Workflow (GetWFStatus)</td>
<td>BeginTransaction (InitWF)</td>
</tr>
<tr>
<td>Get Available Business Processes</td>
<td>BeginTransaction (InitWF)</td>
</tr>
</tbody>
</table>

**TABLE 2**
TABLE 2-continued

<table>
<thead>
<tr>
<th>STF Transaction</th>
<th>AWS APIs called by STF Processor and their purposes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GetAvailableBPs)</td>
<td>AWSTNUMAVAILABLEBP ()</td>
</tr>
<tr>
<td></td>
<td>Get the Available BPs for the specified identity</td>
</tr>
<tr>
<td>(GetPendingActions)</td>
<td>AWSTAVAILABLEBP ()</td>
</tr>
<tr>
<td></td>
<td>Get the Pending WFs</td>
</tr>
<tr>
<td>Get the Workflows</td>
<td>Query for the number of Pending WFs</td>
</tr>
<tr>
<td>where action is</td>
<td>AWSTNUMQUERYWF ()</td>
</tr>
<tr>
<td>(GetPending)</td>
<td>Get the Pending WFs</td>
</tr>
<tr>
<td></td>
<td>AWSTQUERYWF ()</td>
</tr>
<tr>
<td>Notification generated</td>
<td>Poll STFQ for Notifications</td>
</tr>
<tr>
<td>by Workflow Processor</td>
<td>AWSTPOLLSTFQUEUE ()</td>
</tr>
<tr>
<td>(ReturnWFStatus)</td>
<td>Get the status and dates of the WF</td>
</tr>
<tr>
<td></td>
<td>AWSTSTATUS ()</td>
</tr>
<tr>
<td></td>
<td>Get the number of available acts</td>
</tr>
<tr>
<td></td>
<td>AWSTNUMAVAILABLEACTS ()</td>
</tr>
<tr>
<td></td>
<td>Get the available acts</td>
</tr>
<tr>
<td></td>
<td>AWSTAVAILABLEACTS ()</td>
</tr>
<tr>
<td></td>
<td>Get the number of App data structures</td>
</tr>
<tr>
<td></td>
<td>AWSTGETNUMAPPDATA ()</td>
</tr>
<tr>
<td></td>
<td>Get the App data bound of the WF</td>
</tr>
<tr>
<td></td>
<td>AWSTGETAPPDATA ()</td>
</tr>
<tr>
<td></td>
<td>Get the moment specific App Data of the WF</td>
</tr>
<tr>
<td></td>
<td>AWSTGETAPPDATAFIELDATTRIBUTES ()</td>
</tr>
</tbody>
</table>

In the case of a message type of interface, both the STF processor and the Workflow Enabled Application (WEA) read (write) messages from (into) predefined message queues (i.e., directories). Similarly, in a database type of interface they read/write records of a shared database. The STF processor is constantly servicing requests from a WEA. The interface of STF processor with the WEA and the server in both these cases is asynchronous. However, in the case of an IPC based interface, the interaction between WEA and the STF processor is synchronous.

Each type of STF processor is a separate executable (EXE). When installing the STF processor, its ID (name) is registered in the server via the workflow server manager module. To provide the necessary functionality, an STF processor deals with these different requirements through creation of a base Transporter class. Three subclasses are derived from the Transporter for: Msg, Database, and IPC. From Msg, further subclasses can be derived for the various messaging platforms supported, such as MHS, VIM or MAPI. From Database, a subclass such as for SQL can be derived. From IPC, subclasses can be derived for environments such as APPC. FIG. 3 shows the interrelation of these classes in the cases of incoming and outgoing transactions.

STF Transaction Definitions

InitBP
This transaction is used to initialize a business process and the primary workflow associated with the business process. To initialize a business process, InitBP needs the following parameters:
minimal transaction parameters:
STF Transaction ID
Business Process Name
Identity
extended set parameters:
Customer Name
Performer Name
Completion Date
Response Date
Initiate Date
Organization Roles to Identity mappings for the business process as well as the primary workflow.

BoundData
The STF processor makes the workflow server API call to initiate the business process and primary workflow. The workflow name of the primary workflow need not be provided by the WEA. Both the Business Process Transaction ID and the workflow name (of the primary workflow) are returned to the WEA by the STF processor. If there was an error, it is returned to the WEA.

InitWF
This transaction is used to initialize a workflow (other than the primary workflow). InitWF needs the following parameters:
minimal transaction parameters:
STF Transaction ID
Business Process Transaction ID
Workflow name
Identity
extended set parameters:
Customer Name
Performer Name
Completion Date
Reply Date
Organization Roles to Identity mapping for the workflow
BoundData
Act
This transaction specifies an Act to take in a workflow in a business process. In the case of the Act Transaction, the following parameters need to be passed:
minimal transaction parameters:
STF Transaction ID
Business Process Transaction ID
Workflow Name
Act to take
Identity
extended set parameters:
Completion Date
Reply Date
BoundData
If the Completion and Reply dates are not specified then the default values for that workflow are assumed by the server. If there is any process data that the WEA needs to bind to the business process or workflow instance then the name, type and value of the bound data can also be passed along with the Act transaction. The Act Transaction returns whether the Act transaction has been logged successfully in the Transaction database or not.

Bind Data
This transaction is to bind data to a workflow or a business process. In the case of the Bind Data Transaction the following parameters need to be passed:
minimal transaction parameters:
STF Transaction ID
Business Process Transaction ID
Identity
Data to be bound to the business process or workflow instance.
extended set parameters:
Workflow Name
The Bind Data Transaction returns status as to whether the application data has been bound to the business process or workflow instance successfully or not.
Get Workflow Status
This transaction is used to retrieve information related to a workflow instance. This information includes:
- WEA data bound to the workflow instance.
- List of bound data field names and attributes, when requested.
- The status of the workflow instance.
- The acts available in the workflow for the role of the specified identity.
The Get Workflow Status Transaction needs the following parameters to be passed:
minimal transaction parameters:
- STF Transaction ID
- Business Process Transaction ID
- Workflow Name
- Identity
- Workflow Role of the Identity
extended set parameters:
- Act or State Flag (Act or State) for bound data
- Act or State value for bound data
- Return Bound Data Flag (YES or NO)
- If the extended parameter, Bound Data (Boolean) is not specified then its default value is YES and bound data fields and attributes are returned to WEA.
The Get Workflow Status Transaction returns the following:
- Bound Data
- Status String
- Completion and Reply dates
- List of available acts for the Identity
- WF status for Act or State
Get Pending Actions
This transaction is used to retrieve information about workflows for an Identity having an Organization Role from the set of instantiated business processes with the specified business process name, which fall into the specified time slot between the start date and end date. This information includes, for each workflow, BP Name and ID, Workflow Name and ID, Customer ID, Performer ID, Completion and Reply Dates, Status and form name. Essentially, it is the list of workflows where an action is pending.
The Get Pending Actions Transaction needs the following parameters to be passed:
minimal transaction parameters:
- STF Transaction ID
- Identity
- Workflow Role of the Identity
extended set parameters:
- Organization Role
- Business Process Name
- Start Date
- End Date
If Organization Role is absent, then information about workflows for the Identity in all valid Organization Roles from the set of instantiated business processes with the specified business process name is returned. If Business Process name is absent, then information about workflows for the Identity in all valid Organization Roles from the complete set of instantiated business processes is returned. If the Start Date and End Date are absent then the time slot for the required information will span the entire database.
This transaction returns a list of workflows which includes:
- Business Process Name
- Business Process ID
- Workflow Name
- Customer Name
- Performer Name
Completion Date
Reply Dates
Status string
Get Available Business Processes
This transaction is used to get the list of business processes that the specified Identity with a specific Organization role can initiate. The Get Available Business Processes Transaction needs the following parameters to be passed:
minimal transaction parameters:
- STF Transaction ID
- Identity
extended set parameters:
- Business Process Status (Active or Inactive)
The Get Available Business Processes Transaction returns a list of business processes available for the Identity.
ReturnWFStatus
This transaction is generated by the workflow processor whenever workflow participants need to be informed about the workflow status. The STF processor polls the workflow processor continuously for any of the Notification events. If it finds one, it calls the Server APIs to get the workflow status (exactly similar to the GetWFStatus Transaction) and send all the workflow information to the participant.
This transaction is generated by the workflow processor and hence requires no parameters from the WEA.
The ReturnWFStatus Transaction returns the following:
- Notification string
- Notification Event
- Status String
- Completion and Reply dates
- List of available acts for the participant
- Bound Data
STF Transaction Representation in a Messaging Environment
The following describes the STF transaction representation in a messaging environment. The STF transaction travels as an attachment to the message. In a messaging environment, an STF transaction has the following format.
SIGNATURE
WORKFLOW DATA
The signature and addressing information, together constitute the STF Transaction Envelope.
Signature
Each STF Transaction starts with the Signature. It could be a line containing the following:
STFMHS-01
Workflow Data
This includes the STF type, STF Instance, workflow participants, workflow type, transaction type, start date, expected and/or requested dates and completion dates, workflow status etc.
STF Keyword Format
The STF keyword format in a messaging environment consists of four fields in the following format.
<Keyword (field name)> <delimiter> <Field values> <terminator>
Keyword (field name)
See Table 3 below.
Delimiter
The delimiter is a character such as a “:” used to separate the field name and the field value.
Field value

The syntax and semantics of the field value varies and will depend on the field. The following fields have a specific format in which data is to be sent.

