Merchant Integration Guide

ACI Commerce Gateway™

Jun-2006, Release 3.1 Publication Number: EC-AB100-03

Contents

ACI Worldwide Inc.

Preface

The ***ACI Commerce Gateway Merchant Integration Guide*** serves as a collection point for all merchant plug-in details for the ACI Commerce Gateway product.

Audience

The ***ACI Commerce Gateway Merchant Integration Guide*** is mainly intended for merchant developers and integrators who wish to connect to and send transaction information to ACI Commerce Gateway.

Additional Documentation

The ACI Commerce Gateway documentation set is arranged so that each ACI Commerce Gateway manual presents a topic or group of related topics in detail. When one ACI Commerce Gateway manual presents a topic that has already been covered in detail in another ACI Commerce Gateway manual, the topic is summarized and the reader is directed to the other manual for additional information. Information has been arranged in this manner to be more efficient for readers who do not need the additional detail and at the same time provide the source for readers who require the additional information. This manual contains references to the following ACI Commerce Gateway publications:

The ***ACI Commerce Gateway Response and Error Code Support Guide*** contains information about response and error codes.

Software

This manual documents standard processing as of its publication date. Software that is not current and custom software modifications (CSMs) may result in processing that differs from the material presented in this manual. The customer is responsible for identifying and noting these changes.

Publication Identification

Three entries appearing at the bottom of each page uniquely identify this ACI Commerce Gateway publication. The publication number (for example, EC-AB100-03 for the ***ACI Commerce Gateway Merchant Integration Guide***) appears on every page to assist readers in identifying the manual from which a page of information was printed. The publication date (for example, Jun-2006 for June 2006) indicates the issue of the manual. The software release information (for example, R3.1 for release 3.1) specifies the software that the manual describes. This information matches the document information on the copyright page of the manual.

1: BatchPortal Transaction Behavior

This section provides an abstract outline of batch transaction processing to developers so they can focus their attentions on creating transactional objects or implementing an existing e24BatchPipe Object Library. These abstract batch transaction outlines can be implemented somewhat differently, but they provide a foundation for understanding how and what actually happens before, during, and after a transaction.

Refer to the “BatchPortal Raw Interface” section for more information about BatchPortal transaction behavior.

Batch Action 1

Batch Action 1 performs an SSL File Upload of the Batch Transmit File. To perform a Batch Action 1, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=1 (required)
- batchtrackid (required)
- batchdata (required)

Transaction Response Variables

- **Error** - Check response for !ERROR! and examine the error message.
- **batchtrackid** - The original batchtrackid of your submission that is used in all other batch actions.

- **batchid** - The Batch Server issues this referential batchid number to identify the batch.
- **filesize** - The batch transmit file byte size received by the Batch Server. This variable can be used for integrity purposes. If the byte "filesize" is not within a few bytes of the original transmit file, a Batch Action 3 (Request for Cancellation) can be submitted.

Batch Action 2

Batch Action 2 performs an SSL Batch Query and/or File Download of the Batch Response File. To perform a Batch Action 2, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

The Batch Response File is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=2 (required)
- batchtrackid (required)

Transaction Response Variables

- **Error** - Check response for !ERROR! and examine the error message.
- **batchtrackid** - The original batchtrackid of your submission that is used in all other batch actions.
- **batchid** – The Batch Server issues this referential batchid number to identify the batch.
- **filesize** - The batch transmit file byte size received by the Batch Server.
- **batchstatus** - This numeric value indicates the current status of the Batch Transmit File in the Batch Server. The values are defined below:
  - 0 - Waiting In Que - Batch is currently waiting to be processed.
  - 1 - Processing - Batch is currently being processed.

- 2 - Finished - Batch is done processing.
- 3 - Cancelled - The batch was cancelled.
- 4 - Stopped - Batch processing has been paused.
- 5 - Error Occurred - An error occurred during processing.

- **batchstatuspercent** - This numeric value, from 0 to 100, indicates the percentage of the batch file that has been processed. It can be used in automated batch querying mechanisms to determine the amount of time to wait before the next Batch Action 2.
- **batchoutdata** - This data is the actual Batch Response File and only accompanies the Batch Action 2 Query Data if the batchstatus equals 2 or 4. The batchoutdata should be extracted from the Response and dealt with accordingly either by writing it to a file to keep all formatting or parsing it into a data source.

Batch Action 3

Batch Action 3 performs an SSL Query/Request indicating that the original Batch Transmit File should be terminated from its position in the Batch Server. To perform a Batch Action 3, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=3 (required)
- batchtrackid (required)

Transaction Response Variables

- Check response for !ERROR! and examine the error message.
- batchtrackid - The original batchtrackid of your submission that is used in all other batch actions.
- batchid - The Batch Server issues this referential batchid number to identify the batch.

- filesize – The batch transmit file byte size received by the Batch Server.

Batch Action 4

Batch Action 4 performs an SSL Batch Query and/or File Download of the Authorized Batch Response File. To perform a Batch Action 4, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

The Batch Response File containing only the authorized transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=4 (required)
- batchtrackid (required)

Transaction Response Variables

- **Error** - Check response for !ERROR! and examine the error message.
- **batchtrackid** - The original batchtrackid of your submission that is used in all other batch actions.
- **batchid** – The Batch Server issues this referential batchid number to identify the batch.
- **filesize** - The batch transmit file byte size received by the Batch Server.
- **batchstatus** - This numeric value indicates the current status of the Batch Transmit File in the Batch Server. The values are defined below:
  - 0 - Waiting In Que - Batch is currently waiting to be processed.
  - 1 - Processing - Batch is currently being processed.
  - 2 - Finished - Batch is done processing.
  - 3 - Cancelled - The batch was cancelled.
  - 4 - Stopped - Batch processing has been paused.

- 5 - Error Occurred - An error occurred during processing.

- **batchstatuspercent** - This numeric value, from 0 to 100, indicates the percentage of the batch file that has been processed. It can be used in automated batch querying mechanisms to determine the amount of time to wait before the next Batch Action 4.
- **batchoutdata** - This data is the actual Batch Response File and only accompanies the Batch Action 4 Query Data if the batchstatus equals 2, 4, 5 or 6. The batchoutdata should be extracted from the Response and dealt with accordingly either by writing it to a file to keep all formatting or parsing it into a data source.

Batch Action 5

Batch Action 5 performs an SSL Batch Query and/or File Download of the Denied Batch Response File. To perform a Batch Action 5, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

The Batch Response File containing only the denied transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=5(required)
- batchtrackid (required)

Transaction Response Variables

- **Error** - Check response for !ERROR! and examine the error message.
- **batchtrackid** - The original batchtrackid of your submission that is used in all other batch actions.
- **batchid** – The Batch Server issues this referential batchid number to identify the batch.
- **filesize** - The batch transmit file byte size received by the Batch Server.

- **batchstatus** - This numeric value indicates the current status of the Batch Transmit File in the Batch Server. The values are defined below:
  - 0 - Waiting In Que - Batch is currently waiting to be processed.
  - 1 - Processing - Batch is currently being processed.
  - 2 - Finished - Batch is done processing.
  - 3 - Cancelled - The batch was cancelled.
  - 4 - Stopped - Batch processing has been paused.
  - 5 - Error Occurred - An error occurred during processing.
- **batchstatuspercent** - This numeric value, from 0 to 100, indicates the percentage of the batch file that has been processed. It can be used in automated batch querying mechanisms to determine the amount of time to wait before the next Batch Action 5.
- **batchoutdata** - This data is the actual Batch Response File and only accompanies the Batch Action 5 Query Data if the batchstatus equals 2, 4 5 or 6. The batchoutdata should be extracted from the Response and dealt with accordingly either by writing it to a file to keep all formatting or parsing it into a data source.

Batch Action 6

Batch Action 6 performs an SSL Batch Query and/or File Download of the Rejected Batch Response File. To perform a Batch Action 6, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

The Batch Response File containing only the rejected transactions is sent after it is executed. If the Batch Response File is requested while executing, only the response data is sent.

Transaction Transmit Variables

- id (required)
- password (required)
- batchaction=6(required)
- batchtrackid (required)

Transaction Response Variables

- **Error** - Check response for !ERROR! and examine the error message.
- **batchtrackid** - The original batchtrackid of your submission that is used in all other batch actions.
- **batchid** – The Batch Server issues this referential batchid number to identify the batch.
- **filesize** - The batch transmit file byte size received by the Batch Server.
- **batchstatus** - This numeric value indicates the current status of the Batch Transmit File in the Batch Server. The values are defined below:
  - 0 - Waiting In Que - Batch is currently waiting to be processed.
  - 1 - Processing - Batch is currently being processed.
  - 2 - Finished - Batch is done processing.
  - 3 - Cancelled - The batch was cancelled.
  - 4 - Stopped - Batch processing has been paused.
  - 5 - Error Occurred - An error occurred during processing.
- **batchstatuspercent** - This numeric value, from 0 to 100, indicates the percentage of the batch file that has been processed. It can be used in automated batch querying mechanisms to determine the amount of time to wait before the next Batch Action 6.
- **batchoutdata** - This data is the actual Batch Response File and only accompanies the Batch Action 6 Query Data if the batchstatus equals 2, 4 5 or 6. The batchoutdata should be extracted from the Response and dealt with accordingly either by writing it to a file to keep all formatting or parsing it into a data source.

ACI Worldwide Inc.

2: BatchPortal File Format

The BatchPortal File formats are a specific set of protocols and data formatting procedures. Follow these guidelines to ensure transaction compliance and integrity.

BatchPortal File Format Types

Batch Transmit File

The batch transmit file must be generated by the client's commerce system according to the file format map described next. Review the TranPortal set of protocols and procedures if you are unfamiliar with them.

Batch Transmit File Format

Line Terminators: CR, LF, or CRLF

Data Fields: action,card,exp,cvv2,currencycode,transid,zip,addr, member,amt,trackid, UDF1,UDF2,UDF3,UDF4,and UDF5

Batch Response File Format

Line Terminators: CR, LF, or CRLF

Data Fields: All original fields: action,card,exp,cvv2,currency code, transid,zip,addr,member,amt,trackid,UDF1,UDF2,UDF3,UDF4,UDF5
plus: status,error,result,auth,ref,avr,date,brand,transid

Additional Information:

status = 0 (Failure) or 1 (Success)

A status of 1 does not indicate a successfully CAPTURED, PROCESSED, or APPROVED transaction. It might indicate successful transaction with a NOT CAPTURED status. Make sure you evaluate the result of the transaction properly.

error = Text error message available only if status=0(Failure)

3: BatchPortal Raw Interface

The BatchPortal Transaction Processing System provides developers with a method of integrating batch processing of credit card transactions into a variety of software products, websites, server software, and client software across the Internet. The BatchPortal Transaction Processing System also provides the developer with a set of simple protocols for communicating with the BatchPortal Processing Interface.

The BatchPortal Transaction Capturing System can be used to develop an interface or object that provides tools for communicating with the BatchPortal Processing Servers. The BatchPortal Transaction Capturing System is made up of the following components:

- Communication protocols
- Transaction transmit file formats
- Transaction response file formats
- Error messages

All of the BatchPortal Objects or Interfaces have been developed around these specifications in order to provide a higher-level interface.

If building your own interface object, review the following Communication Protocol Specifications section or choose a pre-existing e24BatchPipe Object to incorporate into your commerce project.

Communication Protocol Specifications

- **Protocol**: http
- **Port**: 443 Verisign 3.0 SSL Certificate
- **Target (action)**:
  - https://<host address>/<context>servlet/BatchPortalHTTPServlet
- **Method**: POST

- **Content-Type**: application/www-form-urlencoded or application/x-www-form-urlencoded and ENCTYPE="multipart/form-data" for Batch Action 1 (Batch File Transmissions)
- **Transmit Data Format**: Url Encoded
- **Response Data Format**: Single text string response delimited by colons... string:string:string:
- **Encryption Level**: SSL Version 3.0

The Transaction Processing System requires the developer to communicate a handful of variables across the Internet by way of an http post to the following address:

- https://<hostaddress>/<context>/servlet/BatchPortalHTTPServlet

All posted data must be in the form of an URL Encoded string of name value pairs and the batch file data as "www-form-urlencoded" for Batch Actions 2 and 3 and ENCTYPE="multipart/form-data" for Batch Action 1.

Standard Transaction Example:

Url Encoded Data
id=TranPortal ID&password=password&batchaction=1&batchtrackid=unique tracking id
Multipart Form-Data
batchdata=batchdata type=file

Transmitted Variables

id – The TranPortal identification number issued and used to track the merchant's credit card transactions.

password - The TranPortal Password issued and used to authorize the transaction. Your data will be encrypted and password securely hidden as long as you are issuing an https post.

batchaction - Valid BatchPortal actions are identified below. Use a numerical format only.

1 - Send Batch File For Processing

2 - Query\Receive For Batch Response File

3 - Send Batch Cancellation Request

batchtrackid- A unique tracking ID issued by the merchant's commerce system that is stored with the batch file transaction. Issue unique batch tracking IDs with **batchaction 1** and use your batch tracking IDs to query for batch response files or cancel a processing batch file. (Avoid spaces and extended characters; use only alphanumeric.)

Batch Submission Methods

Batch Action 1 Submission Method

Transmitting a batch for processing requires the following transmitted variables. Query attempts can be made against a submitted batch to determine if the batch has finished processing or what percent of the batch has been processed.

- id
- password
- batchaction
- batchtrackid
- batchdata

Batch Action 2 Submission Method

Requesting a Batch Query\Receive requires the following transmitted variables.

- id
- password
- batchaction
- batchtrackid

Batch Action 3 Submission Method

Transmitting a batch processing cancellation command requires the following transmitted variables. Only the original ID can cancel the submitted batch. Once cancelled, Batch Action 2 (Batch Query\Receive) will transmit the results of the cancelled batch.

- id
- password
- batchaction
- batchtrackid

Batch Transmission File Formats

Note: All lines must be terminated with a CRLF.

Batch Input File Format

- action,card,exp,cvv2,type,transid,zip,addr,member,amt,trackid,UDF1,UDF2,UDF3,UDF4, UDF5

Batch Output File Format

Each line of data is appended to the end of the originating line of input data and returned as a single batch output file.

- ,status,error,result,auth,ref,avr,date,cardtype,transid
- status = 0 (Failure) or 1 (Success)
- error = Text error message available only if status=0(Failure)

A status of 1 does not indicate a successfully CAPTURED, PROCESSED, or APPROVED transaction but rather a successful transaction that could be NOT CAPTURED. Make sure you evaluate the result of the transaction appropriately.