Sent by WEA to STF Processor
OrgRole to Identity
BoundData
Sent by STF Processor to WEA
Acts
Status
BoundData
Workflow List
Terminator

The terminator consists of the carriage return—line feed pair <CR><LF>.

A sample STF transaction in a messaging environment would be as follows:

<table>
<thead>
<tr>
<th>SIGNATURE</th>
<th>STFMHS-01</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKFLOW DATA</td>
<td>STF/TYPE:GetWFStatus</td>
</tr>
<tr>
<td></td>
<td>STFID:10002</td>
</tr>
<tr>
<td></td>
<td>WFNAME:PrintPaySlips</td>
</tr>
<tr>
<td></td>
<td>BPID:10202</td>
</tr>
<tr>
<td></td>
<td>IDENTITY:Man-10</td>
</tr>
<tr>
<td></td>
<td>ACTORSTATE:Act</td>
</tr>
<tr>
<td></td>
<td>ACTSTATE:Request</td>
</tr>
</tbody>
</table>

STF Transaction Keyword Data Formats

The following describes the STF transaction keyword data formats which are medium independent.

<table>
<thead>
<tr>
<th>STF Keyword</th>
<th>Description, Data Type, Format and Valid Values</th>
</tr>
</thead>
</table>
| ACT | Description: Act to be taken
Data type and format: string of up to 64 characters
Valid values:
Request, Offer, Accept_CounterOffer,
Accept_Offer, Agree, Cancel, Counter,
CounterOffer, Counter_with_request,
Decline_Completion,
Decline_Dissatisfaction,
Decline_Satisfaction, Decline_Request,
Decline_Offer, Decline_CounterOffer, Revoke |
| ACTS | Description: List of Acts with their canonical and English names
Data type and format: Canonical name of Act, English Name of the act, ...
Where
Canonical Name of the Act - String of up to 64 characters
English Name of the Act - String of up to 64 characters |
| ACTORSTATE | Description: Act or State flag for which workflow status has been requested
Data type and format: String
Valid values:
Act, State. Default is Act |
| ACTSTATE | Description: Act or State value
Data type and format: String
Valid values for Act:
Request_Offer, Accept_CounterOffer,
Accept_Offer, Agree, Cancel, Counter,
CounterOffer, Counter_with_request,
Decline_Completion,
Decline_Dissatisfaction,
Decline_Satisfaction, Decline_Request,
Decline_Offer, Decline_CounterOffer, or Revoke
Valid values for State:
Inactive, Initial, Request, Offer,
Recounter, OCcounter, Agreement,
Completion, Satisfaction, Cancel,
Decline, or Revoke |
| BOUNDDATA | Description: Application data associated with a workflow
Data type and format: Bounddata format to be sent by WEA to STF Processor:
BDfieldName, BDfieldValue; BDfieldName, BDfieldValue
where,
BDfieldName - string of up to 64 characters
BDfieldValue - string of up to 255 characters. |
Each bounddata field is separated by comma and items are separated by semicolons. The last item need not contain a semicolon. All the fields in BoundData are mandatory and no blank values are allowed.

Data type and format: BoundData format to be sent by STF Processor to WEA
BDfieldName, BDfieldValue, AttributeFlag;
where,
BDfieldName - string of up to 64 characters
BDfieldValue - string of up to 255 characters.
AttributeFlag - string of 2 digits specifying Attribute
Attribute Flag Values
0 - Read Only
1 - Hidden
2 - MustFill
3 - Editable
4 - Reserved
5 - Reserved

BPTID
Description: Business Process Transaction ID returned by workflow server and is subsequently used by WEA for queries.
Data type and format: String of up to 64 characters

BPLIST
Description: Business Process list sent by Workflow Processor
Data type and format: list of Business process names and IDs
BPNAME; BPTID, BPNAME; BPTID, . . .
where
BPNAME = Business Process Name of String up to 64 characters
BPTID = Business Process Transaction ID of String up to 64 characters

BPNAME
Description: Business Process name
Data type and format: String of up to 64 characters

BPSTATUS
Description: Business Process status required (active or inactive)
Data type and format: String
Valid values:
Active, or Inactive

CDTIME
Description: Customer request completion date and time
Data type and format: Date formats
1. mm/dd/yyyy HH:MM:SS
2. MMM dd yyyy HH:MM:SS
3. MMMMMMMM dd yyyy HH:MM:SS
where
mm - Month (01–12)
MMM - Month (Jan–Dec)
MMMM - Month Name (January – December)
d - Day (0–31)
yyyy - Year (1970–200x)
HH - Hour (0–23)
MM - Minute (0–59)
SS - Second (0–59) This is optional
All the fields in the date and time are mandatory except the seconds field. The delimiters separating date fields can be blank, hyphen, forward slash, or dot (.)
Examples of Date and Time for November the 20th 1992 at 10 am can be specified as:
November-20-1992 10:00,
Nov-20-1992 10:00:00, or
11-20-1993 10:00

COMMENT
Description: Comment associated with an STF Transaction
Data type and format: String of up to 255 characters

CUSTOMERID
Description: Workflow customer Name
Data type and format: Customer name.
ENDTIME  String of up to 64 characters
Description: End of date and time range for Get Pending Action
Data type and format: Date format (as in CDTIME)

ERROR  Description: Error code returned by Workflow Processor to an STF Transaction
Data type and format: String of digits up to 8 characters

ERRORMSG  Description: Error message corresponding to ERROR
Data type and format: String of up to 1024 characters

FORMID  Description: ID for form in WF definition, returned as part of Bound Data
Data type and format: String of up to 64 characters

IDENTITY  Description: Identity who is participant in the Workflow
Data type and format: String of up to 64 characters

IDTIME  Description: Initiate time of a Business Process or Workflow
Data type and format: Date format (as in CDTIME)

NOTIFICATION  Description: Notification string returned by Workflow Processor to WEA
Data type and format: String of up to 255 characters specifying the Notification string.
Possible strings are Performer response past due, Performer completion past due, Performer completion coming due, Customer Response past due, Act taken.

NOTIFICATIONTYPE  Description: Notification Type returned by the Workflow Processor to the WEA
Data type and format: String of 1 digit Notification event types
0 - Follow Up
1 - Follow Up
2 - Reminder
3 - Follow Up
4 - Act

OBSERVERID  Description: Name of observer in workflow
Data type and format: String of up to 64 characters

ORG2IDENTITY  Description: Organizational Role to Identity Mapping for the workflow
Data type and format: Organization roles and identities list
ORGROLE~IDENTITY;ORGROLE~IDENTITY;...
where
ORGROLE - String of up to 64 characters
IDENTITY - String of up to 64 characters
All the fields in ORG2IDENTITY are mandatory and no blank values are allowed.

ORGRLE  Description: Organizational Role of the Identity
Data type and format: String of up to 64 characters

PENDINGACTIONS  Description: List of Workflows where some act is pending
Data type and format: List of Workflows where actions are pending
BPNAME, BPTid, WFNAME, CUSTOMER, PERFORMER, COMPLETIONDATE, REPLYDATE, STATUS;...
where
BPNAME - Business Process Name of String up to 64 characters
BPTid - Business Process Transaction ID of String up to 64 characters
WFNAME - Workflow Name of String up to 64 characters
CUSTOMER - Customer Name of String up to 64 characters
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Data type and format</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFORMERID</td>
<td>Description: Name of Performer in the Workflow</td>
<td>String of up to 64 characters</td>
<td></td>
</tr>
<tr>
<td>RDTIME</td>
<td>Description: Date and Time by which a participant asks for a response to his act.</td>
<td>Date format (as in CDTIME)</td>
<td></td>
</tr>
<tr>
<td>RTINBOUNDDATA</td>
<td>Description: Flag indicating whether to return application data to WEA</td>
<td>String</td>
<td>Yes, No</td>
</tr>
<tr>
<td>RTI STATUS</td>
<td>Description: Flag indicating whether to return status to WEA</td>
<td>String</td>
<td>Yes, No</td>
</tr>
<tr>
<td>STARTDTIME</td>
<td>Description: Start date for Pending Actions</td>
<td>Date format (as in CDTIME)</td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td>Description: Status of Workflow</td>
<td>Workflow status string format</td>
<td></td>
</tr>
<tr>
<td>STFID</td>
<td>Description: STF Transaction ID specified by WEA</td>
<td>String of up to 255 characters</td>
<td></td>
</tr>
<tr>
<td>STFTYPE</td>
<td>Description: STF Transaction Type</td>
<td>String of up to 64 characters</td>
<td></td>
</tr>
<tr>
<td>STFVER</td>
<td>Description: STF Transaction Version</td>
<td>String of up to 64 characters</td>
<td></td>
</tr>
<tr>
<td>WFNAME</td>
<td>Description: Workflow Name</td>
<td>String of up to 64 characters</td>
<td></td>
</tr>
</tbody>
</table>
The STF transaction is placed in an attachment file. This file needs to be specified in the attachment IPM Header. An STF processor makes use of the following SMF headers from the MHS Message to retrieve addressing information for the STF Transaction. Other header information is ignored by the STF Processor.

From To Attachment

The following are the complete STF transactions as received by a STF processor from a WEA via MHS. It also illustrates the STF Transaction sent by STF Processor in reply to the WEA via MHS.

Example of MHS message header file received (from WEA) by STF Processor through MHS

The corresponding workflow server API Calls would be:

- continued

- continued

Suppose that an error occurred in the InitBP Transaction that BPNAME keyword was not found, then the values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES). By default, STF Processor assumes RTNSTATUS as YES). ERROR keyword is set to 0 if there was no error processing the Transaction. Otherwise ERROR will be set to a value and the error message corresponding to the ERROR will be sent in ERRORMSG.

- continued
The corresponding workflow server API calls would be:

- AWSBeginTransaction() 
- AWSTInitWF(); 
- AWSTBindAppData(); 
- AWSTEndTransaction() 

The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES).

The corresponding workflow server API call for the Bind Data Transaction would be:

- AWSBeginTransaction() 
- AWSBindAppData(); 
- AWSEndTransaction() 

The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES).

The corresponding workflow server API calls would be:

- AWSBeginTransaction() 
- AWSAvailableActs(); 
- AWSGetAppData(); 

The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES).

Since this transaction is used to retrieve all information related to a workflow instance, it maps to several workflow server APIs.

- STFTYPE:ReturnStatus 
- STFFID:10404 
- ERROR:00000000 
- ERRORMSG:ReturnStatus to Get Workflow Status Transaction
27

STFTYPE: GetAvailableBPs
STFID: 10404
STATUS: Completed, 02-01-1993 17:00, 02-01-1993 17:00, 02-01-1993 17:00; 02-01-1993 17:00
ACTS: Agree, PrintSlips
FORMID: SAMPLEFORM
BOUNDDATA: GR8-BASIC, 3000.00, 01; GR7-BASIC, 3500.00, 01; GR6-BASIC, 4000.00, 01
ERROR: 00000000
ERRORMSG: Transaction with workflow server is successful

Note: STATUS and ACTS keyword format
STATUS: WFStatus, Completion Date1, Reply Date1, Completion Date2, Reply Date2
where,
WFStatus string Workflow status string
(Inactive, Initial, Request, Offer, Recounter, Counter, Agreement, Completion, Satisfaction, Cancel, Decline, or Revoke)
Completion Date1 Completion requested by Customer (Completion due for Performer)
Reply Date1 Reply due to Performer from Customer (Reply due to customer from Performer)
Completion Date2 Completion due by Performer (Completion requested by Customer from Performer)
Reply Date2 Reply due by Performer from Customer (Reply due by Customer from Performer)

ACTS: Act Type, Act Name
where,
Act Type string Act Type string (Request, Offer, Accept__CounterOffer, Accept__Offer, Agree, Cancel, Counter, CounterOffer, Counter__with_request, Declare__Completion, Declare__Rejection, Decline__Rejection, Decline__Request, Decline__Offer, Decline__CounterOffer, Decline__CounterOffer, Revoke, or Null)
Act Name Act Name string.

Get Available Business Processes
Attachment file containing Get Available Business Processes Transaction
STFTYPE: STFMHS-01
STFID: 10405
IDENTITY: Mani-ID
BPSTATUS: Active
The Get Available Business Processes returns the list of Business Processes that the identity (in the specific role) can initiate. The BPSTATUS is an optional parameter which specifies whether active or inactive (all) BPs are required by the WEA.
The corresponding workflow server API Call would be:
AWSTAvailableBPQ;
The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES)

Attachment file containing ReturnStatus to Get Available BPs Transaction
STFTYPE: ReturnStatus
STFID: 10405
PENDINGACTIONS: WPNAME1; WPNAME2
ERROR: 00000000
ERRORMSG: Transaction with workflow server is successful

Return WFStatus
Attachment file containing ReturnWFStatus Transaction
None
The Get Pending Actions returns information about the workflows for the specified identity (having the specific Organization role) from the set of instantiated business processes with the specified Business Process name.
The corresponding workflow server API Call would be:
AWSTQueryWF();
The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES)

Attachment file containing ReturnStatus to Get Pending Actions Transaction
STFTYPE: ReturnStatus
STFID: 10405
PENDINGACTIONS: WPNAME1; WPNAME2
ERROR: 00000000
ERRORMSG: Transaction with workflow server is successful

The Get Pending Actions returns information about the workflows for the specified identity (having the specific Organization role) from the set of instantiated business processes with the specified Business Process name.
The corresponding workflow server API Call would be:
AWSTQueryWF();
The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES)

Attachment file containing ReturnStatus to Get Available BPs Transaction
STFTYPE: ReturnStatus
STFID: 10405
PENDINGACTIONS: WPNAME1; WPNAME2
ERROR: 00000000
ERRORMSG: Transaction with workflow server is successful

The Get Pending Actions returns information about the workflows for the specified identity (having the specific Organization role) from the set of instantiated business processes with the specified Business Process name.
The corresponding workflow server API Call would be:
AWSTQueryWF();
The values returned by the workflow server APIs will be sent to the WEA as Return Status transaction (if WEA had set RTNSTATUS to YES. By default, STF Processor assumes RTNSTATUS as YES)

Attachment file containing ReturnStatus to Get Available BPs Transaction
STFTYPE: ReturnStatus
STFID: 10405
PENDINGACTIONS: WPNAME1; WPNAME2
ERROR: 00000000
ERRORMSG: Transaction with workflow server is successful

This Transaction is generated by the Workflow Processor whenever the Workflow participants are needed to be informed about the Workflow status. The STF Processor will poll the Workflow Processor continuously for any of the Notification events. If it finds one, it calls the Server APIs to get the Workflow Status (exactly similar to the GetWFStatus Transaction) and send all the workflow information to the participant.
The corresponding workflow server API Calls would be:
AWSTBeginTransaction()
AWSPollSTFQueue()
AWSTStatus();
AWSTAvailableActs();
AWSTGetAppData();
AWSTEndTransaction()
The values returned by the workflow server APIs will be sent to the WEA as Notification transaction.