Note: Batch Transmission and Batch Receive Data Files must contain a CRLF, CR, or CRLF at the end of every line.

Batch Response Variables

Every transaction processed through the Transaction Processing System is returned as a single text string. Colons separate the return values of the processed transaction. Each Batch Action returns a different response string. See the examples below:

Batch Action 1 Example:
batchtrackid:batchid:filesize

- **batchtrackid** - The original submitted batchtrackid for purposes of redundancy.
- **batchid** - Issued by the Batch Interface indicating the batch was received successfully.
- **filesize** - Issued by the Batch Interface as a single integer that represents the size of the batch received in bytes. This should be used for redundancy purposes to check for lost transmission data.

Batch Action 2 Example:
batchtrackid:batchid:filesize:batchstatus:batchstatuspercent:batchoutdata

- **batchtrackid** - The original submitted batchtrackid for purposes of redundancy.
- **batchid** - Same as the batchid received during submission of the original batch.
- **filesize** - Issued by the Batch Interface as a single integer that represents the size of the batch received in bytes. This should be used for redundancy purposes to check for lost transmission data.
- **batchstatus** - Issued by the Batch Interface as a single integer that represents the status of the batch during processing. The values are defined below:
  - 0 - Waiting In Queue - Batch is currently waiting to be processed.
  - 1 - Processing - Batch is currently being processed.
  - 2 - Finished - Batch is done processing.
  - 3 - Cancelled - The batch was cancelled.
  - 4 - Stopped - Batch processing has been paused.
  - 5 - Error Occurred - An error occurred during processing.
- **batchstatuspercent** - Issued by the Batch Interface as a single integer that represents the percent of the batch submission that has been processed.
- **batchoutdata** - The resulting batch output data when the batch is finished processing. This data should be archived to a file and stored securely in a data source.

Batch Action 3 Example:
batchtrackid:batchid:filesize

- **batchtrackid** - The original submitted batchtrackid for purposes of redundancy.

- **batchid** - Same as the batchid received during submission of the original batch.
- **filesize** - Issued by the Batch Interface as a single integer that represents the size of the batch received in bytes. This should be used for redundancy purposes to check for lost transmission data.

Error Response Messages - If any errors occur during the transmission of the transaction data, the response format will contain a single string indicating that an error occurred. All response error messages begin with a !ERROR! identifier. The developer should examine the actual response message string and determine if an error has occurred. Refer to the ***ACI Commerce Gateway Response and Error Code Support Guide*** for more information.

Error Response Conditions

Normal Response Condition

- Normal transaction processing
- PerformTransaction returns 0
- Merchant web software site should not need to initiate any special processing.

Commerce Gateway Error Response Condition

- Error transaction processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- The ACI Commerce Gateway encountered an error with either the data validation, database access, or system-related issues.
- Merchant web software site should not need to initiate any special processing.

Plug-in Error Response Condition

- Error DLL processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- e24 dll plugins utilize the Windows HTTP Services (WinHTTP).
- Merchant web software may need to format reversal transaction for:

ERROR_WINHTTP_TIMEOUT and ERROR_WINHTTP_OPERATION_CANCELLED since it is possible that the request message was transmitted and the response results are unknown.

- e24 maps the most common WinHTTP errors into text.

  ErrorMsg Text Common WinHTTP Errors

  "Connection timed out", // ERROR_WINHTTP_TIMEOUT

  "Internal Error", // ERROR_WINHTTP_INTERNAL_ERROR

  "Invalid URL", // ERROR_WINHTTP_INVALID_URL

  "Name not resolved", // ERROR_WINHTTP_NAME_NOT_RESOLVED

  "Cannot Connect", // ERROR_WINHTTP_CANNOT_CONNECT

  "Connection Aborted", // ERROR_WINHTTP_OPERATION_CANCELLED

- All other WinHTTP errors will result in the following ErrorMsg:

  "Connection Failed with error <WinHTTP Error>"

  WinHTTP Errors: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winhttp/http/winhttpsendrequest.asp

ACI Worldwide Inc.

4: BatchPortal C/C++ Plug-in

The e24BatchPipe C++ component is designed for C++ programmers using Visual C++. The component is based on a pure virtual interface to the object (Ie24BatchPipe), obtained by calling a single factory function exported by the DLL (Createe24BatchPipe). The component has the ability to deliver asynchronous status messages during the transaction. Any object that wants to receive these notifications must implement the Ie24BatchPipeStatus interface.

File name: e24BatchPipe

Location: Plugin Downloads

e24BatchPipe C++ Components

class **Ie24BatchPipeStatus**

Purpose:
This is a pure virtual interface. Any class that wants asynchronous status notifications from the e24BatchPipe C++ component must derive from this class and implement its methods.

---

virtual void OnStatusUpdate(const char* pMsg);

Purpose:
This is the method that is called on an object that receives asynchronous status messages during a transaction.

Parameters:
pMsg - a character string pointer that contains the message.

---

class **Ie24BatchPipe**

Purpose:
This is a pure virtual interface. It is implemented by the C++ e24BatchPipe component. It provides methods for accessing the setup parameters and results as properties of the object. An instance of the e24BatchPipe C++ component is created using the factory method named Createe24BatchPipe. An instance of this object cannot be created using the new operator. Likewise, the delete operator cannot be used to delete an instance of the object. Instead, call the Release method to delete an instance of the object.

---

void Release();

Purpose:
This method is used to delete the instance of the object. Remember that the delete operator cannot be used.

---

void SetAlias(LPCSTR alias);

Purpose:
Set the value of the ALIAS object property.

Parameters:
alias - A pointer to a character string that contains the new property value.

---

void SetResourcePath(LPCSTR resourcepath);

Purpose:
Set the value of the resource path object property.

Parameters:
resourcepath - A pointer to a character string that contains the new property value.

---

void SetAction(LPCSTR action);

Purpose:
Set the value of the ACTION object property.

Parameters:
action – A pointer to a character string that contains the new property value.

---

char* GetAction();

Purpose:
Get the value of the ACTION object property.

---

void SetBatchTrackId(LPCSTR trackid);

Purpose:
Set the value of the BATCHTRACKID object property.

Parameters:
trackid - A pointer to a character string that contains the new property value.

---

char* GetBatchTrackId();

Purpose:
Get the value of the TRACKID object property.

---

void SetDataFile(LPCSTR datafile);

Purpose:
Set the value of the BATCHDATAFILE object property. The DATAFILE property contains the full path to the local file that contains the batch data to be processed if the ACTION property equals 1. It contains the file that will contain the batch processing results if the ACTION property equals 2.

Parameters:
datafile - A pointer to a character string that contains the new property value.

---

char* GetDataFile();

Purpose:
Get the value of the BATCHDATAFILE object property.

---

char* GetBatchId();

Purpose:
Get the value of the BATCHID object property.

---

char* GetStatus();

Purpose:
Get the value of the STATUS object property.

---

char* GetStatusPercent();

Purpose:
Get the value of the STATUSPERCENT object property.

---

char* GetOutData();

Purpose:
Get the value of the OUTDATA object property.

---

char* GetErrorMsg();

Purpose:
Get the value of the ErrorMsg object property.

---

short PerformTransaction(Ie24BatchPipeStatus* pStatus);

Purpose:
Performs the transaction.

Parameters:
pStatus – A pointer to the Ie24BatchPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

function Createe24BatchPipe

Purpose:
This is the factory function used to create an instance of the e24BatchPipe C++ component. This is the only way to create the component. The new operator cannot be used to create an instance of the component.

Definition:
Ie24BatchPipe* Createe24BatchPipe();

---

typedef SBFACTORY

Purpose:
This is the typedef of the function pointer to the Createe24BatchPipe function. It can be used by applications that load this DLL dynamically using LoadLibrary, and get the function pointer using GetProcAddress.

Definition:
typedef Ie24BatchPipe* (WINAPI *SBFACTORY)();

ACI Worldwide Inc.

5: BatchPortal Java Class Object

The e24BatchPipe Java Class Object can be used in a variety of developmental environments for a wide range of desktop to web-based applications that are platform independent. The e24BatchPipe Java Class Object is a royalty-free, distributable component that any developer can use to enable their Internet e-Commerce applications or websites for BatchPortal transactions. The menus below document usage and sample applications.

- Class Hierarchy
- Class e24BatchPipe
- Class NotEnoughDataException

Class Hierarchy

- class java.lang.Object
  - class **e24BatchPipe**

Class e24BatchPipe

java.lang.Object

| +--**e24BatchPipe**

---

public class **e24BatchPipe**

extends java.lang.Object

Constructor Summary

e24BatchPipe()

Method Summary

java.lang.String	**getBatchAction()** Get the batch action value.
java.lang.String	**getBatchData**() Get the batch data.
java.lang.String	**getBatchDataFile**() Get the batch data file path and name.
java.lang.String	**getBatchId**() Get the batch id.
java.lang.String	**getBatchTrackId**() Get the batch track ID value.
java.lang.String	**getError**() Get the error if on exists.
java.lang.String	**getFileSize**() Get the batch file size.
java.lang.String	**getStatus**() Get the batch status.
java.lang.String	**getStatusPercent**() Get the percent complete of the batch.
boolean	**performTransaction**() This is the main method.
void	**setBatchAction**(java.lang.String action) Set the batch action value.
void	**setBatchData**(java.lang.String data) Set the batch data value.

void	**setBatchDataFile**(java.lang.String file) Set the batch data file value.
Void	**setBatchTrackId**(java.lang.String tid) Set the batch track ID value.
Void	**setAlias**(java.lang.String alias) Set the terminal alias.
Void	**setResourcePath**(java.lang.String resourcepath) Set the resource path for the Gateway server.

Methods Inherited from Class java.lang.Object

- Clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

e24BatchPipe

public **e24BatchPipe()**

Method Detail

setResourcePath

public void **setResourcePath**(java.lang.String resourcepath)

Set the resource path for the Gateway server.

Parameters:

resourcepath – The resource path of the resource file.

setAlias

public void setAlias(java.lang.String alias)

Set the terminal alias.

Parameters:

alias - The terminal alias value.

---

getBatchAction

public java.lang.String **getBatchAction**()

Get the batch action value.

Returns:

The batch action.

---

setBatchAction

public void **setBatchAction**(java.lang.String action)

Set the batch action value.

Parameters:

action - The password.

---

getBatchTrackId

public java.lang.String **getBatchTrackId**()

Get the batch track ID value.

Returns:

The batch track ID.

---

setBatchTrackId

public void **setBatchTrackId**(java.lang.String tid)

Set the batch track ID value.

Parameters:

tid - The batch track ID.

---

getBatchData

public java.lang.String **getBatchData**()

Get the batch data. This includes all the transactions for a given batch.

Returns:

The batch data.

---

setBatchData

public void **setBatchData**(java.lang.String data)

Set the batch data value. This includes all the transactions for a given batch.

Parameters:

data - The batch data.

---

getBatchDataFile

public java.lang.String **getBatchDataFile**()

Get the batch data file path and name.

Returns:

The batch data file string.

---

setBatchDataFile

public void **setBatchDataFile**(java.lang.String file)

Set the batch data file value.

Parameters:

file - The path and file name of the batch data file.

---

getFileSize

public java.lang.String **getFileSize**()

Get the batch file size.

Returns:

The file size.

---

getError

public java.lang.String **getError**()

Get the error if on exists.

Returns:

The error message, or null if there was no error.

---

getBatchId

public java.lang.String **getBatchId**()

Get the batch id.

Returns:

The batch id.

---

getStatus

public java.lang.String **getStatus**()

Get the batch status.

Returns:

The batch status.

---

getStatusPercent

public java.lang.String **getStatusPercent**()

Get the percent complete of the batch.

Returns:

The percent complete.

---

performTransaction

public boolean **performTransaction**()

throws java.lang.Exception

This is the main method. Calling this method from another class will initiate the e24BatchPipe. Prior to calling this method, you must set all the required fields using the **gets** and **sets**.

Returns:

true if successful; false if an error occurred.

Throws:

java.lang.Exception - if an exception occurred.

Class NotEnoughDataException

java.lang.Object

|

+--java.lang.Throwable

|

+--java.lang.Exception

|

+--NotEnoughDataException

All Implemented Interfaces:

java.io.Serializable

---

public class **NotEnoughDataException**

extends java.lang.Exception

This exception is thrown if ProcessTransaction method of the SecureCharge class is invoked without setting the values of any properties.

See Also:

- Serialized Form

Constructor Summary

NotEnoughDataException()

The empty constructor.

NotEnoughDataException(java.lang.String msg)

The constructor with info.

Methods inherited from class java.lang.Throwable

- fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

---

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Constructor Detail

NotEnoughDataException

public NotEnoughDataException()

The empty constructor.

NotEnoughDataException

public **NotEnoughDataException**(java.lang.String msg)

The constructor with info.

Parameters:

msg - The text message that describes the exception.

6: CFX_e24BatchPipe

CFX_e24BatchPipe allows ColdFusion developers the ability to integrate real-time automated batch credit card processing capabilities into their applications and websites. It is a simple inline tag that requires a handful of variables. When called, the CFX_e24BatchPipe tag initiates an SSL transaction processing pipeline and returns the results of the batch credit card transaction.

File name: e24BatchPipe.dll

Install the CFX_TranPortal

Installation Procedures

1. Copy the file e24BatchPipe.dll to the ColdFusion BIN directory located under the ColdFusion Installation directory. Example: c:\cfusion\bin\.
2. Go to your ColdFusion Documentation. This is normally located at: http://www.yourdomain.com/cfdocs/.
3. Select your ColdFusion Administrator and select the Tags button on your left.
4. Type the following in the form field to add a new CFX tag: CFX_e24BatchPipe. Then, click the Add button.
5. Verify that the location of the e24BatchPipe.dll form field indicates Server Library. Example: c:\cfusion\bin\e24BatchPipe.dll.
6. Verify that the procedure is ProcessTagRequest.

Implement the CFX_e24BatchPipe

CFX_e24BatchPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “BatchPortal Raw Interface” section for information about implementing the CFX_e24BatchPipe tag.

Tag Properties

- <CFX_e24BatchPipe
- resourcepath="Path to generated resource file"
- alias="Terminal alias"
- action="A Valid Transaction Action (1, 2 or 3)"batchtrackid="Unique Tracking ID"
  datafile="The full path to the local file that contains the batch data to be processed if the ACTION property equals 1.
- The full path of the local file to write the batch processing results into if the ACTION property equals 2.">

Tag Results

The following variables will be available to your ColdFusion application or website once the CFX_e24BatchPipe tag has completed its execution.