STF Transaction Representation in a Database Environment

In the case of a shared database environment, the STF transaction format is a set of tables. Each table has a record structure as follows:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act</td>
<td>Data required Act</td>
</tr>
<tr>
<td>Acts</td>
<td>List of canonical acts and their English names</td>
</tr>
<tr>
<td>BindData</td>
<td>Binding Application Specific Data to Workflow</td>
</tr>
<tr>
<td>BoundData</td>
<td>Application specific Bound Data</td>
</tr>
<tr>
<td>BusinessProcessList</td>
<td>List of Available Business Processes</td>
</tr>
<tr>
<td>Error</td>
<td>Error information returned by STF Processor</td>
</tr>
<tr>
<td>GetAvailableBPsi</td>
<td>Get Available Transactions</td>
</tr>
<tr>
<td>GetPendingActsi</td>
<td>Get Pending Actions Transaction</td>
</tr>
<tr>
<td>GetWFStatus</td>
<td>Get Workflow Status Transaction</td>
</tr>
<tr>
<td>InitBP</td>
<td>Initialize Business Process Transaction</td>
</tr>
<tr>
<td>InitWF</td>
<td>Initialize Workflow Status Transaction</td>
</tr>
<tr>
<td>Org2Identity</td>
<td>Organizational Role to Identity Mapping</td>
</tr>
<tr>
<td>PendingActions</td>
<td>List of Workflows where some act is pending</td>
</tr>
<tr>
<td>ReturnWFStatus</td>
<td>Return Workflow Status</td>
</tr>
<tr>
<td>WorkflowStatus</td>
<td>Workflow Status data returned by GetWFStatus and ReturnWFStatus</td>
</tr>
</tbody>
</table>

Each table consists of a set of columns in a relational table. The first field in the table is the Transaction ID which is the primary key for the table—it's value can never be void. Other fields may contain void values depending upon whether those fields are of the minimal set or the extended set of parameters of the respective STF transaction.

An alternative implementation of the STF transaction representation in a database environment would consist of three tables as follows:

<table>
<thead>
<tr>
<th>Keyword Index Table</th>
<th>STF Keyword</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ACTS</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

where the STF Trans ID column contains STF transaction identification, the keyword column contains a keyword index corresponding to the keyword index table; and the value column contains one of the valid values from the STF Transaction Keyword Data Formats table.

STF Transaction Representation in an Inter-Process Communication Environment

The STF transaction in an IPC environment is represented by a set of parameters in a remote procedure call, in a manner similar to the parameters of a workflow API call as described in U.S. application Ser. No. 08/014,796 filed Feb. 8, 1993. The following is a description of the three types of STF processors needed for messaging, shared database and inter-process communication. Details for the messaging type are explained by way of an example using MHS and VIM. Details needed for the shared database type by way of examples using SQL. Details for the inter-process communication type should be apparent to persons skilled in the art from this description.

In the preferred embodiment as set forth below, the invented system is implemented using the Model, View, Class (MVC) paradigm of object oriented programming.

Transporter Module

The transporter module of an STF processor handles the reception of inputs arriving via messaging, shared database or IPC and the preparation of outputs via messaging, shared database or IPC. The transporter recognizes, reads and writes entire transactions. The transporter accesses all items of bound data or transaction data via keyword entries. Client input/output content is required to be formatted as keyword, value(s), flags for each item of a transaction or bound data.

The transporter class provides basic interaction with the WEA software. In the case of MHS, for example, the transporter class provides the messaging interface. It gets and puts messages and passes them to the TxParser, which is a class that understands and reads the MHS message and extracts from it the STF transaction. The transporter also receives the message from the TxFormatter which is a class that constructs the message in MHS format. When a developer creates a new STF processor, it is necessary to derive from the subclasses of the transporter class an interface to the required transport medium to the STF processor.

The TxProcessor and STF.Transaction (STF_Tx) classes form the core of the STF processor. The TxProcessor is the controlling class that keeps track of transactions queued up to be done. The STF_Tx creates all the objects which between them contain all the methods necessary to interface to the workflow server.

Since the STF processor converts a WEA-formatted transaction into a workflow server-formatted transaction, the transaction classes center the conversion process and provide methods to both sides of the conversion.
Transaction Processor

The TxProcessor is the central controller class in STF
Processor. It has pointers to Error, INIFile, Transporter,
TxParser, TxFormatter and STF__Tx base classes in the STF
Processor. FIG. 6 illustrates the generic class hierarchy
diagram of a STF processor and how the classes in STF
processors are linked.

TxProcessor has methods to process input and output
transactions. The TxProcessor constructor instantiates
TransMsgMHS (for a messaging environment based on
MHS), TxParser and TxFormatter.

Within the TxProcessor method, to process input
transactions, a parser method to get input from Transporter
is called which loads the input transaction.

Next, STF__Tx is instantiated in inbound or outbound
mode depending upon TXProcessor method for processing
input or output Transactions.

Pointers to Transporter, TxParser and TxFormatter are
passed on to STF__Tx private data members through the
constructor. The STF__Tx constructor then gets the
Transaction Type by calling parser method for processing inbound
Transactions or calls a method for polling STFQ for for­
processing outbound Transactions.

Depending upon the Transaction Type appropriate trans­
action objects are instantiated which in turn process the
Bound data, Envelope data and call appropriate workflow
server APIs.

The return status of the workflow server API's is handled
by creating a return Transaction which is internally passed
over to TxFormatter for onward delivery to Transporter.
Cleanup operation is done by calling TxParser's methods
to do the same. This completes one inbound or outbound
transaction processing.

All the output transactions are processed. If there are no
output transactions pending, a message is posted to process
input transactions.

STF Processor Class Descriptions

Class Name: INIFile

Class Description

This is the INI file base class that handles STF Processor
INI file loading. The INI file is created during STF Processor
installation. STF Processor will assume a default STF:INI in
the current directory. If the STF:INI is not found in the
current directory, it will be searched in PATH environment
variable. The INI file is similar to OS/2 INI files. The INI file
will have keywords and values. Following is the content of the
INI file in STF Processor Ver1.0.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szINIFile</td>
<td>INI file name</td>
</tr>
<tr>
<td>szSTFProcID</td>
<td>STF Processor ID</td>
</tr>
</tbody>
</table>
virtual BOOL bWriteWEOOutput(PPSZ ppszFmtOutput, USHORT uFOCount)

virtual BOOL bGetInUserInfo(PPSZ ppszInUserInfo, USHORT* ppInUserinfo Count )

virtual BOOL bPutOutUserInfo(PPSZ ppszOutUserInfo, USHORT* ppOutUserinfo, USHORT* puiUserinfo Count)

virtual BOOL bDeleteTx(USHORT* uDeleteTxFlag)

Attribute Name | Attribute Description
--- | ---
ppszInUserInfo | input user info used for storing addressing information of inbound transactions.
uiUserInfoCount | number of elements in ppszInUserInfo.
ppszOut UserInfo | Output user info used for addressing outbound transactions.
uOutUserInfoCount | number of elements in ppszOutUserInfo.

Class Name: STFTransMsg

Class Description
STFTransMsg derived class consists of a set of public methods which overload the appropriate methods of the STFTransporter class to incorporate message specific features. The STFTransMsg derived class also contains a set of protected virtual methods which are used in defining the set of the STFTransMsg public methods, i.e. in writing the code of these public methods. This set of protected virtual methods in turn is overloaded by the methods of the derived class STFTransMsgMHS in case of messaging environment based on MHS and by methods of STFTransMsgVIM derived class in the case of a messaging environment based on VIM.

Super Class
STFTransporter

STFTransMsgMHS handles MHS messaging environment
STFTransMsgVIM handles VIM messaging environment

Classes Used
None

Attributes

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
</table>
| STFTransMsg( ) | Constructor used for initializing private data members of STFTransMsg class. This method uses protected virtual methods which are overloaded by appropriate methods of STFTransMsgMHS and STFTransMsgVIM derived classes. It gets the inbound transaction, user information and number of strings present in Input transaction and user information data structures respectively. The user information data is stored in its private data members. Various arguments are described below:
1. ppszWEASTFInputTx argument receives reference to array of pointers to strings containing transactions sent by WEA.
2. uITCount specifies count of strings in ppszWEASTFInputTx.
3. pbMore flag indicates if more unread input messages are present. This flag is useful in processing multiple input transactions. This method accepts parameters for creating an outbound transaction to be sent to WEA. It accepts array of pointers to strings from which a return Transaction is formulated (in case of messaging environment it creates an attachment file) and posts the message along with formulated transaction (attachment file in Messaging environment) as WEA output. It makes use of protected virtual methods that are overloaded by the appropriate methods of STFTransMsgMHS and STFTransMsgVIM derived classes. Various arguments are given below:
1. ppszFmtOutput argument contains formatter output Transaction for onward delivery to WEA.
2. uFOCount argument specifies number of strings in ppszFmtOutput. This method gets User Information that is stored in ppszUserInfo private data member. This is used for addressing the responses to the current transaction. The various arguments passed are given as under:
1. ppszUserInfo argument contains Input user information for addressing purpose.
2. pbUCount points to number of strings in ppszUserInfo. |
-continued

virtual BOOL bOpenMsg( )
This method opens and reads the incoming unread message file. It returns Boolean indicating presence or absence of WEA input. It as well sets the more flag in case more unread messages are present. This method is overloaded by bOpenMsg( ) method of STFTransMsgMHS or STFTransMsgVIM derived classes. It opens and reads the incoming unread message file for attachment file name and user information.

virtual BOOL bMarkMsgRead(USHORT *uMarkMsgRead) This method is overloaded by bMarkMsgRead(USHORT *uMarkMsgRead) method of this class.

virtual BOOL bDeleteTx(USHORT *uDeleteTxFlag)
This method is overloaded by bDeleteTx(USHORT *uDeleteTxFlag) method of STFTransMsgMHS or STFTransMsgVIM derived classes. This method accepts user information to be used within transporter for addressing outbound transaction. The various arguments are described below:
1. pszOutUserInfo argument contains array of pointers to strings containing user information for addressing outbound transaction. 2. uOutUICount specifies number of strings in pszOutUserInfo data structure.

virtual BOOL bPutOutUserinfo(PPSZ *ppszOutUserinfo, USHORT *uOutUICount)
This method is overloaded by bPutOutUserinfo(PPSZ *ppszOutUserinfo, USHORT *uOutUICount) method of this class. This method is used to post the outbound transaction. The STFTransMsgVIM derived classes pass the referenece to array of pointers containing item names, values and flags of the transaction. It will be used by bPutOutUserinfo() public method of this class.

virtual BOOL bQueryNewMsg(BOOL *pbMore)
This method accepts user information to be used within transporter for addressing outbound transaction. The various arguments are described below:
1. pszOutUserInfo argument contains array of pointers to strings containing user information for addressing outbound transaction. 2. uOutUICount specifies number of strings in pszOutUserInfo data structure.

virtual BOOL bSendMsg( )
This method is overloaded by bSetMsgHdr( ) methods of STFTransMsgMHS or STFTransMsgVIM derived classes. It creates a message header file from array of pointers to strings passed as argument to it.

virtual BOOL bCreateMsg(PPSZ *pszFmtOut, USHORT *uFmtOutCount)
This method accepts user information to be used within transporter for addressing outbound transaction. The various arguments are described below:
1. pszOutUserInfo argument contains array of pointers to strings containing user information for addressing outbound transaction. 2. uFmtOutCount specifies number of strings in pszOutUserInfo data structure.
The procedure followed by this method is broadly categorized in following steps.

1. Allocate storage for ppszMsgHdr data member.
2. Create SMF message header as array of strings in ppszMsgHdr using pszInAttFilName member. The steps followed in this method are given below:
   1. Create a unique file from a global integer variable which is appended with STF keyword.
   2. Open the file using DOS Open command in non-shareable, write only mode.
   3. Writes strings from ppszMsgOut into file separating them with a new line character (\015\040).
   4. Assign the Attachment field of SMF object given by pszAttachName. Extracts the name of file and sets it in pszOutAtFName data member.
5. Assign From:field of SMF to application name given by pszStfAppName variable.
6. Assign From:field of SMF to application name given by pszShAddrName variable.
7. Sends the message created using bCreateMsg() method.

The following table provides the method names starting with b* along with the value for attribute Name and Description:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bCreateMsg()</td>
<td>This method creates a message file from array of pointers to strings compiled by bSetMsgHdr() in ppszMsgHdr data structure in output mail directory. This method formulates an array of pointers to strings in ppszInputTx and opens a VIM session by bOpenMsg() method. This method formulates an array of pointers to strings in ppszInputTx and assigns the unique file name created to pszOutAtFName data member. The steps followed in this method are given below.</td>
</tr>
<tr>
<td>bSetMsgHdr()</td>
<td>This method opens up the first unread message given by private data member pszUnreadMsgFileName, scans message file for attachment file name and From fields. The steps followed by this method are described below.</td>
</tr>
<tr>
<td>bQueryNewMsg(BOOL *bMore)</td>
<td>This method polls the (STF processor) mail directory to check for incoming messages. Returns Boolean indicating presence or absence of inputs from WEA and stores the name of first unread file in pszUnreadMsgFileName private data member. It as well updates a more flag if more unread messages are present.</td>
</tr>
<tr>
<td>bOpenMsg()</td>
<td>This method opens up the first unread message given by private data member pszUnreadMsgFileName, scans message file for attachment file name and From fields. The steps followed by this method are given below.</td>
</tr>
</tbody>
</table>

### Class Name: STIFTransMsgVIM

Class Description

The STIFTransMsg VIM derived class consists of a set of public methods which overload the methods of the STIFTransMsg derived class. This set of public methods are specific to VIM messaging environment.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vSession</td>
<td>VIM session identifier</td>
</tr>
<tr>
<td>vMsg</td>
<td>VIM open message identifier</td>
</tr>
<tr>
<td>vInMsg</td>
<td>VIM open Message identifier for inbound message</td>
</tr>
<tr>
<td>vRef</td>
<td>VIM message reference position</td>
</tr>
<tr>
<td>vContainer</td>
<td>VIM open message container identifier</td>
</tr>
</tbody>
</table>

### Public Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateMsg(PPSZ ppszFmtOut, USHORT uFmtOutCount)</td>
<td>This method creates an outbound message. It uses the IVCtMsg(sender) VIM API</td>
</tr>
<tr>
<td>CreateMsg(PPSZ ppszFmtOut, USHORT uFmtOutCount)</td>
<td>This method creates an outbound message. It uses the IVCtMsg(sender) VIM API</td>
</tr>
</tbody>
</table>

The following table provides the method names starting with b* along with the value for attribute Name and Description:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bCreateMsg()</td>
<td>This method creates a message file from array of pointers to strings compiled by bSetMsgHdr() in ppszMsgHdr data structure in output mail directory. This method formulates an array of pointers to strings in ppszInputTx and assigns the unique file name created to pszOutAtFName data member. The steps followed in this method are given below.</td>
</tr>
<tr>
<td>bSetMsgHdr()</td>
<td>This method opens up the first unread message given by private data member pszUnreadMsgFileName, scans message file for attachment file name and From fields. The steps followed by this method are given below.</td>
</tr>
</tbody>
</table>
virtual BOOL SetMsgHdr( )
virtual BOOL SendMsg( )
virtual BOOL QueryNewMsg(DWORD pMore)
virtual BOOL OpenMsg( )
virtual BOOL GetMsgItem(DWORD ppszinputTx, USHORT* *pMoreinputs)
virtual BOOL CloseMsg( )
virtual BOOL MarkMsgRead( )

Destructor of STFTransMsgVIM.

Class Name: STFTransDB
Class Description
STFTransDB derived class consists of a set of public methods which overload the appropriate methods of the STFTransporter class to incorporate Database specific features. The STFTransDB derived class also contains a set of protected virtual methods which are used in defining the set of the STFTransDB public methods, i.e. in writing the code of these public methods. This set of protected virtual methods in turn is overloaded by the methods of the derived class STFTransDBSQL in case of Database environment based on SQL.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppszDBServerInfo</td>
<td>Database Management System</td>
</tr>
<tr>
<td>uDBServerInfoCount</td>
<td>Information number of elements in ppszDBServerInfo</td>
</tr>
</tbody>
</table>

Public Methods

Method Name | Method Description
-------------|-----------------------------------------------------
STFTransDB( ) | Constructor used for initializing the database management system information. This method uses protected virtual methods which are overloaded by appropriate methods of STFTransDBSQL derived class. It gets the inbound transaction, database server information.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppszAWEASTFInputTx</td>
<td>WEA string pointer to string used for writing the database records onto WEA.</td>
</tr>
<tr>
<td>ulnTxCount</td>
<td>The ulnTxCount specifies count of strings in ppszAWEASTFInputTx.</td>
</tr>
<tr>
<td>USHORT* ulMoreInputs</td>
<td>More Inputs flag indicates if more inbound STF database records are present. This flag is useful in processing multiple input transactions.</td>
</tr>
</tbody>
</table>

This method accepts parameters for creating an outbound transaction to be sent to WEA. It accepts array of pointers to strings from which a return Transaction is formulated as a database record and writes it onto the STF shared database. It makes use of protected virtual methods that are overloaded by the appropriate methods of STFTransDBSQL derived class. Various arguments are given below:

1. ppszFmtOutput argument contains formatter output Transaction for onward delivery to WEA.
2. uFOCount argument specifies number of strings in ppszFmtOutput.
3. pMoreInputs flag indicates if more inbound STF database records are present. This flag is useful in processing multiple input transactions.