SC_SUCCESS - A Boolean value indicating if the transaction was successful or not. "Yes" if successful; "No" otherwise.

SC_ERROR - Descriptive text of the error if the transaction was not successful.

SC_BATCHID- Issued Batch ID.

SC_STATUS - Status of a pending batch transaction. This property will be blank if the ACTION property was set to 1.

SC_STATUSPERCENT - Percent complete of a pending batch transaction. This property will be blank if the ACTION property was set to 1.

Retrieving the Results

To retrieve the tag results use a ColdFusion routine similar in respect to the following code:

```
<CFOUTPUT>
<CFIF sc_Success>
Success!<br>
Batchid is #sc_batchid#.<br>
Status is #sc_status#.<br>
StatusPercent is #sc_statuspercent#.<br>
<CFELSE>
Failure!<br>
The error is #sc_error#
</CFIF>
</CFOUTPUT>
```

Security Concerns - The CFX_e24BatchPipe ColdFusion Tag uses Secure Socket Layer communications to transmit all tag properties to the Transaction Processing Network. All transmitted data is safe and secure. However, it is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

ACI Worldwide Inc.

7: TranPortal Transaction Behavior

This section provides an abstract outline of transaction processing to developers so they may focus their attentions on creating transactional objects or implementing an existing e24TranPipe Object Library. These abstract transaction outlines may be implemented somewhat differently, but they provide a foundation for understanding how and what actually happens before, during, and after a transaction.

Refer to the “TranPortal Raw Interface” section for more information about TranPortal Transaction Behavior.

Purchase Abstract

To perform a purchase, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=1 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip

- amt (required)
- currencycode
- type
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Credit Abstract

To perform a credit, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=2 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)
- currencycode
- type
- transid
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance

- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Purchase Abstract

To perform a void purchase, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=3 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)
- currencycode
- type

- transid (required) - Transid must be the transaction ID of the original purchase.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Authorization Abstract

To perform an Authorization, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=4 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)
- currencycode
- type
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance

- AVR - Note: Authorizations are generally performed to evaluate the AVR Response. If the AVR Response is acceptable, then a subsequent Capture is performed.
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Capture Abstract

To perform a Capture, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=5 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)

- currencycode
- type
- transid (required) - Transid must be the transaction ID of the original authorization.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Credit Abstract

To perform a Void Credit, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=6 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)
- currencycode
- type
- transid (required) - Transid must be the transaction ID of the original credit.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message

- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Capture Abstract

To perform a Void Capture, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=7 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr

- zip
- amt (required)
- currencycode
- type
- transid (required) - Transid must be the transaction ID of the original capture.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Authorization Abstract

To perform a Void Authorization, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=9 (required)
- card (required)
- expyear (required)
- expmonth (required)
- expday
- cvv2
- member (required)
- addr
- zip
- amt (required)
- currencycode
- type
- transid (required) - Transid must be the transaction ID of the original authorization.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

ACI Worldwide Inc.

8: TranPortal Raw Interface

Communication Protocol Specifications

- **Protocol**: http
- **Port**: 443 Verisign 3.0 SSL Certificate
- **Target**:
  - URL Encoded (legacy):

    https://<host address>/<context>/servlet/TranPortalHTTPServlet
  - XML (new format support by plug-ins)

    https://<host address>/<context>/servlet/TranPortalXMLServlet
- **Method**: POST
- **Content-Type**
  - URL Encoded (legacy): application/www-form-urlencoded or application/x-www-form-urlencoded
  - XML (new): application/xml
- **Transmit Data Format**
  - Url Encoded (legacy)
  - XML (new).
- **Response Data Format**
  - **URL Encoded (legacy)**: Single text string response delimited by colons . . . string:string:string:
  - **XML (new)**: <response element tag>response data</response element tag>
- **Encryption Level**: SSL Version 3.0

The Transaction Capturing System requires the developer to communicate a handful of variables across the Internet by way of an http post to the following address:

URL Encoded Transaction Example

https://<host address>/<context>/servlet/TranPortalHTTPServlet

id=2000&password=password&card=4444333322221111&cvv2=149&expYear=2005&expMonth=12&expDay=31&action=1&type=CC&zip=87120&addr=2712 North 161 Street&member=John William Doe&amt=65.00&currencycode=840&trackid=100000&udf1=AA&udf2=BB&udf3=CC&udf4=DD&udf5=EE

XML Transaction Example

https://<host address>/<context>/servlet/TranPortalXMLServlet

<id>2000</id><password>password</password><card>4444333322221111</card><cvv2>149</cvv2><expYear>2005</expYear><expMonth>12</expMonth><expDay>31</expDay><action>1</action><type>CC</type><zip>87120</zip><addr>2712 North 161 Street</addr><member>John William Doe</member><amt>65.00</amt><currencycode>840</currencycode><trackid>100000</trackid><udf1>AA</udf1><udf2>BB</udf2><udf3>CC</udf3><udf4>DD</udf4><udf5>EE</udf5>

Transaction Transmit Variables and Definitions

id - TranPortal Identification Number. The ACI Commerce Gateway system administrator issues the TranPortal ID to identify the merchant and terminal for transaction processing.

password - TranPortal password. The ACI Commerce Gateway system administrator issues the TranPortal password to authenticate the merchant and terminal. Your data will be encrypted and password securely hidden as long as you are issuing an https post.

action - The following are valid actions and they must always be in numeric format.

- Purchase - 1
- Credit - 2

- Void Purchase - 3
- Authorization - 4
- Capture – 5
- Void Credit - 6
- Void Capture - 7
- Void Authorization - 9

card - The credit card number.

expyear - Expiration date year (must be in number format YYYY).

expmonth - Expiration date month (must be in number format MM).

expday - Expiration date day (must be in number format DD).

cvv2 - CVV2 code (must be in number format).

member - Cardholder's full name.

addr - Street address of the consumer.

zip - ZIP Code of the consumer.

amt - The amount of the transaction.

currencycode - The currency code of the transaction.

type - The payment instrument type. Currently, Credit Card ("CC") and Stored Value ("SV") are supported.

transid - Transaction ID used to identify the original transaction for a void and capture transaction.

trackid - A unique tracking ID issued by the merchant's commerce system which is stored with the transaction. (Avoid spaces and extended characters. Use only alphanumeric format.)

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Transaction Response Variables

Every transaction processed through the Transaction Capturing System is returned as a single text string. The return values of the processed transaction separated (delimited) by colons. The developer is responsible for parsing this text string into an employable object or type.

Example:

URL Encoded Result:
Auth:Ref:CardBalance:AVR:Date:TransId:TrackId:UDF1:UDF2:UDF3:UDF4:UDF5

XML Result: <result>CAPTURED</result><auth>999999</auth><ref>504487010197</ref><cardbalance>$123.00</cardbalance><avr>N</avr><postdate>0213</postdate><tranid>4874570301450440</tranid><trackid>100000</trackid><udf1>AA</udf1><udf2>BB</udf2><udf3>CC</udf3><udf4>DD</udf4><udf5>EE</udf5>

Response Definitions

Result - Returned as the transaction response evaluator. Check the Result for error and then evaluate the transaction to determine if it performed successfully.

- CAPTURED - Transaction was captured.
- APPROVED - Transaction was approved.
- VOIDED - Transaction was voided.
- NOT CAPTURED - Transaction was not captured.
- NOT APPROVED - Transaction was not approved.
- NOT VOIDED - Transaction was not voided
- DENIED BY RISK - Risk denied the transaction.
- FAILED AVS - Transaction did not pass Address Verification.

- HOST TIMEOUT - The authorization system did not respond within the timeout limit.

Auth - The resulting authorization number of the transaction.

Ref - The resulting reference number of the transaction. This number or series of letters is used for referential purposes by some acquiring institutions and should be stored properly.

CardBalance - The resulting card balance of the stored value transaction instrument. This result field will be transmitted only when the type field in the request contain a Stored Value (“SV”) indicator.

AVR - A single letter providing information about the cardholder’s submitted data. The letter indicates how closely the submitted card number, address, and ZIP Code match at the card-issuing bank.

- A - Address matches.
- E - Address match error.
- N - No address match.
- R - AVS not available.
- S - Service not supported.
- U - Address match not capable.
- W - 9 digit ZIP Code matches.
- X - Address and 9 digit ZIP Code matches.
- Y - Address and 5 digit ZIP Code matches.
- Z - 5 digit ZIP Code matches.
- 0 - Address could not be verified.

Date - Transaction date in the format of the authorization system.

TransId - Unique transaction ID issued by the ACI Commerce Gateway.

TrackId - The track ID sent by the merchant in the transaction request.

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Error Response Messages - If any errors occur during the transmission of the transaction data, the response format will contain a single string indicating that an error occurred. All response error messages begin with a !ERROR! identifier. The developer should examine the actual response message string and determine if an error has occurred. Refer to the ***ACI Commerce Gateway Response and Error Code Support Guide*** for more information.

Error Response Conditions

Normal Response Condition

- Normal transaction processing
- PerformTransaction returns 0
- Merchant web software site should not need to initiate any special processing.

Commerce Gateway Error Response Condition

- Error transaction processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- The ACI Commerce Gateway encountered an error with either the data validation, database access, or system-related issues.
- Merchant web software site should not need to initiate any special processing.

Example:

XML Response: <error_code_tag>GW00165</error_code_tag><error_service_tag>null</error_service_tag>

<error_text>!ERROR!-GW00165-Invalid Track ID data.</error_text>

<error_code_tag> and <error_text> are documented in the ***ACI Commerce Gateway Response and Error Code Support Guide***.

Plug-in Error Response Condition

- Error DLL processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- e24 dll plugins utilize the Windows HTTP Services (WinHTTP).
- Merchant web software may need to format reversal transaction for:
- ERROR_WINHTTP_TIMEOUT and ERROR_WINHTTP_OPERATION_CANCELLED since it is possible that the request message was transmitted and the response results are unknown.
- e24 maps the most common WinHTTP errors into text.

  ErrorMsg Text Common WinHTTP Errors

  "Connection timed out", // ERROR_WINHTTP_TIMEOUT

  "Internal Error", // ERROR_WINHTTP_INTERNAL_ERROR

  "Invalid URL", // ERROR_WINHTTP_INVALID_URL

  "Name not resolved", // ERROR_WINHTTP_NAME_NOT_RESOLVED

  "Cannot Connect", // ERROR_WINHTTP_CANNOT_CONNECT

  "Connection Aborted", // ERROR_WINHTTP_OPERATION_CANCELLED
- All other WinHTTP errors will result in the following ErrorMsg:

  "Connection Failed with error <WinHTTP Error>"

  WinHTTP Errors: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winhttp/http/winhttpsendrequest.asp

ACI Worldwide Inc.

9: TranPortal Visual C/C++ Plug-in

The TranPortal C++ component is designed for C++ programmers using Visual C++. The component is based on a pure virtual interface to the object (Ie24TranPipe), obtained by calling a single factory function exported by the DLL (Createe24TranPipe). The component has the ability to deliver asynchronous status messages during the transaction. Any object that wants to receive these notifications must implement the Ie24TranPipe Status interface.

The interface described for all of the get methods is using the IQuery Interface. If a different interface is needed, reference the associated .tld, .h, and idl files which are included in the Plugin Download that can be obtained from the ACI Commerce Gateway Merchant Website under the Developers menu.

File name: e24TranPipe.dll

Location: Plugin Downloads

e24TranPipe C++ Components

class **Ie24TranPipeStatus**

Purpose:
This is a pure virtual interface. Any class that wants asynchronous status notifications from the e24TranPipe C++ component must derive from this class and implement its methods.

---

virtual void OnStatusUpdate(const char* pMsg);

Purpose:
This is the method that is called on an object that receives asynchronous status messages during a transaction.

Parameters:
pMsg – A character string pointer that contains the message.

---

class **Ie24TranPipe**

Purpose:
This is a pure virtual interface. It is implemented by the C++ e24TranPipe component. It provides methods for accessing the setup parameters and results, all as properties of the object. An instance of the e24TranPipe C++ component is created using the factory method named Createe24TranPipe. An instance of this object cannot be created using the new operator. Likewise, the delete operator cannot be used to delete an instance of the object. Instead, call the Release method to delete an instance of the object.

---

void Release();

Purpose:
This method is used to delete the instance of the object. Remember that the delete operator cannot be used.

---

void SetAlias(LPCSTR alias);

Purpose:
Set the value of the ALIAS object property.

Parameters:
alias - A pointer to a character string that contains the new property value.

---

void SetResourcePath(LPCSTR resourcepath);

Purpose:
Set the value of the resource path object property.

Parameters:
resourcepath - A pointer to a character string that contains the new property value.

---

void SetAction(LPCSTR action);

Purpose:
Set the value of the ACTION object property.

Parameters:
action - A pointer to a character string that contains the new property value.

---

void GetAction(char* buf, int bufsize);

Purpose:
Get the value of the ACTION object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCard(LPCSTR card);

Purpose:
Set the value of the CARD object property.

Parameters:
card - A pointer to a character string that contains the new property value.

---

void GetCard(char* buf, int bufsize);

Purpose:
Get the value of the CARD object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpYear(LPCSTR expYear);

Purpose:
Set the value of the EXPYEAR object property.

Parameters:
expYear - A pointer to a character string that contains the new property value.

---

void GetExpYear(char* buf, int bufsize);

Purpose:
Get the value of the EXPYEAR object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpMonth(LPCSTR expMonth);

Purpose:
Set the value of the EXPMONTH object property.

Parameters:
expMonth - A pointer to a character string that contains the new property value.

---

void GetExpMonth(char* buf, int bufsize);

Purpose:
Get the value of the EXPMONTH object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpDay(LPCSTR expDay);

Purpose:
Set the value of the EXPDAY object property.

Parameters:
expDay - A pointer to a character string that contains the new property value.

---

void GetExpDay(char* buf, int bufsize);

Purpose:
Get the value of the EXPDAY object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCVV2(LPCSTR cvv2);

Purpose:
Set the value of the CVV2 object property.

Parameters:
cvv2 - A pointer to a character string that contains the new property value.

---

void GetCVV2(char* buf, int bufsize);

Purpose:
Get the value of the CVV2 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetMember(LPCSTR member);

Purpose:
Set the value of the MEMBER object property.

Parameters:
member - A pointer to a character string that contains the new property value.

---

void GetMember(char* buf, int bufsize);

Purpose:
Get the value of the MEMBER object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetAddr(LPCSTR addr);

Purpose:
Set the value of the ADDR object property.

Parameters:
addr - A pointer to a character string that contains the new property value.

---

void GetAddr(char* buf, int bufsize);

Purpose:
Get the value of the ADDR object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetZip(LPCSTR zip);

Purpose:
Set the value of the ZIP object property.

Parameters:
zip - A pointer to a character string that contains the new property value.

---

void GetZip(char* buf, int bufsize);

Purpose:
Get the value of the ZIP object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetAmt(LPCSTR amt);

Purpose:
Set the value of the AMT object property.

Parameters:
amt - A pointer to a character string that contains the new property value.

---

void GetAmt(char* buf, int bufsize);

Purpose:
Get the value of the AMT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCurrency(LPCSTR currency);

Purpose:
Set the value of the CURRENCY object property.

Parameters:
currency - A pointer to a character string that contains the new property value.

---

void GetCurrency(char* buf, int bufsize);

Purpose:
Get the value of the CURRENCY object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetType(LPCSTR type);

Purpose:
Set the value of the TYPE object property.

Parameters:
type - A pointer to a character string that contains the new property value.

---

void SetTransId(LPCSTR transid);

Purpose:
Set the value of the TRANSID object property.

Parameters:
transid - A pointer to a character string that contains the new property value.

---

virtual void GetTransId(char* buf, int bufsize);

Purpose:
Get the value of the TRANSID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetTrackId(LPCSTR trackid);

Purpose:
Set the value of the TRACKID object property.

Parameters:
trackid - A pointer to a character string that contains the new property value.

---

void GetTrackId(char* buf, int bufsize);

Purpose:
Get the value of the TRACKID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUdf1(LPCSTR udf1);

Purpose:
Set the value of the UDF1 object property.

Parameters:
udf1 - A pointer to a character string that contains the new property value.

---

void GetUdf1(char* buf, int bufsize);

Purpose:
Get the value of the UDF1 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUdf2(LPCSTR udf2);

Purpose:
Set the value of the udf2 object property.

Parameters:
udf2 - A pointer to a character string that contains the new property value.

---

void GetUdf2(char* buf, int bufsize);

Purpose:
Get the value of the udf2 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUdf3(LPCSTR udf3);

Purpose:
Set the value of the udf3 object property.

Parameters:
udf3 - A pointer to a character string that contains the new property value.

---

void GetUdf3(char* buf, int bufsize);

Purpose:
Get the value of the udf3 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUdf4(LPCSTR udf4);

Purpose:
Set the value of the udf4 object property.

Parameters:
udf4 - A pointer to a character string that contains the new property value.

---

void GetUdf4(char* buf, int bufsize);

Purpose:
Get the value of the udf4 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUdf5(LPCSTR udf5);

Purpose:
Set the value of the udf5 object property.

Parameters:
udf5 - A pointer to a character string that contains the new property value.

---

void GetUdf5(char* buf, int bufsize);

Purpose:
Get the value of the udf5 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetResult(char* buf, int bufsize);

Purpose:
Get the value of the RESULT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetAuth(char* buf, int bufsize);

Purpose:
Get the value of the AUTH object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

virtual void GetRef(char* buf, int bufsize);

Purpose:
Get the value of the REF object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void getCardBalance(char* buf, int bufsize)

Purpose:
Get the value of the CARDBALANCE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetAvr(char* buf, int bufsize);

Purpose:
Get the value of the AVR object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetDate(char* buf, int bufsize);

Purpose:
Get the value of the DATE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetErrorMsg(char* buf, int bufsize);

Purpose:
Get the value of the ErrorMsg object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetRawResponse(char* buf, int bufsize);

Purpose:
Get the value of the RawResponse object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

short PerformTransaction(Ie24TranPipeStatus* pStatus);

Purpose:
Performs the transaction.

Parameters:
pStatus – A pointer to the Ie24TranPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

function Createe24TranPipe

Purpose:
This is the factory function used to create an instance of the e24TranPipe C++ component. This is the only way to create the component. The new operator cannot be used to create an instance of the component.

Definition:
Ie24TranPipe* Createe24TranPipe();

---

typedef SCFACTORY

Purpose:
This is the typedef of the function pointer to the Createe24TranPipe function. It can be used by applications that load this DLL dynamically using LoadLibrary and get the function pointer using GetProcAddress.

Definition:
typedef Ie24TranPipe* (WINAPI *SCFACTORY)();

10: TranPortal Java Class Object

The e24TranPipe Java Class Object and Source Code can be used in a variety of development environments for a wide range of desktop to web-based applications that are platform independent. The e24TranPipe Java Class Object is a royalty-free distributable component source code that any developer can use to enable their Internet e-Commerce applications or websites for TranPortal transactions. The menu below provides you with documentation on usage and sample applications.

- **Class Hierarchy**
- **Class e24TranPipe**
- **Class NotEnoughDataException**

Class Hierarchy

- class java.lang.Object
  - class **e24TranPipe**
  - class java.lang.Throwable (implements java.io.Serializable)

    class java.lang.Exception

    class NotEnoughDataException

Class e24TranPipe

java.lang.Object

|

+--**e24TranPipe**

---

public final class **e24TranPipe**

extends java.lang.Object

This is the class. It is the primary class used for accessing the Portal backend system. The class is marked as final because it should not be subclassed. The class is thread-safe via synchronized methods.

Field Summary

static int	**FAILURE** Indicates failure.
static int	**SUCCESS** Indicates success.

Constructor Summary

- e24TranPipe()

Method Summary

void	**clearAllFields**() Clears all of the property values to empty strings.
void	**clearResults**() Clears all of the result property values to empty strings.

java.lang.String	**getAction**() Get the value of the Secure-Bank action.
java.lang.String	**getAddress**() Get the value of the Street address.

java.lang.String	**getAmt**() Get the value of the Amount of the transaction.
java.lang.String	**getAuth**() Get the value of the Authorization number.
java.lang.String	**getAvr**() Get the value of the Address verification response.
java.lang.String	**getCard**() Get the value of the Credit card number.
java.lang.String	**getCardBalance**() Get the value of the card balance of the Stored Value card.
java.lang.String	**getCurrencyCode**() Get the value of the currency code.
java.lang.String	**getCvv2**() Get the CVV2 code.
java.lang.String	**getDate**() Get the value of the Transaction Date.
java.lang.String	**getDebugMsg**() Get the debug message.
java.lang.String	**getErrorMsg**() Get the description of the error if the transaction was not successful.
java.lang.String	**getExpDay**() Get the value of the Credit card expiration day.
java.lang.String	**getExpMonth**() Get the value of the Credit card expiration month.
java.lang.String	**getExpYear**() Get the value of the Credit card expiration year.
java.lang.String	**getMember**() Get the value of the Credit cardholder's full name.

java.lang.String	**getRef**() Get the value of the Reference number.
java.lang.String	**getResult**() Get the value of the Transaction response.
java.lang.String	**getTrackId**() Get the value of the Tracking ID

java.lang.String	**getTransId**() Get the value of the Transaction ID.
java.lang.String	**getUdf1**() Get the value of the User-defined field 1.
java.lang.String	**getUdf2**() Get the value of the User-defined field 2.
java.lang.String	**getUdf3**() Get the value of the User-defined field 3.
java.lang.String	**getUdf4**() Get the value of the User-defined field 4.
java.lang.String	**getUdf5**() Get the value of the User-defined field 5.
java.lang.String	**getZip**() Get the value of the ZIP Code.
short	**performTransaction**() Performs the transaction based on the property values that have been set.
void	**setAction**(java.lang.String action) Set the Secure-Bank action.
void	**setAlias**(java.lang.String addr) Set the terminal alias
void	**setAmt**(java.lang.String amt) Set the amount of the transaction.
void	**setCard**(java.lang.String card) Set the credit card number.

Void	**setType**(java.lang.String type) Set the Payment Instrument Type.

void	**setCurrencyCode**(java.lang.String currency) Set the currency code.
void	**setCvv2**(java.lang.String cvv2) Set the CVV2 code.
void	**setExpDay**(java.lang.String expday) Set the Credit card expiration day.
void	**setExpMonth**(java.lang.String expmonth) Set the Credit card expiration month.
void	**setExpYear**(java.lang.String expyear) Set the Credit card expiration year.
void	**setMember**(java.lang.String member) Set the Credit cardholder's full name.
void	**setResourcePath**(java.lang.String resourcepath) Set the Resource path.
void	**setRef**(java.lang.String ref) Set the Reference number.
void	**setSSL**(int ssl) Set the SSL Flag (0 = off; 1 = on).
void	**setTrackId**(java.lang.String trackid) Set the Tracking ID.
void	**setTransId**(java.lang.String transid) Set the Transaction ID.
void	**setUdf1**(java.lang.String udf1) Set the User-defined field 1.
void	**setUdf2**(java.lang.String udf2) Set the User-defined field 2.
void	**setUdf3**(java.lang.String udf3) Set the User-defined field 3.

void	**setUdf4**(java.lang.String udf4) Set the User-defined field 4.

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

Field Detail

SUCCESS

public static final int **SUCCESS**

Indicates success.

FAILURE

public static final int **FAILURE**

Indicates failure.

Constructor Detail

e24TranPipe

public **e24TranPipe**()

Method Detail

clearAllFields

public void clearAllFields()

Clears all of the property values to empty strings.

---

clearResults

public void **clearResults**()

Clears all of the result property values to empty strings.

---

getAction

public java.lang.String **getAction**()

Get the value of the action.

Returns:

The value of the action.

---

getAddress

public java.lang.String **getAddress**()

Get the value of the Street address.

Returns:

The value of the Street address.

---

getAmt

public java.lang.String **getAmt**()

Get the value of the Amount of the transaction.

Returns:

The value of the Amount of the transaction.

---

getAuth

public java.lang.String **getAuth**()

Get the value of the Authorization number.

Returns:

The value of the Authorization number.

---

getAvr

public java.lang.String **getAvr**()

Get the value of the Address verification response.

Returns:

The value of the Address verification response.

---

getCard

public java.lang.String **getCard**()

Get the value of the Credit card number.

Returns:

The value of the Credit card number.

---

getCardBalance

public java.lang.String **getCardBalance**()

Get the value of the card balance of the Stored Value card.

Returns:

The value of the Card Balance.

---

getCurrencyCode

public java.lang.String **getCurrencyCode**()

Get the value of the currency code.

Returns:

The value of the currency code.

---

getCvv2

public java.lang.String **getCvv2**()

Get the CVV2 code.

Returns:

The value of the CVV2 code.

---

getDate

public java.lang.String **getDate**()

Get the value of the Transaction Date.

Returns:

The value of the Transaction Date.

---

getDebugMsg

public java.lang.String **getDebugMsg**()

Get the debug message.

Returns:

The value of the debug message.

---

getErrorMsg

public java.lang.String **getErrorMsg**()

Get the description of the error if the transaction was not successful.

Returns:

The description of the error if the transaction was not successful.

---

getExpDay

public java.lang.String **getExpDay**()

Get the value of the Credit card expiration day.

Returns:

The value of the Credit card expiration day.

---

getExpMonth

public java.lang.String **getExpMonth**()

Get the value of the Credit card expiration month.

Returns:

The value of the Credit card expiration month.

---

getExpYear

public java.lang.String **getExpYear**()

Get the value of the Credit card expiration year.

Returns:

The value of the credit card expiration year.

---

getMember

public java.lang.String **getMember**()

Get the value of the Credit cardholder's full name.

Returns:

The value of the credit cardholder's full name.

---

getRef

public java.lang.String **getRef**()

Get the value of the reference number.

Returns:

The value of the reference number.

---

getResult

public java.lang.String **getResult**()

Get the value of the transaction response.

Returns:

The value of the transaction response.

---

getTrackId

public java.lang.String **getTrackId**()

Get the value of the tracking ID.

Returns:

The value of the tracking ID.

---

getTransId

public java.lang.String **getTransId**()

Get the value of the transaction ID.

Returns:

The value of the transaction ID.

---

getUdf1

public java.lang.String **getUdf1**()

Get the value of the user-defined field 1.

Returns:

The value of the user-defined field 1.

---

getUdf2

public java.lang.String **getUdf2**()

Get the value of the user-defined field 2.

Returns:

The value of the user-defined field 2.

getUdf3

public java.lang.String **getUdf3**()

Get the value of the user-defined field 3.

Returns:

The value of the user-defined field 3.

---

getUdf4

public java.lang.String **getUdf4**()

Get the value of the user-defined field 4.

Returns:

The value of the user-defined field 4.

---

getUdf5

public java.lang.String **getUdf5**()

Get the value of the user-defined field 5.

Returns:

The value of the user-defined field 5.

---

getZip

public java.lang.String **getZip**()

Get the value of the ZIP Code.

Returns:

The value of the ZIP Code.

---

performTransaction

public short **performTransaction**()

throws NotEnoughDataException

Performs the transaction based on the property values that have been set.

Returns:

e24TranPipe.SUCCESS if successful; otherwise e24TranPipe.FAILURE.

Throws:

NotEnoughDataException - Thrown if no property values have been set.

---

setAction

public void **setAction**(java.lang.String action)

Set the action.

Parameters:

action - The new value of the action.

---

setAddress

public void **setAddress**(java.lang.String addr)

Set the street address.

Parameters:

addr - The new value of the street address.

---

setAmt

public void **setAmt**(java.lang.String amt)

Set the amount of the transaction.

Parameters:

amt - The new value of the amount of the transaction.

---

setCard

public void **setCard**(java.lang.String card)

Set the credit card number.

Parameters:

card - The new value of the credit card number.

---

setAlias

public void **setAlias**(java.lang.String alias)

Set the terminal alias value.

Parameters:

alias - The new value of the alias.

---

setCurrencyCode

public void **setCurrencyCode**(java.lang.String currency)

Set the currency code.

Parameters:

currency - The new value of the currency code.

---

setCvv2

public void **setCvv2**(java.lang.String cvv2)

Set the CVV2 code.

Parameters:

cvv2 - The new value of the CVV2 code.

---

setExpDay

public void **setExpDay**(java.lang.String expday)

Set the credit card expiration day.

Parameters:

expday - The new value of the credit card expiration day.

---

setExpMonth

public void **setExpMonth**(java.lang.String expmonth)

Set the credit card expiration month.