This method accepts parameters for creating an outbound transaction to be sent to WEA. It accepts array of pointers to strings from which a return Transaction is formulated as a database record and writes it onto the STF shared database. It makes use of protected virtual methods that are overloaded by the appropriate methods of STFTransDBSQL derived class. Various arguments are given below:

1. ppszFmtOutput argument contains formatter output Transaction for onward delivery to WEA.
2. uFOCount argument specifies number of strings in ppszFmtOutput.

This method gets ST database and dictionary information that is stored in ppszDBServerInfo private data member. This is used for writing the database records onto STF Database. The various arguments passed are given as under:

1. ppszDBServerInfo argument contains Input STF Database management system information for reading and writing records to the database.
2. pDBInfoCount points to number of strings in ppszDBServerInfo.

This method accepts database management system information to be used within the transporter for addressing outbound
virtual BOOL bDeleteTx(USHORT *uDeleteTxFlag) {
    This method is overloaded by bDeleteSTFRecord() method of STFTransDBSQL derived class. It
    deletes memory for some of its private data members.
}

virtual BOOL bDeleteSTFRecord(USHORT *uDeleteTxFlag) {
    This method is overloaded by bDeleteSTFRecord() method of STFTransDBSQL derived class. It
    deletes memory for some of its private data members.
}

virtual BOOL bQueryNewSTFRecord(BOOL *pMore) {
    This method formulates an array of pointers to strings from ppszFmtOut into the database tables.
    This method polishes the STF processor database tables for new transactions with Transaction ID as the key.
    Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.
}

virtual BOOL bCreateSTFRecord(PPSZ ppszFmtOut, USHORT *uFmtOutCount) {
    This method formulates an array of pointers to strings from ppszFmtOut into the database tables.
    This method polishes the STF processor database tables for new transactions with Transaction ID as the key.
    Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.
}

virtual BOOL bReadSTFRecord(USHORT *uDeleteTxFlag) {
    This method formulates an array of pointers to strings from ppszFmtOut into the database tables.
    This method polishes the STF processor database tables for new transactions with Transaction ID as the key.
    Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.
}

class TxParser:

- Class Name: STFTransDBSQL

    Class Description

    This class parses the WEA input which is stored as an array of strings stored in the attributes. It has methods for parsing and returning keyword values from input STF Transactions. Please refer to Appendix-B for a list of keywords and their explanations. The TxParser object is created by the TxProcessor for getting Input Transaction from STFTransporter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pppszSTFTxt</td>
<td>Pointer to array of strings containing input transactions read from transporter.</td>
</tr>
<tr>
<td>pTransporter</td>
<td>Pointer to STFTransporter base class</td>
</tr>
</tbody>
</table>

Class Name: STFTransDBSQL

Class Description

The STFTransDBSQL derived class from STFTransDB consists of a set of public methods which overload the methods of the STFTransDB derived class. This set of public methods are specific to SQL Database Server environment.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STFTransDBSQL( )</td>
<td>Constructor of STFTransDBSQL class. It initializes private data members of STFTransDBSQL. This creates record in the output STF DB tables and writes strings from ppszFmtOut into the database tables. This method polishes the STF processor database tables for new transactions with Transaction ID as the key. Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.</td>
</tr>
<tr>
<td>bCreateSTFRecord(PPSZ ppszFmtOut, USHORT *uFmtOutCount)</td>
<td>This method formulates an array of pointers to strings from ppszFmtOut into the database tables. This method polishes the STF processor database tables for new transactions with Transaction ID as the key. Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.</td>
</tr>
<tr>
<td>bQueryNewSTFRecord(BOOL *pMore)</td>
<td>This method formulates an array of pointers to strings from ppszFmtOut into the database tables. This method polishes the STF processor database tables for new transactions with Transaction ID as the key. Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.</td>
</tr>
<tr>
<td>bDeleteSTFRecord(USHORT *uDeleteTxFlag)</td>
<td>This method formulates an array of pointers to strings from ppszFmtOut into the database tables. This method polishes the STF processor database tables for new transactions with Transaction ID as the key. Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.</td>
</tr>
<tr>
<td>bDeleteTx(USHORT *uDeleteTxFlag)</td>
<td>This method formulates an array of pointers to strings from ppszFmtOut into the database tables. This method polishes the STF processor database tables for new transactions with Transaction ID as the key. Returns Boolean indicating presence or absence of inputs from WEA and stores the WEA input record. It updates a more flag if more unread records are present in the STF DB.</td>
</tr>
</tbody>
</table>
The following methods are used to format STF Transaction keyword and value are overloaded for different keyword types.

### Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxFormatter</td>
<td>This method is the constructor for this class which initializes the attributes.</td>
</tr>
<tr>
<td>bFmtStart()</td>
<td>This method signals the formatter that all the data that needs to be sent to WEA has been formatted and can call STFTransporter method to format the STFTransaction to the WEA.</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppszOutSTFtx</td>
<td>data structure containing output transaction.</td>
</tr>
</tbody>
</table>

The following methods are used to get STF Transaction keyword and value for the STFTransporter. The TxFormatter keeps on accumulating STF keyword strings till it gets bFmtEnd() message. The bFmtEnd() will call STFTransporter method to write the STF Transaction to the WEA.

### Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bGetSTFtxKwdValue</td>
<td>This method gets the keyword values for the STFTransporter. This class will hide the Transaction from knowing about the external interface to WEA. It will essentially format the data required by the STFTransporter to output an STF Transaction. This class is used by Transaction class. The TxFormatter will call</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uKwdValue</td>
<td>The index of the keyword in the array of strings.</td>
</tr>
</tbody>
</table>

### Class Name: TxFormatter

### Class Description

This class contains methods to produce STF Transaction from keyword values for the STFTransporter. This class will hide the Transaction from knowing about the external interface to WEA. It will essentially format the data required by the STFTransporter to output an STF Transaction. This class is used by Transaction class. The TxFormatter will call
### Class Name: Envelope

**Class Description**
This class handles addressing information of WEA. It is used by STF_Tx, BoundData and Transaction classes.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppszOutUserlnfo</td>
<td>pointer to array of strings containing inbound user information.</td>
</tr>
<tr>
<td>ulnUserlnfoCount</td>
<td>Number of items in ppszOutUserlnfo</td>
</tr>
<tr>
<td>ppszInUserlnfo</td>
<td>pointer to array of strings containing outbound user information.</td>
</tr>
<tr>
<td>ulnInUserlnfoCount</td>
<td>Number of items in ppszInUserlnfo</td>
</tr>
</tbody>
</table>

**Methods**

- `~TxFormatter()`: This method is the destructor for this class which de-initializes the attributes.
- `Envelope()`: This method is the constructor which initializes the attributes.
- `BOOL bFmtKwd (PSZ pszKwd, LPBPLIST pBPList, INT iCount)`: This method formats the pszKwd and pBPList contents in the form `pszKwd[pBPList[0]][pBPList[1]][...pBPList[iCount-1]]`.

**Methods (Continued)**

- `BOOL bPutOutUserlnfo(STF_Tx *pSTF_Tx)`: This method accepts string containing user information and calls STFTransporter for addressing and binding user information for current outbound transaction.

**Class Name: BoundData**

**Class Description**
This class stores and handles all bound data associated with a transaction.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSTFTx</td>
<td>Pointer to STF_Tx object</td>
</tr>
<tr>
<td>szBDKwd</td>
<td>BOUNDDATA keyword string</td>
</tr>
<tr>
<td>szFormName</td>
<td>Form Name</td>
</tr>
<tr>
<td>iBDCount</td>
<td>Number of BD structures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoundData(STF_Tx *pSTF_Tx)</td>
<td>This method is the constructor which updates pSTFTx with the passed parameter and szBDKwd by instantiating STFTxKwd</td>
</tr>
<tr>
<td>BOOL bPutInTxBoundData( )</td>
<td>This method updates the bound data attributes from the input transaction using TxParser method.</td>
</tr>
<tr>
<td>BOOL bPutGetBDFieldAttributes(BOOL bActOrState, INT iActOrState)</td>
<td>This method gets the ActOrState Flag and ActOrState value and updates the bound data attributes by calling AWSGetBoundData( ) and AWSGetBDFieldAttributes( ). This method is for 'Get' type of transactions.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bPutRtnTxBoundData(LONG iTxd)</td>
<td>This method gets the Transaction ID and calls the overloaded methods of AWSGetBoundData( ) and AWSGetBDFieldAttributes( ) and updates the bound data attributes. This method is for 'Return' type of transactions.</td>
</tr>
<tr>
<td>BOOL bBindAppData( )</td>
<td>This method is used to call AWSBindAppData( ) to bind the application data with respect to a business process or a workflow.</td>
</tr>
<tr>
<td>PSZ pszGetBoundDataStream( )</td>
<td>This method is used to return the bound data structure values as a string.</td>
</tr>
<tr>
<td>BoundData( )</td>
<td>This method is the destructor which de-initializes the attributes.</td>
</tr>
</tbody>
</table>
Class Name: STF TX  
Class Description  
This class is used to process the inbound as well as the outbound transactions. It has methods to get the transaction type. It initiates the Transaction class based on the transaction type.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uInOrOutTx</td>
<td>Flag to indicate Inbound or Outbound Transaction</td>
</tr>
<tr>
<td>uTxType</td>
<td>Transaction type</td>
</tr>
<tr>
<td>lTxID</td>
<td>Transaction ID assigned by the workflow server</td>
</tr>
<tr>
<td>bConstFailed</td>
<td>Flag to indicate whether STF_Tx constructor failed or not</td>
</tr>
<tr>
<td>pTransaction</td>
<td>Pointer to Transaction class</td>
</tr>
<tr>
<td>pTransporter</td>
<td>Pointer to STFTransporter class</td>
</tr>
<tr>
<td>pTxParser</td>
<td>Pointer to TxParser class</td>
</tr>
<tr>
<td>pTxFormatter</td>
<td>Pointer to TxFormatter class</td>
</tr>
<tr>
<td>pIniFile</td>
<td>Pointer to INIFile class</td>
</tr>
<tr>
<td>pEnvelope</td>
<td>Pointer to Envelope</td>
</tr>
<tr>
<td>pBD</td>
<td>Pointer to BoundData</td>
</tr>
</tbody>
</table>

Private Methods  
None  

Public Methods  

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxProcessor( )</td>
<td>This method is the constructor of TxProcessor which instantiates STFTransporter, TxParser, TxFormatter and INIFile.</td>
</tr>
<tr>
<td>bProcessInputTx(BOOL *pMore Tx)</td>
<td>This method processes the input Transaction data by instantiating STF_Tx in Input mode.</td>
</tr>
<tr>
<td>bProcessOutputTx()</td>
<td>This method processes the output Transaction data by instantiating STF_Tx in Output mode.</td>
</tr>
</tbody>
</table>

Class Name: Transaction  
Class Description  
This class is the base class for the various types of WEA transaction classes. It has virtual method to process the required Transactions.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSTF_Tx</td>
<td>Pointer to STF_Tx class</td>
</tr>
<tr>
<td>pIniFile</td>
<td>Pointer to INIFile class</td>
</tr>
</tbody>
</table>

Public Methods  

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction(STF_Tx *pSTF_Tx)</td>
<td>This method is the constructor which accepts the pointer to STF_Tx class and substitutes it in pSTF_Tx data member.</td>
</tr>
<tr>
<td>bDoIt()</td>
<td>This virtual method is used to call workflow server API(s) and Formatter to process the Transactions. By default it processes invalid Transactions.</td>
</tr>
</tbody>
</table>

Class Name: TxProcessor  
Class Description  
This is the main controlling class of STF processor. It has pointers to STFTransporter TxParser, TxFormatter and STF_Tx and INIFile classes in the STF Processor. TxProcessor will be instantiated once during an STF Session (in the main program).

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTransporter</td>
<td>Pointer to STFTransporter class</td>
</tr>
<tr>
<td>pTxPars</td>
<td>Pointer to TxParser class</td>
</tr>
</tbody>
</table>

Public Methods  

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction()</td>
<td>This method is the destructor which de-allocates all the memory occupied by data members.</td>
</tr>
</tbody>
</table>
Class Name: STFTxKwd
Class Description
This class defines STF Transaction keywords. This is used by the Transaction class to get keyword string and values for calling workflow server APIs and to send return values of workflow server APIs to TxFormatter. The constructor of this class gets the keyword string from Resource.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSTF_Tx</td>
<td>Pointer to STF_Tx class (to get Parser pointer)</td>
</tr>
<tr>
<td>szKwd</td>
<td>STF Keyword string</td>
</tr>
</tbody>
</table>

Public Methods

- **STFTxKwd(STF_Tx *pSTF_Tx, USHORT uKwdID)**
  This constructor accepts the pointer to STF_Tx class and the keyword ID (defined in the Resource). It fills the szKwd data member corresponding to uKwdID from the Resource.

- **PSZ pszGetKwd()**
  This method returns szKwd stored in the object.

- **VOID vGetKwd(PSZ pszKwd)**
  This method updates pszKwd with szKwd stored in the object. This method assumes that the caller allocates memory for pszKwd.

- **USHORT uGetKwdValue(PSZ pszKwdValueStr)**
  This method outputs the keyword value corresponding to the szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().

The following methods are overloaded for different keyword data types.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USHORT uGetKwdValue(USHORT *puKwd)</td>
<td>This method outputs the keyword value in puKwd corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().</td>
</tr>
<tr>
<td>USHORT uGetKwdValue(INT *piKwd)</td>
<td>This method outputs the keyword value in piKwd corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().</td>
</tr>
<tr>
<td>USHORT uGetKwdValue(LONG *plKwd)</td>
<td>This method outputs the keyword value in plKwd corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().</td>
</tr>
<tr>
<td>USHORT uGetKwdValue(BOOL *pbKwd)</td>
<td>This method outputs the keyword value in pbKwd corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().</td>
</tr>
<tr>
<td>USHORT uGetKwdValue(LPDATEITEMET pDateTime)</td>
<td>This method outputs the keyword value in pDateTime corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue().</td>
</tr>
</tbody>
</table>

Method Name | Method Description |
-------------|--------------------|
USHORT uGetKwdValue(LPORG2ID pOrg2ID, INT *piCount) | This method outputs the keyword values in pOrg2ID and piCount corresponding to the data member szKwd stored in the object using Parser’s method bGetSTFTxKwdValue(). |

- **~STFTxKwd()**
  This method is the destructor which does nothing.

Class Name: InitBP
Class Description
This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Private Methods

- **BOOL bCallAPI()**
  This method is used to call the workflow server API to initiate business process. To get the parameters needed for the workflow server API call, it instantiates the STFTxKwd objects for different parameters.

- **BOOL bCallFormatter(STF_Tx *pSTF_Tx)**
  This method is used to call TxFormatter to format the return values got from workflow server API. Using pSTF_Tx, it gets the pointer to TxFormatter.

Public Methods

- **InitBP(STF_Tx *pSTFTx)**
  This method is the constructor which accepts the pointer to STF_Tx class which will be used by other methods of this class.

- **virtual BOOL bDoIt()**
  This method calls the private methods bCallAPI() and bCallFormatter() to process this Transaction.

- **~InitBP()**
  This method is the destructor which de-initializes all the attributes.