Parameters:

expmonth - The new value of the credit card expiration month.

---

setExpYear

public void **setExpYear**(java.lang.String expyear)

Set the credit card expiration year.

Parameters:

expyear - The new value of the Credit card expiration year.

---

setType

public void **setType**(java.lang.String type)

Set the payment instrument type.

Parameters:

type - The new value of the payment instrument type.

---

setMember

public void **setMember**(java.lang.String member)

Set the credit cardholder's full name.

Parameters:

member - The new value of the Credit cardholder's full name.

---

setResourcePath

public void **setResourcePath**(java.lang.String resourcepath)

Set the resourcepath.

Parameters:

resourcepath - The new value of the resourcepath.

---

setRef

public void **setRef**(java.lang.String ref)

Set the Reference number.

Parameters:

ref - The new value of the Reference number.

---

setSSL

public void **setSSL**(int ssl)

Set the SSL flag (0 = off; 1 = on).

Parameters:

ssl - The new value of the SSL flag.

---

setTrackId

public void **setTrackId**(java.lang.String trackid)

Set the Tracking ID.

Parameters:

trackid - The new value of the Tracking ID.

---

setTransId

public void **setTransId**(java.lang.String transid)

Set the Transaction ID.

Parameters:

transid - The new value of the Transaction ID.

setUdf1

public void **setUdf1**(java.lang.String udf1)

Set the user-defined field 1.

Parameters:

udf1 - The new value of the user-defined field 1.

---

setUdf2

public void **setUdf2**(java.lang.String udf2)

Set the user-defined field 2.

Parameters:

udf2 - The new value of the user-defined field 2.

---

setUdf3

public void **setUdf3**(java.lang.String udf3)

Set the user-defined field 3.

Parameters:

udf3 - The new value of the user-defined field 3.

---

setUdf4

public void **setUdf4**(java.lang.String udf4)

Set the user-defined field 4.

Parameters:

udf4 - The new value of the user-defined field 4.

---

setUdf5

public void **setUdf5**(java.lang.String udf5)

Set the user-defined field 5.

Parameters:

udf5 - The new value of the user-defined field 5.

---

setZip

public void **setZip**(java.lang.String zip)

Set the ZIP Code.

Parameters:

zip - The new value of the ZIP Code.

Class NotEnoughDataException

java.lang.Object

|

+--java.lang.Throwable

|

+--java.lang.Exception

|

+--**NotEnoughDataException**

All Implemented Interfaces:

java.io.Serializable

---

public class **NotEnoughDataException**

extends java.lang.Exception

This exception is thrown if the ProcessTransaction method of the e24TranPipe class is invoked without setting the values of any properties.

See Also:

- Serialized Form

Constructor Summary

NotEnoughDataException()
The empty constructor

NotEnoughDataException(java.lang.String msg)
The constructor with info.

Methods inherited from class java.lang.Throwable

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Constructor Detail

NotEnoughDataException

public NotEnoughDataException()

The empty constructor.

NotEnoughDataException

public **NotEnoughDataException**(java.lang.String msg)

The constructor with info.

Parameters:

msg - The text message that describes the exception.

11: CF_e24TranPipe

CF_e24TranPipe allows ColdFusion developers to integrate real-time automated credit card processing capabilities into their applications and websites. It is an inline tag that requires a handful of variables. When called, the CF_e24TranPipe tag initiates a transaction processing pipeline and returns the results of the credit card transaction.

File name: e24TranPipe.cfm

Install the e24TranPipe.cfm

Copy the .cfm File

Copy the file e24TranPipe.cfm to the CustomTags directory located under the ColdFusion Installation directory. Example: **c:\cfusion\CustomTags**

Implement the CF_e24TranPipe

CF_e24TranPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “TranPortal Raw Interface” section for more information about implementing the CF_e24TranPipe tag.

Tag Properties

<CF_e24TranPipe
id="TranPortal ID"
password="TranPortal Password"
action="A Valid Transaction Action"
card="Credit Card To Debit"
expyear="Card Expiration Year (YYYY)"
expmonth="Card Expiration Month (MM)"
expday="Card Expiration Day (DD)"
CVV2="Card CVV2 Code"
member="Cardholder Name"
addr="Cardholder Address"
zip="Cardholder ZIP Code"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"
type=”CC”

transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion application or website once the CF_e24TranPipe Tag has completed its execution. When the CF_e24TranPipe tag has finished execution, the variable **Success** will return a value of 0 (Failure) or 1 (Success). If you receive a **Success** value of 0, you should evaluate the **Error** result.

The results are dependent upon:

- SUCCESS (0 or 1)
- ERROR
- RESULT
- AUTH
- REF

- CARDBALANCE
- AVR
- DATE
- TRANSID
- TRACKID
- UDF1
- UDF2
- UDF3
- UDF4
- UDF5

Test the CF_e24TranPipe Tag

A very simple application page has been provided to test your tag. It indicates how to initiate the tag and sends a formatted transaction result to the browser.

Security Concerns - The CF_e24TranPipe Cold Fusion Tag uses Secure Socket Layer communications to transmit tag properties to the Transaction Processing Network. All transmitted data is safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

ACI Worldwide Inc.

12: CFX_e24TranPipe

CFX_e24TranPipe allows ColdFusion developers to integrate real-time automated credit card processing capabilities into their applications and websites. It is a simple inline tag that requires a handful of variables. When called the CFX_ e24TranPipe tag initiates a transaction processing pipeline and returns the results of the credit card transaction.

File name: e24TranPipe.dll

Install the CFX_e24TranPipe

Installation Steps

1. Copy the file e24TranPipe.dll to the ColdFusion Bin directory located under the ColdFusion Installation directory. Example: c:\cfusion\bin\
2. Go to your ColdFusion Documentation. This is normally located at: http://www.yourdomain.com/cfdocs/
3. Select your ColdFusion Administrator and select the **Tags** button on your left.
4. Type the following in the form field to add a new CFX tag: **CFX_e24TranPipe**. Then, click the Add button.
5. Verify that the location of the e24TranPipe.dll indicates **Server Library** in the form field. Example: c:\cfusion\bin\e24TranPipe.dll
6. Make sure that the Procedure is **ProcessTagRequest**.

Implement the CFX_e24TranPipe

CFX_e24TranPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “TranPortal Raw Interface” section for more information about implementing the CFX_e24TranPipe tag.

Tag Properties

<CFX_e24TranPipe
alias=”Terminal Alias”

resourcepath=”Path for resource file”
action="A Valid Transaction Action"
card="Credit Card To Debit"
expyear="Card Expiration Year (YYYY)"
expmonth="Card Expiration Month (MM)"
expday="Card Expiration Day (DD)"
CVV2="Card CVV2 Code"
member="Cardholder Name"
addr="Cardholder Address"
zip="Cardholder ZIP Code"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"

type=”Payment Instrument Type”
transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion Application or Website once the CFX_e24TranPipe Tag has completed its execution.

SC_RESULT
SC_AUTH
SC_REF
SC_CARDBALANCE
SC_AVR
SC_DATE

SC_TRANSID
SC_TRACKID
SC_UDF1
SC_UDF2
SC_UDF3
SC_UDF4
SC_UDF5
SC_SUCCESS- A boolean value indicating if the transaction was successful or not. "YES" if successful; "NO" otherwise.
SC_ERROR- Descriptive text of the error if the transaction was not successful.

Retrieving the Results

To retrieve the tag results use a ColdFusion routine similar in respect to the following code:

```
<CFOUTPUT>
<CFIF sc_Success>
Success!<br>
Result is #sc_result#.<br>
Auth is #sc_auth#.<br>
Ref is #sc_ref#.<br>
CardBalance is #sc_cardbalance#.<br>
Avr is #sc_avr#.<br>
Date is #sc_date#.<br>
TransId is #sc_transid#.<br>
TrackId is #sc_trackid#.<br>
<CFELSE>
Failure!<br>
The error is #sc_error#
</CFIF>
</CFOUTPUT>
```

Test the CFX_e24TranPipe Tag

A very simple application page has been provided to test your tag. It indicates how to initiate the tag and sends a formatted transaction result to the browser.

Security Concerns - The CF_e24TranPipe ColdFusion Tag uses Secure Socket Layer Communications to transmit tag properties to the Transaction Processing Network. Therefore, all of your transmitted data is extremely safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

13: ASP e24TranPipe

The e24TranPipe ActiveX/COM Object provides developers with a set of properties and methods to perform real-time credit card transactions securely across the Internet. This object can be used in an Active Server Pages environment. Please note, since ASP projects do not support asynchronous communications and processing, the e24TranPipe object will not return status messages. A simple ASP demonstration is provided below.

ASP e24TranPipe Details

File name: e24TranPipe.dll

```
<%Dim MyObj

Set MyObj = Server.CreateObject("e24TranPipe.e24TranPipe.1")

MyObj.Alias=”Term2000”

MyObj.Action = "1"

MyObj.Card = "4444-3333-2222-1111"

MyObj.ExpYear = "2005"  'YYYY

MyObj.ExpMonth = "12"  'MM

MyObj.ExpDay = “31”  ‘DD

MyObj.CVV2 = "123"

MyObj.Member = "Jeff Doe"

MyObj.Addr = "123 Main Street"

MyObj.Zip = "12345"
```

```
MyObj.Amt = "10.00"

MyObj.Currency = "840"

MyObj.Type = “CC”

MyObj.TrackId = strTrackID 'TODO create a new trackid for each transaction

MyObj.Udf1 = "Your UserDefined Field 1"

MyObj.Udf2 = "Your UserDefined Field 2"

MyObj.Udf3 = "Your UserDefined Field 3"

MyObj.Udf4 = "Your UserDefined Field 4"

MyObj.Udf5 = "Your UserDefined Field 5"

'perform the transaction

Dim TransVal,varResult, varAuth, varAVR, varRef, varTransId, varTrackId,

varDate,varErrorMsg,varRawResponse

TransVal = MyObj.PerformTransaction 'returns 0 for success; -1 for failure

varRawResponse = MyObj.RawResponse

varResult = MyObj.Result

varAuth = MyObj.Auth

varAVR = MyObj.Avr

varRef = MyObj.Ref

varCardBalance = MyObj.CardBalance

varTransId = MyObj.TransId

varDate = MyObj.Date

varErrorMsg = MyObj.ErrorMsg%>
```

```
<%Response.Write("<P>Transval  = " &Transval)

Response.Write("<P>varRawResponse  = " & varRawResponse )

Response.Write("<P>varResult  = " & varResult )

Response.Write("<P>varAuth  = " &  varAuth)

Response.Write("<P>varCardBalance  = " &  varCardBalance)

Response.Write("<P>varAVR  = " & varAVR )

Response.Write("<P>varRef  = " & varRef )

Response.Write("<P>varTransId  = " &  varTransId)

Response.Write("<P>varDate  = " & varDate )

Response.Write("<P>varErrorMsg  = " & varErrorMsg)%>
```

Blank Page

ACI Worldwide Inc.

14: Order API Tran Behavior

This section provides an abstract outline of transaction processing to developers so they may focus their attentions on creating transactional objects or implementing an existing e24PaymentPipe Object Library. These abstract transaction outlines may be implemented somewhat differently, but they provide a foundation for understanding how and what actually happens before, during, and after a transaction.

Refer to the “Order API Raw Transaction Capturing System” section for more information.

Order Initialization Abstract

To perform an Order Initialization, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=1 or 4 (required)
- amt (required)
- currencycode
- langid
- responseURL (required)
- errorURL (required)
- trackid
- udf1
- udf2

- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- PaymentId
- PaymentURL

Order Purchase Abstract

To perform an Order Purchase, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=1 (required)
- amt (required)
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result

- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Order Credit Abstract

To perform an Order Credit, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=2 (required)
- amt (required)
- transid
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Order Purchase Abstract

To perform a Void Order Purchase, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=3 (required)
- amt (required)
- transid (required) - Transid must be the transaction ID of the original purchase.
- trackid
- udf1

- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Order Authorization Abstract

To perform an Order Authorization, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=4 (required)

- amt (required)
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- CardBalance
- AVR - Note: Authorizations are generally performed to evaluate the AVR Response. If the AVR Response is acceptable, then a subsequent Order Capture is performed.
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Order Capture Abstract

To perform an Order Capture, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=5 (required)
- amt (required)
- transid (required) - Transid must be the transaction ID of the original authorization.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Order Credit Abstract

To perform a Void Order Credit, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=6 (required)
- amt (required)
- transid (required) - Transid must be the transaction ID of the original credit.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2

- udf3
- udf4
- udf5

Void Order Capture Abstract

To perform a Void Order Capture, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=7 (required)
- amt (required)
- transid (required) - Transid must be the transaction ID of the original capture.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Auth
- Ref
- AVR
- Date

- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Order Authorization Abstract

To perform a Void Order Authorization, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- paymentid (required)
- action=9 (required)
- amt (required)
- transid (required) - Transid must be the transaction ID of the original authorization.
- trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Transaction Response Variables

- Check response for !ERROR! and examine the error message

- Result
- Auth
- Ref
- AVR
- Date
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf4

ACI Worldwide Inc.

15: Order API Raw Transaction Capturing System

The Order API Raw Transaction Capturing System provides developers with a method of integrating order transaction processing into a variety of software products, websites, server software, and client software across the Internet. The Order API Raw Transaction Capturing System is a set of communication protocols, transaction transmit formats, transaction response formats, and error messages that can be used to develop an interface or object that provides the developer with a set of simple tools for communicating with the Order API Processing Servers. All of the e24PaymentPipe Objects and Interfaces have been developed around this specification and provide a higher level interface.

If building your own interface object, ACI recommends reviewing the following documentation or choose a pre-existing e24PaymentPipe Object to incorporate into your commerce project.