Class Name: InitWF
Class Description
This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.
Class Name: Act
Class Description
This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Private Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallAPI()</td>
<td>This method is used to call the workflow server API to initiate workflow. To get the parameters needed for the workflow server API call, it instantiates the STFTxKwd objects for different parameters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallFormatter(STF_Tx *pSTF_Tx)</td>
<td>This method is used to call TxFormatter to format the return values got from workflow server API. Using pSTF_Tx, it gets the pointer to TxFormatter.</td>
</tr>
</tbody>
</table>

Public Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act(STF_Tx *pSTFTx)</td>
<td>This method is the constructor which accepts the pointer to STF_Tx class which will be used by other methods of this class.</td>
</tr>
<tr>
<td>virtual BOOL bDol()</td>
<td>This method calls the private methods bCallAPI() and bCallFormatter() to process this Transaction.</td>
</tr>
<tr>
<td>~Act()</td>
<td>This method is the destructor which de-initializes all the attributes.</td>
</tr>
</tbody>
</table>

Class Name: BindData
Class Description
This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Private Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallAPI()</td>
<td>This method is used to call the workflow server API to bind the application data to a business process or a workflow. It instantiates Bound Data object to perform this operation.</td>
</tr>
<tr>
<td>BOOL bCallFormatter(STF_Tx *pSTF_Tx)</td>
<td>This method is used to call TxFormatter to format the return values got from workflow server API. Using pSTF_Tx, it gets the pointer to TxFormatter.</td>
</tr>
</tbody>
</table>

Public Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BindData(STF_Tx *pSTFTx)</td>
<td>This method is the constructor which accepts the pointer to STF_Tx class which will be used by other methods of this class.</td>
</tr>
<tr>
<td>virtual BOOL bDol()</td>
<td>This method calls the private methods bCallAPI() and bCallFormatter() to process this Transaction.</td>
</tr>
<tr>
<td>~BindData()</td>
<td>This method is the destructor which de-initializes all the attributes.</td>
</tr>
</tbody>
</table>

Class Name: GetWFStatus
Class Description
This class is derived from Transaction class and has methods to call workflow server API and the TxFormatter.
### Class Name: GetPendingActions

**Class Description**

This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pStatus</td>
<td>Structure which has the Status string</td>
</tr>
<tr>
<td>iActCount</td>
<td>Number of ActInfo structures returned</td>
</tr>
<tr>
<td>ppActInfo</td>
<td>Array of ActInfo structures</td>
</tr>
</tbody>
</table>

**Private Methods**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallAPI( )</td>
<td>This method is used to call the workflow server API to get the list of workflows for which actions are pending.</td>
</tr>
<tr>
<td>BOOL bCallFormatter(STF_Tx *pSTFx)</td>
<td>This method is used to call TxFormatter to format the return values got from workflow server API. Using pSTFx, it gets the pointer to TxFormatter.</td>
</tr>
</tbody>
</table>

**Public Methods**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPendingActions(STF_Tx *pSTFx)</td>
<td>This method is the constructor which accepts the pointer to STF_Tx class which will be used by other methods of this class.</td>
</tr>
<tr>
<td>virtual BOOL bDoIt( )</td>
<td>This method calls the private methods bCallAPI( ) and bCallFormatter( ) to process this Transaction.</td>
</tr>
</tbody>
</table>

**Class Name: GetAvailableBPs**

**Class Description**

This class is derived from Transaction class and it has methods to call workflow server API and the TxFormatter.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iBPCount</td>
<td>Number of Business Process list structures returned</td>
</tr>
<tr>
<td>ppBPList</td>
<td>Array of Business Process structures</td>
</tr>
</tbody>
</table>

**Private Methods**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallAPI( )</td>
<td>This method is used to call the workflow server API to get the list of available business processes.</td>
</tr>
<tr>
<td>BOOL bCallFormatter(STF_Tx *pSTFx)</td>
<td>This method is used to call TxFormatter to format the return values got from workflow server API. Using pSTFx, it gets the pointer to TxFormatter.</td>
</tr>
</tbody>
</table>

**Public Methods**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAvailableBPs(STF_Tx *pSTFx)</td>
<td>This method is the constructor which accepts the pointer to STF_Tx class which will be used by other methods of this class.</td>
</tr>
<tr>
<td>virtual BOOL bDoIt( )</td>
<td>This method calls the private methods bCallAPI( ) and bCallFormatter( ) to process this Transaction.</td>
</tr>
</tbody>
</table>
Whenever a non-fatal error occurs, it will be logged in a virtual array of Actlnfo structures, called for each workflow. beach calls an error formatter to format the return status of the workflow processor. It also calls the TxFormatter to send the WF status data to the WEA.

Class Name: ReturnWFStatus
Class Description
This class is derived from Transaction class and has methods to poll for workflow processor generated notifications and call workflow server API for getting Workflow Status and Bound Data. It also calls the TxFormatter to send the WF status data to the WEA.

### Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszStatus</td>
<td>Structure which has the Status string</td>
</tr>
<tr>
<td>iActCount</td>
<td>Number of Actlnfo structures returned</td>
</tr>
<tr>
<td>ppActInfo</td>
<td>Array of Actlnfo structures</td>
</tr>
</tbody>
</table>

### Private Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL bCallAPI()</td>
<td>This method is called to get the status of the workflow.</td>
</tr>
<tr>
<td>BOOL bCallFormatter(STF _Tx *pSTF_Tx)</td>
<td>This method calls the workflow server API(s) to get the status of the workflow.</td>
</tr>
</tbody>
</table>

### Public Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReturnWFStatus(STF _Tx *pSTF_Tx)</td>
<td>This constructor accepts the pointer to STF_Tx class which will be used by other methods.</td>
</tr>
<tr>
<td>virtual BOOL bDoIt( )</td>
<td>This method polls to Notification events in the STF queue of the server.</td>
</tr>
</tbody>
</table>

Class Name: Error
Class Description
This class is used for error handling. It records the errors during an STF Processor session. The Error object is updated with error information by other objects in the STF Processor whenever an error occurs. The error messages are stored in a Resource file (RC) and is loaded as and when required. Whenever a non-fatal error occurs, it will be logged in an ASCII log file whose path is specified in the STF Processor initialization file (INI file). All fatal errors will be logged and displayed on the STF Processor display as pop-up dialog boxes and after user intervention, the STF Processor will shut down. Please Refer to Sec. 10 for details of error handling in STF Processor. The Error object is Global which is used by all the classes in STF Processor when error logging is required. The format of error logged in the error log file is:

<STF Processor Name>-<Date/Time>-<Errorcode >-<Error Message>  
<STF01>-12-14-92 12:30:0000003168-<Could not find keyword(s):BPNAME>

### Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iErrCode</td>
<td>Error Code (corresponding to string table ID in Resource file)</td>
</tr>
<tr>
<td>pszErrLogStr</td>
<td>Error log string</td>
</tr>
<tr>
<td>bFatalFlag</td>
<td>Whether the error is fatal or not</td>
</tr>
<tr>
<td>szErrLogFile</td>
<td>Error log file path (from INI file)</td>
</tr>
<tr>
<td>pINIFile</td>
<td>pointer to INI file</td>
</tr>
</tbody>
</table>

### Public Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error( )</td>
<td>This method is the constructor which initializes data members.</td>
</tr>
<tr>
<td>VOID vPutError(LONG iErrCode, BOOL bFatalFlag, PSZ pszParam1, PSZ pszParam2, PSZ pszParam3)</td>
<td>This method updates Error code and message and writes it into error log file. The error string pszErrLogStr corresponding to ErrorCode is accessed from the Resource. If bFatalFlag is TRUE, then the error message is logged and then popped up on the screen.</td>
</tr>
<tr>
<td>pszSzGetErrMsg( )</td>
<td>This method gets the error message stored in pszErrLogSt.</td>
</tr>
</tbody>
</table>
workflow transaction information which has been formatted in said predetermined standard transaction format;

b) transaction processor means for i) processing said workflow transaction information which has been received and parsed by said transporter means to prepare said workflow transaction information for sending to and use by an application program interface of said workflow system, and ii) processing workflow transaction information received from said application program interface of said workflow system for sending to said transporter means to prepare said received workflow transaction information for formatting into said predetermined standard transaction format, sending to and use by said workflow enabled application.

2. The system defined by claim 1 wherein said standard transaction format workflow transaction information received from said workflow enabled application is at least one of an action based transaction and a query based transaction.

3. The system defined by claim 1 wherein said standard transaction format workflow transaction information sent to said workflow enabled application is at least one of requested workflow status, returned bound data, returned available business processes, returned pending actions and notification.

4. The system defined by claim 1 wherein said predetermined standard transaction format comprises an envelope and workflow data.

5. The system defined by claim 2 wherein said action based transaction is one of initiate business process, initiate workflow, act in a workflow and bind data.

6. The system defined by claim 2 wherein said query based transaction is one of request workflow status, get available business processes and get pending actions.

7. The system defined by claim 4 wherein said envelope contains address information which is platform and environment dependent.

8. The system defined by claim 4 wherein said workflow data contains workflow specific data and bound process data.

9. The system defined by claim 8 wherein said workflow specific data includes standard transaction format transaction type and standard transaction format identification.

10. The system defined by claim 8 wherein said bound process data are data elements used by a workflow server for management purposes.

11. The system defined by claim 9 wherein said workflow specific data further comprises at least one of workflow participants, workflow type, transaction type, expected workflow completion date, requested workflow completion date and workflow status.

* * * * *