Communication Protocol Specifications

- **Protocol**: http
- **Port**: 443 Verisign 3.0 SSL Certificate
- **Target (action)**:
  - URL Encoded (legacy)
  - https://<hostaddress>/<context>/servlet/PaymentInitHTTPServlet or
  - https://<hostaddress>/<context>/servlet/PaymentTranHTTPServlet
  - XML (new)
  - https://<hostaddress>/<context>/servlet/PaymentInitXMLServlet or
  - https://<hostaddress>/<context>/servlet/PaymentTranXMLServlet
- **Method**: POST
- **Content-Type**

- **URL Encoded**: application/www-form-urlencoded or application/x-www-form-urlencoded
- **XML**: application/xml

- **Transmit Data Format**:
  - Url Encoded (legacy)
  - XML (new)
- **Response Data Format**:
  - **URL Encoded**: Single text string response delimited by colons... string:string:string:
  - **XML**: <response element tag>response data</response element tag>
- **Encryption Level**: SSL Version 3.0

The Order Initialization and Capture environment requires the developer to communicate a handful of variables across the Internet by way of an http post to the following address:

URL Encoded Order Initialization Example

https://<hostaddress>/<context>/servlet/gateway/payment/PaymentInitHTTPServlet
id=TranPortalID&password=password&action=4&langid=USA&currencycode=840&amt=10.00&responseURL=www.merchant.com/response&errorURL=www.merchant.com/error &trackid=unique tracking id&udf1=User Defined Field&udf2=User Defined Field &udf3=User Defined Field&udf4=User Defined Field&udf5=User Defined Field

URL Encoded Order Transaction Example

https://<hostaddress>/<context>/servlet/gateway/payment/PaymentTranHTTPServlet

id=TranPortalID&password=password&action=5&amt=10.00&paymentid=123456789&transid=1122334455 &trackid=unique tracking id&udf1=User Defined Field&udf2=User Defined Field &udf3=User Defined Field&udf4=User Defined Field&udf5=User Defined Field

XML Order Initialization Example

https://<hostaddress>/<context>/servlet/gateway/payment/PaymentInitXMLServlet
<id>2000</id><password>password</password><action>4</action><amt>25.00</amt><currencycode>840</currencycode><langid>USA</langid><responseurl>http://merchant.response.com</responseurl><errorurl>http://merchant.error.com</errorurl><trackid>100000</trackid><udf1>UserData1</udf1><udf2>UserData2</udf2><udf3>UserData3</udf3><udf4>UserData4</udf4><udf5>UserData5</udf5>

XML Order Transaction Example

https://<hostaddress>/<context>/servlet/gateway/payment/PaymentTranXMLServlet

<id>2000</id><password>password</password><action>5</action><amt>10.00</amt><paymentid>1234567890123456</paymentid><transid>1234567890123456</transid><trackid>10000</trackid><udf1>UserData1</udf1><udf2>UserData2</udf2><udf3>UserData3</udf3><udf4>UserData4</udf4><udf5>UserData5</udf5>

Transaction Transmit Variables and Definitions

id - TranPortal Identification Number. The ACI Commerce Gateway system administrator issues the TranPortal ID to identify the merchant and terminal for transaction processing.

password - TranPortal Password. The ACI Commerce Gateway system administrator issues the TranPortal password to authenticate the merchant and terminal. Your data will be encrypted and password securely hidden as long as you are issuing an https post.

action - The following are valid actions, and they must always be in numeric format.

- Purchase - 1
- Credit - 2
- Void Purchase - 3
- Authorization - 4
- Capture - 5
- Void Credit - 6

- Void Capture - 7
- Void Authorization - 9

amt - The amount of the transaction.

currencycode - The currency code of the transaction.

langid - The language to use when presenting consumer pages.

paymentid - Payment ID, used to identify the payment for subsequent transactions.

transid - Transaction ID, used to identify the original transaction for a void and capture transaction.

responseURL - The URL to send the merchant notification request containing the authorization response.

errorURL - The URL to redirect the consumer browser to if an error occurs.

trackid - A unique tracking ID issued by the merchant's commerce system which is stored with the transaction. (Avoid spaces and extended characters, use only alpha-numeric)

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Transaction Response Variables

Every transaction processed through the Transaction Capturing System is returned as a single text string. The return values of the processed transaction are seperated (delimited) by colons. It is the responsibility of the developer to parse this text string into an employable object or type for their software.

Standard Order Initialization Response:
PaymentId:PaymentURL

Standard Order Transaction Response:
Result:Auth:Ref:CardBalance:AVR:Date:TransId:TrackId:UDF1:UDF2:UDF3:UDF4:UDF5

Response Definitions

PaymentId - Unique order ID generated by Commerce Gateway.

PaymentURL - The URL to redirect the consumer browser to enter payment details.

Result - Returned as the transaction response evaluator. Check the Result for error, then evaluate the transaction to determine if it performed successfully.

- CAPTURED - Transaction was captured.
- APPROVED - Transaction was approved.
- VOIDED - Transaction was voided.
- NOT CAPTURED - Transaction was not captured.
- NOT APPROVED - Transaction was not approved.
- NOT VOIDED - Transaction was not voided
- DENIED BY RISK - Risk denied the transaction.
- FAILED AVS - Transaction did not pass Address Verification.
- HOST TIMEOUT - The authorization system did not respond within the timeout limit.

Auth - The resulting authorization number of the transaction.

Ref - The resulting reference number of the transaction. This number or series of letters is used for referential purposes by some acquiring institutions and should be stored properly.

AVR - A single letter providing information about the cardholder's submitted data. The letter indicates how closely the submitted card number, address, and ZIP Code match at the card-issuing bank.

- A - Address matches.
- E - Address match error.
- N - No address match.
- R - AVS not available.

- S - Service not supported.
- U - Address match not capable.
- W - 9 digit ZIP Code matches.
- X - Address and 9 digit ZIP Code matches.
- Y - Address and 5 digit ZIP Code matches.
- Z - 5 digit ZIP Code matches.
- 0 - Address could not be verified.

Date - Transaction Date in the format of the authorization system.

TransId - Unique transaction ID issued by the ACI Commerce Gateway.

TrackId - The Track ID sent by the merchant in the transaction request.

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Error Response Messages - If any errors occur during the transmission of the transaction data, the response format will contain a single string indicating that an error occurred. All response error messages begin with a !ERROR! identifier. The developer should examine the actual response message string and determine if an error has occurred. Refer to the ***ACI Commerce Gateway Response and Error Code Support Guide*** for more information.

Error Response Conditions

Normal Response Condition

- Normal transaction processing
- PerformTransaction returns 0
- Merchant web software site should not need to initiate any special processing.

Commerce Gateway Error Response Condition

- Error transaction processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- The ACI Commerce Gateway encountered an error with either the data validation, database access, or system-related issues.
- Merchant web software site should not need to initiate any special processing.

Example:

XML Response: <error_code_tag>GW00165</error_code_tag><error_service_tag>null</error_service_tag>

<error_text>!ERROR!-GW00165-Invalid Track ID data.</error_text>

<error_code_tag> and <error_text> are documented in the ***ACI Commerce Gateway Response and Error Code Support Guide***.

Plug-in Error Response Condition

- Error DLL processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- e24 dll plugins utilize the Windows HTTP Services (WinHTTP).
- Merchant web software may need to format reversal transaction for:

  ERROR_WINHTTP_TIMEOUT and ERROR_WINHTTP_OPERATION_CANCELLED since it is possible that the request message was transmitted and the response results are unknown.
- e24 maps the most common WinHTTP errors into text.

  ErrorMsg Text Common WinHTTP Errors

  "Connection timed out", // ERROR_WINHTTP_TIMEOUT

  "Internal Error", // ERROR_WINHTTP_INTERNAL_ERROR

  "Invalid URL", // ERROR_WINHTTP_INVALID_URL

  "Name not resolved", // ERROR_WINHTTP_NAME_NOT_RESOLVED

  "Cannot Connect", // ERROR_WINHTTP_CANNOT_CONNECT

"Connection Aborted", // ERROR_WINHTTP_OPERATION_ CANCELLED

- All other WinHTTP errors will result in the following ErrorMsg:

  "Connection Failed with error <WinHTTP Error>"

WinHTTP Errors: http://msdn.microsoft.com/library/default.asp?url=/library/en-es/winhttp/http/winhttpsendrequest.aspder

16: API Visual C/C++ Plug-in

The e24PaymentPipe C++ component is designed for C++ programmers using Visual C++. The component is based on a pure virtual interface to the object (Ie24PaymentPipe), obtained by calling a single factory function exported by the DLL (Createe24PaymentPipe). The component has the ability to deliver asynchronous status messages during the transaction. Any object that wants to receive these notifications must implement the Ie24PaymentPipeStatus interface.

File name: e24PaymentPipe.dll

Location: Plugin Downloads

The e24PaymentPipe C++ Component

class **Ie24PaymentPipeStatus**

Purpose:
This is a pure virtual interface. Any class that wants asynchronous status notifications from the e24PaymentPipe C++ component must derive from this class and implement its methods.

---

virtual void OnStatusUpdate(const char* pMsg);

Purpose:
This is the method that is called on an object that receives asynchronous status messages during a transaction.

Parameters:
pMsg - a character string pointer that contains the message.

---

class **Ie24PaymentPipe**

Purpose:
This is a pure virtual interface. It is implemented by the C++ e24PaymentPipe component. It provides methods for accessing the setup parameters and results, all as properties of the object. An instance of the e24PaymentPipe C++ component is created using the factory method named Createe24PaymentPipe. An instance of this object cannot be created using the new operator. Likewise, the delete operator cannot be used to delete an instance of the object. Instead, call the Release method to delete an instance of the object.

---

void Release();

Purpose:
This method is used to delete the instance of the object. Remember that the delete operator cannot be used.

---

void SetAlias(LPCSTR alias);

Purpose:
Set the value of the ALIAS object property.

Parameters:
alias - A pointer to a character string that contains the new property value.

---

void SetResourcePath(LPCSTR resourcepath);

Purpose:
Set the value of the resource path object property.

Parameters:
resourcepath - A pointer to a character string that contains the new property value.

---

void SetAction(LPCSTR action);

Purpose:
Set the value of the ACTION object property.

Parameters:
action - A pointer to a character string that contains the new property value.

---

void GetAction(char* buf, int bufsize);

Purpose:
Get the value of the ACTION object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetAmt(LPCSTR amt);

Purpose:
Set the value of the AMT object property.

Parameters:
amt - A pointer to a character string that contains the new property value.

---

void GetAmt(char* buf, int bufsize);

Purpose:
Get the value of the AMT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCurrency(LPCSTR currency);

Purpose:
Set the value of the CURRENCY object property.

Parameters:
currency - A pointer to a character string that contains the new property value.

void GetCurrency(char* buf, int bufsize);

Purpose:
Get the value of the CURRENCY object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter

---

void SetLanguage(LPCSTR language);

Purpose:

Set the value of the LANGUAGE object property.

Parameters:

language - A pointer to a character string that contains the new property value.

---

virtual void GetLanguage(char* buf, int bufsize);

Purpose:
Get the value of the LANGUAGE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetResponseURL(LPCSTR responseurl);

Purpose:

Set the value of the RESPONSEURL object property.

Parameters:

responseurl - A pointer to a character string that contains the new property value.

---

virtual void GetResponseURL(char* buf, int bufsize);

Purpose:
Get the value of the RESPONSEURL object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetErrorURL(LPCSTR errorurl);

Purpose:

Set the value of the ERRORURL object property.

Parameters:

errorurl - A pointer to a character string that contains the new property value.

---

virtual void GetErrorURL(char* buf, int bufsize);

Purpose:
Get the value of the ERRORURL object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetPaymentId(LPCSTR payid);

Purpose:
Set the value of the PAYMENTID object property.

Parameters:
payid - A pointer to a character string that contains the new property value.

virtual void GetPaymentId(char* buf, int bufsize);

Purpose:
Get the value of the PAYMENTID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetTransId(LPCSTR transid);

Purpose:
Set the value of the TRANSID object property.

Parameters:
transid - A pointer to a character string that contains the new property value.

---

virtual void GetTransId(char* buf, int bufsize);

Purpose:
Get the value of the TRANSID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetTrackId(LPCSTR trackid);

Purpose:
Set the value of the TRACKID object property.

Parameters:
trackid - A pointer to a character string that contains the new property value.

---

void GetTrackId(char* buf, int bufsize);

Purpose:
Get the value of the TRACKID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUDF1(LPCSTR udf1);

Purpose:
Set the value of the UDF1 object property.

Parameters:
udf1 - A pointer to a character string that contains the new property value.

---

void GetUDF1(char* buf, int bufsize);

Purpose:
Get the value of the UDF1 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf2(LPCSTR udf2);

Purpose:
Set the value of the udf2 object property.

Parameters:
udf2 - A pointer to a character string that contains the new property value.

---

void Getudf2(char* buf, int bufsize);

Purpose:
Get the value of the udf2 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf3(LPCSTR udf3);

Purpose:
Set the value of the udf3 object property.

Parameters:
udf3 - A pointer to a character string that contains the new property value.

---

void Getudf3(char* buf, int bufsize);

Purpose:
Get the value of the udf3 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf4(LPCSTR udf4);

Purpose:
Set the value of the udf4 object property.

Parameters:
udf4 - A pointer to a character string that contains the new property value.

---

void Getudf4(char* buf, int bufsize);

Purpose:
Get the value of the udf4 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf5(LPCSTR udf5);

Purpose:
Set the value of the udf5 object property.

Parameters:
udf5 - A pointer to a character string that contains the new property value.

---

void Getudf5(char* buf, int bufsize);

Purpose:
Get the value of the udf5 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetPaymentPage(char* buf, int bufsize);

Purpose:
Get the value of the PAYMENTPAGE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetResult(char* buf, int bufsize);

Purpose:
Get the value of the RESULT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetAuth(char* buf, int bufsize);

Purpose:
Get the value of the AUTH object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

virtual void GetRef(char* buf, int bufsize);

Purpose:
Get the value of the REF object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetAvr(char* buf, int bufsize);

Purpose:
Get the value of the AVR object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetDate(char* buf, int bufsize);

Purpose:
Get the value of the DATE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetErrorMsg(char* buf, int bufsize);

Purpose:
Get the value of the ErrorMsg object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetRawResponse(char* buf, int bufsize);

Purpose:
Get the value of the RawResponse object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

short PerformInitTransaction(Ie24PaymentPipeStatus* pStatus);

Purpose:
Performs the order initialization.

Parameters:
pStatus - pointer to the Ie24PaymentPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

short PerformTransaction(Ie24PaymentPipeStatus* pStatus);

Purpose:
Performs an order transaction.

Parameters:
pStatus - pointer to the Ie24PaymentPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

function **Createe24PaymentPipe**

Purpose:
This is the factory function used to create an instance of the e24PaymentPipe C++ component. This is the only way to create the component. The new operator cannot be used to create an instance of the component.

Definition:
Ie24PaymentPipe* Createe24PaymentPipe();

---

typedef SCFACTORY

Purpose:
This is the typedef of the function pointer to the Createe24PaymentPipe function. It can be used by applications that load this DLL dynamically using LoadLibrary, and get the function pointer using GetProcAddress.

Definition:
typedef Ie24PaymentPipe* (WINAPI *SCFACTORY)();

17: Order API Java Class Object

The e24PaymentPipe Java Class Object and Source Code can be used in a variety of development environments for a wide range of desktop to web-based applications that are platform independent. The e24PaymentPipe Java Class Object is a royalty-free distributable component source code that any developer can use to enable their Internet e-Commerce applications or websites for Order API transactions. The menu below provides you with documentation on usage and sample applications.

- Class Hierarchy
- Class e24PaymentPipe
- **Class NotEnoughDataException**

Class Hierarchy

- class java.lang.Object
  - class **e24PaymentPipe**

    class java.lang.Throwable (implements java.io.Serializable)

    class java.lang.Exception

    - class **NotEnoughDataException**

Class e24PaymentPipe

java.lang.Object

|

+--**e24PaymentPipe**

---

public final class **e24PaymentPipe**

extends java.lang.Object

This is the class. It is the primary class used for accessing the Portal backend system. The class is marked as final because it should not be subclassed. The class is thread-safe via synchronized methods.

Field Summary

static int	**FAILURE** Indicates failure.
static int	**SUCCESS** Indicates success.

Constructor Summary

e24PaymentPipe()

Method Summary

void	**clearFields**() Clears all of the property values to empty strings.
java.lang.String	**getAction**() Get the value of the action.
java.lang.String	**getAmt**() Get the value of the Amount of the transaction.
java.lang.String	**getAuth**() Get the value of the Authorization number.
java.lang.String	**getAvr**() Get the value of the Address verification response.
java.lang.String	**getCurrencyCode**() Get the value of the currency code.
java.lang.String	**getDate**() Get the value of the Transaction Date.
java.lang.String	**getDebugMsg**() Get the debug message.
java.lang.String	**getErrorMsg**() Get the description of the error if the transaction was not successful.
java.lang.String	**getErrorURL**() Get the value of the error URL.
java.lang.String	**getLanguage**() Get the value of the consumer's browser.
java.lang.String	**getPaymentId**() Get the value of the payment ID.
java.lang.String	**getPaymentPage**() Get the value of the payment page.

java.lang.String	**getRawResponse**() Get the entire contents of the response.
java.lang.String	**getRef**() Get the value of the Reference number.
java.lang.String	**getResponseURL**() Get the value of the response URL.
java.lang.String	**getResult**() Get the value of the Transaction response.
java.lang.String	**getTrackId**() Get the value of the Tracking ID.
java.lang.String	**getTransId**() Get the value of the Transaction ID.
java.lang.String	**getUdf1**() Get the value of the user-defined field 1.
java.lang.String	**getUdf2**() Get the value of the user-defined field 2.
java.lang.String	**getUdf3**() Get the value of the user-defined field 3.
java.lang.String	**getUdf4**() Get the value of the user-defined field 4.
java.lang.String	**getUdf5**() Get the value of the user-defined field 5.
short	**performPaymentInitialization**() Performs the payment initialization transaction based on the property values that have been set.
short	**performTransaction**() Performs the order transaction based on the property values that have been set.
void	**setAction**(java.lang.String action) Set the action.
void	**setAmt**(java.lang.String amt) Set the Amount of the transaction.

void	**setCurrencyCode**(java.lang.String currency) Set the currency code.
void	**setErrorUrl**(java.lang.String cvv2) Set the error URL.
void	**setLanguage**(java.lang.String member) Set the consumer's language.
void	**setPaymentId**(java.lang.String password) Set the payment id.
void	**setPaymentPage**(java.lang.String password) Set the payment page.
void	**setResourcePath**(java.lang.String resourcepath) Set the resource path.
void	**setResponseURL**(java.lang.String ref) Set the response URL.
void	**setSSL**(int ssl) Set the SSL Flag (0 = off; 1 = on).
void	**setTrackId**(java.lang.String trackid) Set the Tracking ID.
void	**setTransId**(java.lang.String transid) Set the Transaction ID.
void	**setUdf1**(java.lang.String udf1) Set the user-defined field 1.
void	**setUdf2**(java.lang.String udf2) Set the user-defined field 2.
void	**setUdf3**(java.lang.String udf3) Set the user-defined field 3.
void	**setUdf4**(java.lang.String udf4) Set the user-defined field 4.
void	**setUdf5**(java.lang.String udf5) Set the user-defined field 5.
void	**setAlias**(java.lang.String alias) Set the URL for the Gateway server.

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Field Detail

SUCCESS

public static final int **SUCCESS**

Indicates success.

FAILURE

public static final int **FAILURE**

Indicates failure.

Constructor Detail

e24PaymentPipe

public **e24PaymentPipe**()

Method Detail

clearFields

public void **clearFields**()

Clears all of the property values to empty strings.

---

getAction

public java.lang.String **getAction**()

Get the value of the action.

Returns:

The value of the action.

---

getAmt

public java.lang.String **getAmt**()

Get the value of the Amount of the transaction.

Returns:

The value of the Amount of the transaction.

---

getAuth

public java.lang.String **getAuth**()

Get the value of the Authorization number.

Returns:

The value of the Authorization number.

---

getAvr

public java.lang.String **getAvr**()

Get the value of the Address verification response.

Returns:

The value of the Address verification response.

---

getCurrencyCode

public java.lang.String **getCurrencyCode**()

Get the value of the currency code.

Returns:

The value of the currency code.

---

getDate

public java.lang.String **getDate**()

Get the value of the Transaction Date.

Returns:

The value of the Transaction Date.

---

getDebugMsg

public java.lang.String **getDebugMsg**()

Get the debug message.

Returns:

The value of the debug message.

---

getErrorMsg

public java.lang.String **getErrorMsg**()

Get the description of the error if the transaction was not successful.

Returns:

The error description if the transaction was not successful.

---

getErrorURL

public java.lang.String **getErrorURL**()

Get the URL to which to redirect the consumer browser if an error occurs.

Returns:

The value of the error URL.

---

getLanguage

public java.lang.String **getLanguage**()

Get the value of the consumer's language.

Returns:

The value of the consumer's langauge.

---

getPaymentId

public java.lang.String **getPaymentId**()

Get the value of the payment ID.

Returns:

The value of the payment ID.

---

getPaymentPage

public java.lang.String **getPaymentPage**()

Get the value of the payment page.

Returns:

The value of the payment page.

---

getRawResponse

public java.lang.String **getRawResponse**()

Get the entire contents of the response.

Returns:

The entire contents of the response.

---

getRef

public java.lang.String **getRef**()

Get the value of the Reference number.

Returns:

The value of the Reference number.

---

getResponseURL

public java.lang.String **getResponseURL**()

Get the value of the response URL.

Returns:

The value of the response URL.

---

getResult

public java.lang.String **getResult**()

Get the value of the Transaction response.

Returns:

The value of the Transaction response.

---

getTrackId

public java.lang.String **getTrackId**()

Get the value of the Tracking ID.

Returns:

The value of the Tracking ID.

---

getTransId

public java.lang.String **getTransId**()

Get the value of the Transaction ID.

Returns:

The value of the Transaction ID.

---

getUdf1

public java.lang.String **getUdf1**()

Get the value of the user-defined field 1.

Returns:

The value of the user-defined field 1.

---

getUdf2

public java.lang.String **getUdf2**()

Get the value of the user-defined field 2.

Returns:

The value of the user-defined field 2.

---

getUdf3

public java.lang.String **getUdf3**()

Get the value of the user-defined field 3.

Returns:

The value of the user-defined field 3.

---

getUdf4

public java.lang.String **getUdf4**()

Get the value of the user-defined field 4.

Returns:

The value of the user-defined field 4.

---

getUdf5

public java.lang.String **getUdf5**()

Get the value of the user-defined field 5.

Returns:

The value of the user-defined field 5.

---

performPaymentInitialization

public short **performPaymentInitialization**()

throws **NotEnoughDataException**

Performs the payment initialization based on the property values that have been set.

Returns:

e24PaymentPipe.SUCCESS if successful; otherwise e24PaymentPipe. FAILURE.

Throws:

NotEnoughDataException - Thrown if no property values have been set.

---

performTransaction

public short **performTransaction**()

throws **NotEnoughDataException**

Performs the order transaction based on the property values that have been set.

Returns:

e24PaymentPipe.SUCCESS if successful; otherwise e24PaymentPipe. FAILURE.

Throws:

NotEnoughDataException - Thrown if no property values have been set.

---

setAction

public void **setAction**(java.lang.String action)

Set the action.

Parameters:

action - The new value of the action.

---

setAmt

public void **setAmt**(java.lang.String amt)

Set the Amount of the transaction.

Parameters:

amt - The new value of the Amount of the transaction.

---

setCurrencyCode

public void **setCurrencyCode**(java.lang.String currency)

Set the currency code.

Parameters:

currency - The new value of the currency code.

---

setErrorURL

public void **setErrorURL**(java.lang.String errorurl)

Set the URL to redirect the consumer browser to if an error occurs.

Parameters:

errorurl - The URL to which to redirect the consumer browser if an error occurs.

---

setLanguage

public void **setLanguage**(java.lang.String language)

Set the consumer's language.

Parameters:

member - The new value of the consumer's language.

---

setPaymentId

public void **setPaymentId**(java.lang.String paymentid)

Set the payment ID.

Parameters:

paymentid - The new value of the payment ID.

---

setPaymentPage

public void **setPaymentPage**(java.lang.String paymentpage)

Set the payment page.

Parameters:

paymentpage - The new value of the payment page.

---

setResourcePath

public void **setResourcePath**(java.lang.String resourcepath)

Set the resourcepath for the Gateway server.

Parameters:

resourcepath - The new value of the resourcepath.

---

setResponseURL

public void **setResponseURL**(java.lang.String responseurl)

Set the response URL.

Parameters:

responseurl - The new value of the response URL.

---

setSSL

public void **setSSL**(int ssl)

Set the SSL Flag (0 = off; 1 = on).

Parameters:

ssl - The new value of the SSL flag.

---

setTrackId

public void **setTrackId**(java.lang.String trackid)

Set the Tracking ID.

Parameters:

trackid - The new value of the Tracking ID.

---

setTransId

public void **setTransId**(java.lang.String transid)

Set the Transaction ID.

Parameters:

transid - The new value of the Transaction ID.

---

setUdf1

public void **setUdf1**(java.lang.String udf1)

Set the user-defined field 1.

Parameters:

udf1 - The new value of the user-defined field 1.

---

setUdf2

public void **setUdf2**(java.lang.String udf2)

Set the user-defined field 2.

Parameters:

udf2 - The new value of the user-defined field 2.

---

setUdf3

public void **setUdf3**(java.lang.String udf3)

Set the user-defined field 3.

Parameters:

udf3 - The new value of the user-defined field 3.

---

setUdf4

public void **setUdf4**(java.lang.String udf4)

Set the user-defined field 4.

Parameters:

udf4 - The new value of the user-defined field 4.

---

setUdf5

public void **setUdf5**(java.lang.String udf5)

Set the user-defined field 5.

Parameters:

udf5 - The new value of the user-defined field 5.

---

setAlias

public void **setAlias**(java.lang.String alias)

Set the ALIAS for the Gateway server.

Parameters:

alias - The new value of the ALIAS.

Class NotEnoughDataException

java.lang.Object

|

+--java.lang.Throwable

```
|
+--java.lang.Exception
    |
    +--NotEnoughDataException
```

All Implemented Interfaces:

- java.io.Serializable

---

public class **NotEnoughDataException**

extends java.lang.Exception

This exception is thrown if ProcessTransaction method of the SecureCharge class is invoked without setting the values of any properties.

See Also:

- Serialized Form

Constructor Summary

NotEnoughDataException()
The empty constructor

NotEnoughDataException(java.lang.String msg)
The constructor with info.

Methods inherited from class java.lang.Throwable

- fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Constructor Detail

NotEnoughDataException

public **NotEnoughDataException**()

The empty constructor

NotEnoughDataException

public **NotEnoughDataException**(java.lang.String msg)

The constructor with info.

Parameters:

msg - The text message that describes the exception.

18: CF_e24PaymentPipe

CF_e24PaymentPipe allows ColdFusion developers to integrate real-time order processing capabilities into their applications and websites. It is an inline tag that requires a handful of variables. When called, the CF_e24PaymentPipe tag initiates a transaction processing pipeline and returns the results of the order transaction.

File name: e24PaymentPipe.cfm

Install the CF_e24PaymentPipe

Copy the e24PaymentPipe File

Copy the file e24PaymentPipe.cfm to the CustomTags directory located under the ColdFusion Installation directory. Example: **c:\cfusion\CustomTags**

Implement the CF_e24PaymentPipe

CF_e24PaymentPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “Order API Raw Transaction Capturing System” section for more information.

Tag Properties

<CF_e24PaymentPipe
id="TranPortal ID"
password="TranPortal Password"
action="A Valid Transaction Action"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"
language="Consumer language" responseurl="URL to send notification message"
errorurl="URL to use if there is an error" transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion Application or Website once the CF_e24PaymentPipe Tag has completed its execution. When the CF_e24PaymentPipe Tag has finished execution, the variable **Success** will return a value of 0 (Failure) or 1 (Success). If you receive a **Success** value of 0, you should evaluate the **Error** result.

The results are dependent upon:

SUCCESS (0 or 1)
ERROR
PAYMENTID
PAYMENTPAGE
RESULT
AUTH
REF
AVR
DATE
TRANSID
TRACKID
UDF1
UDF2
UDF3
UDF4
UDF5

Security Concerns - The CF_e24PaymentPipe Cold Fusion tag uses Secure Socket Layer communications to transmit tag properties to the Transaction Processing Network. Therefore, the transmitted data is safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

ACI Worldwide Inc.

19: CFX_e24PaymentPipe

CFX_e24PaymentPipe allows ColdFusion developers to integrate real-time order processing capabilities into their applications and websites. It is a simple inline tag that requires a handful of variables. When called the CFX_e24PaymentPipe tag initiates a transaction processing pipeline and returns the results of the order transaction.

File name: e24TranPipe.dll

Install the CFX_e24PaymentPipe

Installation Procedures

1. Copy the file e24PaymentPipe.dll to the ColdFusion Bin directory located under the ColdFusion Installation directory. Example: c:\cfusion\bin\
2. Go to your ColdFusion Documentation. This is normally located at: http://www.yourdomain.com/cfdocs/
3. Select your ColdFusion Administrator and select the **Tags** button on your left.
4. In the form field to add a new CFX tag type the following: **CFX_ e24PaymentPipe** and click the **Add** button once.
5. Make sure the location of the e24PaymentPipe.dll is indicated **Server Library** form field. Example: **c:\cfusion\bin\e24PaymentPipe.dll**
6. Make sure that the Procedure is **ProcessTagRequest**.

Implement the CFX_e24PaymentPipe

CFX_e24TranPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “Order API Raw Transaction Capturing System” section for more information.

Tag Properties

<CFX_e24PaymentPipe
resourcepath="Resource Path."
alias="Terminal Alias."
action="A Valid Transaction Action"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"
language="Consumer language" responseurl="URL to send notification message"
errorurl="URL to use if there is an error" transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion Application or Website once the CFX_e24PaymentPipe Tag has completed its execution.

SC_PAYMENTID
SC_PAYMENTPAGE
SC_RESULT
SC_AUTH
SC_REF
SC_AVR
SC_DATE
SC_TRANSID
SC_TRACKID
SC_UDF1
SC_UDF2
SC_UDF3
SC_UDF4
SC_UDF5

SC_SUCCESS- A boolean value indicating if the transaction was successful or not. "YES" if successful; "NO" otherwise.
SC_ERROR- Descriptive text of the error if the transaction was not successful.

Retrieving the Results

To retrieve the tag results use a ColdFusion routine similar in respect to the following code:

```
<CFOUTPUT>
<CFIF sc_Success>
Success!<br>
Result is #sc_result#.<br>
Auth is #sc_auth#.<br>
Ref is #sc_ref#.<br>
Avr is #sc_avr#.<br>
Date is #sc_date#.<br>
TransId is #sc_transid#.<br>
TrackId is #sc_trackid#.<br>
<CFELSE>
Failure!<br>
The error is #sc_error#
</CFIF>
</CFOUTPUT>
```

Security Concerns - The CF_e24PaymentPipe ColdFusion Tag uses Secure Socket Layer Communications to transmit tag properties to the Transaction Processing Network. Therefore, transmitted data is safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

ACI Worldwide Inc.

20: ASP e24PaymentPipe

The e24PaymentPipe ActiveX/COM Object provides developers with a set of properties and methods to perform real-time order transactions securely across the Internet. This object can be used in an Active Server Pages environment. Please note, since ASP projects do not support asynchronous communications and processing the e24PaymentPipe object will not return status messages. An ASP demonstration is provided below.

File name: e24PaymentPipe.dll

e24PaymentPipe Components

```
<%Dim MyObj

Set MyObj = Server.CreateObject("e24PaymentPipe.e24PaymentPipe.1")

MyObj.ResourcePath = "c:\resource path\"

MyObj.Alias = "Term2000"

MyObj.Action = "1"

MyObj.Amt = "10.00"

MyObj.Currency = "840"

MyObj.Language = "USA"

MyObj.ResponseURL = "www.merchant.com/response"

MyObj.ErrorURL = "www.merchant.com/error"

MyObj.TrackId = strTrackID 'TODO create a new trackid for each transaction

MyObj.Udf1 = "Your UserDefined Field 1"
```

```
MyObj.Udf2 = "Your UserDefined Field 2"

MyObj.Udf3 = "Your UserDefined Field 3"

MyObj.Udf4 = "Your UserDefined Field 4"

MyObj.Udf5 = "Your UserDefined Field 5"

'perform the transaction

Dim TransVal,varPaymentId, varPaymentPage,varErrorMsg,varRawResponse

TransVal = MyObj.PerformPaymentInitialization 'returns 0 for success; -1 for
failure

varRawResponse = MyObj.RawResponse

varPaymentId = MyObj.PaymentId

varPaymentPage = MyObj.PaymentPage

varErrorMsg = MyObj.ErrorMsg%>

<%Response.Write("<P>Transval  = " &Transval)

Response.Write("<P>varPaymentId  = " & varPaymentId )

Response.Write("<P>varPaymentPage  = " & varPaymentPage )

Response.Write("<P>varErrorMsg  = " & varErrorMsg)%>
```

21: Card Management Transaction Behavior

This section provides an abstract outline of transaction processing to developers so they may focus their attentions on creating transactional objects or implementing an existing e24CardPipe Object Library. These abstract transaction outlines may be implemented somewhat differently, but they provide a foundation for understanding how and what actually happens before, during, and after a transaction.

Refer to the “TranPortal Raw Interface” section for more information about the Card Management Transaction Behavior.

Replenishment Abstract

To perform a Replenishment, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=12 (required)
- card (required)
- type=SV(optional)
- expyear (optional)
- expmonth (optional)
- expday (optional)
- cvv2 (optional)
- amt (required)

- currencycode (optional)
- trackid (optional)
- udf1 (optional)
- udf2 (optional)
- udf3 (optional)
- udf4 (optional)
- udf5 (optional)

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Ref
- CardBalance
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Void Replenishment Abstract

To perform a Void Replenishment, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=12 (required)

- card (required)
- type=SV(optional)
- expyear (optional)
- expmonth (optional)
- expday (optional)
- cvv2 (optional)
- amt (required)
- currencycode (optional)
- transid (required) - Transid must be the transaction ID of the original purchase.
- trackid (optional)
- udf1 (optional)
- udf2 (optional)
- udf3 (optional)
- udf4 (optional)
- udf5 (optional)

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Ref
- CardBalance
- Transid
- Trackid
- udf1
- udf2
- udf3
- udf4
- udf5

Balance Inquiry Abstract

To perform a Balance Inquiry, supply the data listed under Transaction Transmit Variables and review the data listed under Transaction Response Variables. Fields that are required are marked as such. Other fields are optional.

Transaction Transmit Variables

- id (required)
- password (required)
- action=10 (required)
- type=SV(optional)
- card (required)
- expyear (optional)
- expmonth (optional)
- expday (optional)
- cvv2 (optional)
- trackid (optional)
- udf1 (optional)
- udf2 (optional)
- udf3 (optional)
- udf4 (optional)
- udf5 (optional)

Transaction Response Variables

- Check response for !ERROR! and examine the error message
- Result
- Ref
- CardBalance
- Transid
- Trackid
- udf1

- udf2
- udf3
- udf4
- udf5

ACI Worldwide Inc.

22: Card Management Raw Interface

Communication Protocol Specifications

- **Protocol**: http
- **Port**: 443 Verisign 3.0 SSL Certificate
- **Target**
  - URL Encoded (legacy): https://localhost/context/servlet/CardManagementHTTPServlet
  - XML (new format supported by plugins): https://localhost/context/servlet/CardManagementXMLServlet
- **Method**: POST
- **Content-Type**
  - URL Encoded (legacy): application/www-form-urlencoded or application/x-www-form-urlencoded
  - XML (new): application/xml
- **Transmit Data Format**
  - Url Encoded (legacy)
  - XML (new).
- **Response Data Format**
  - **URL Encoded (legacy)**: Single text string response delimited by colons. ..string:string:string:
  - **XML (new)**: <response element tag>response data</response element tag>
- **Encryption Level**: SSL Version 3.0

The Transaction Capturing System requires the developer to communicate a handful of variables across the Internet by way of an http post to the following address:

- URL Encoded Replenishment Example:

https://localhost/conext/servlet/CardManagementHTTPServlet

id=TranPortalID&password=password&action=11&type=SV&card=4444333322221111&expYear=2005&expMonth=12&CVV2=123&amt=10.00&trackid=unique tracking id&udf1=User Defined Field &udf2=User Defined Field&udf3=User Defined Field&udf4=User Defined Field&udf5=User Defined Field

XML Replenishment Example:

https://localhost/conext/servlet/CardManagementHTTPServlet

<id>2000</id><password>password</password><action>1</action><type>CC</type><card>4444333322221111</card><expYear>2005</expYear><expMonth>12</expMonth><expDay>31</expDay><amt>65.00</amt><currencycode>840</currencycode><trackid>100000</trackid><cvv2>416</cvv2><udf1>AA</udf1><udf2>BB</udf2><udf3>CC</udf3><udf4>DD</udf4><udf5>EE</udf5><transid>432233322222222222</transid>

Transaction Transmit Variables and Definitions

id - TranPortal Identification Number. The ACI Commerce Gateway system administrator issues the TranPortal ID to identify the merchant and terminal for transaction processing.

password - TranPortal password. The ACI Commerce Gateway system administrator issues the TranPortal password to authenticate the merchant and terminal. Your data will be encrypted and password securely hidden as long as you are issuing an https post.

action - The following are valid actions, and they must always be in numeric format.

- Balance Inquiry - 10
- Replenishment - 11
- Void Replenishment - 12

type – Payment Instrument Type

- Stored Value – SV
- Credit Card – CC (currently not supported by CardManagement Servlets)

card - Credit card number.

expyear - Expiration date year (must be in number format YYYY).

expmonth - Expiration date month (must be in number format MM).

expday - Expiration date day (must be in number format DD).

cvv2 - CVV2 code (must be in number format).

amt - The amount of the transaction.

currencycode - The currency code of the transaction.

transid - Transaction ID used to identify the original transaction for a void transaction.

trackid - A unique tracking ID issued by the merchant's commerce system which is stored with the transaction. (Avoid spaces and extended characters; use only alphanumeric format.)

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Transaction Response Variables

Every transaction processed through the Transaction Capturing System is returned as a single text string. The return values of the processed transaction separated (delimited) by colons. The developer is responsible for parsing this text string into an employable object or type for their software.

Example:

Result:Ref:CardBalance:TransId:TrackId:UDF1:UDF2:UDF3:UDF4:UDF5

Response Definitions

Result - Returned as the transaction response evaluator. Check the Result for error, then evaluate the transaction to determine if it performed successfully.

- PROCESSED - Transaction was processed.
- VOIDED - Transaction was voided.
- NOT PROCESSED - Transaction was not processed.
- NOT VOIDED - Transaction was not voided.
- HOST TIMEOUT - The authorization system did not respond within the timeout limit.

Ref - The resulting reference number of the transaction. This number or series of letters is used for referential purposes by some acquiring institutions and should be stored properly.

CardBalance - The amount still available associated with this card.

TransId - Unique transaction ID issued by Commerce Gateway.

TrackId - The Track ID sent by the merchant in the transaction request.

udf1-udf5 - The user defines these fields. The field data is passed along with a transaction request and then returned in the transaction response.

If present, the user-defined field data will be archived in the TRANLOG table with the transaction. This allows the Tranlog Search and Reporting feature to include the data as part of the search criteria.

Also, the field can transmit additional data to an external host.

Error Response Messages - If any errors occur during the transmission of the transaction data, the response format will contain a single string indicating that an error occurred. All response error messages begin with a !ERROR! identifier. The developer should examine the actual response message string and determine if an error has occurred. Refer to the ***ACI Commerce Gateway Response and Error Code Support Guide*** for more information.

Error Response Conditions

Normal Response Condition

- Normal transaction processing
- PerformTransaction returns 0
- Merchant web software site should not need to initiate any special processing.

Example:

XML Response: <result>PROCESSED</result><ref>504487010197</ref><cardbalance>$123.00</cardbalance><tranid>4874570301450440</tranid><trackid>100000</trackid><udf1>AA</udf1><udf2>BB</udf2><udf3>CC</udf3><udf4>DD</udf4><udf5>EE</udf5>

Commerce Gateway Error Response Condition

- Error transaction processing
- PerformTransaction returns -1
- GetErrorMsg should return text
- The ACI Commerce Gateway encountered an error with either the data validation, database access, or system-related issues.
- Merchant web software site should not need to initiate any special processing.

Example:

XML Response: <error_code_tag>GW00165</error_code_tag><error_service_tag>null</error_service_tag>

<error_text>!ERROR!-GW00165-Invalid Track ID data.</error_text>

<error_code_tag> and <error_text> are documented in the ***ACI Commerce Gateway Response and Error Code Support Guide***.

Plug-in Error Response Condition

- Error DLL processing
- PerformTransaction returns -1
- GetErrorMsg should return text

- e24 dll plugins utilize the Windows HTTP Services (WinHTTP).
- Merchant web software may need to format reversal transaction for:

  ERROR_WINHTTP_TIMEOUT and ERROR_WINHTTP_OPERATION_CANCELLED since it is possible that the request message was transmitted and the response results are unknown.
- e24 maps the most common WinHTTP errors into text.

  ErrorMsg Text // Common WinHTTP Errors

  "Connection timed out", // ERROR_WINHTTP_TIMEOUT

  "Internal Error", // ERROR_WINHTTP_INTERNAL_ERROR

  "Invalid URL", // ERROR_WINHTTP_INVALID_URL

  "Name not resolved", // ERROR_WINHTTP_NAME_NOT_RESOLVED

  "Cannot Connect", // ERROR_WINHTTP_CANNOT_CONNECT

  "Connection Aborted", // ERROR_WINHTTP_OPERATION_CANCELLED
- All other WinHTTP errors will result in the following ErrorMsg:

  "Connection Failed with error <WinHTTP Error>"

  WinHTTP Errors: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winhttp/http/winhttpsendrequest.asp

23: Card Management Visual C/C++ Plug-in

The Card Management C++ component is designed for C++ programmers using Visual C++. The component is based on a pure virtual interface to the object (Ie24CardPipe), obtained by calling a single factory function exported by the DLL (Createe24CardPipe). The component has the ability to deliver asynchronous status messages during the transaction. Any object that wants to receive these notifications must implement the Ie24CardPipe Status interface.

The interface described for all of the get methods is using the IQuery Interface. If a different interface is needed, reference the associated .tld, .h, and idl files which are included in the Plugin Download that can be obtained from the ACI Commerce Gateway Merchant Website under the Developers menu.

File name: e24CardPipe.dll

Location: Plugin Downloads

e24CardPipe C++ Component

class **Ie24CardPipeStatus**

Purpose:
This is a pure virtual interface. Any class that wants asynchronous status notifications from the e24CardPipe C++ component must derive from this class and implement its methods.

---

virtual void OnStatusUpdate(const char* pMsg);

Purpose:
This is the method that is called on an object that receives asynchronous status messages during a transaction.

Parameters:
pMsg - a character string pointer that contains the message.

---

class **Ie24CardPipe**

Purpose:
This is a pure virtual interface. It is implemented by the C++ e24CardPipe component. It provides methods for accessing the setup parameters and results, all as properties of the object. An instance of the e24CardPipe C++ component is created using the factory method named Createe24CardPipe. An instance of this object cannot be created using the new operator. Likewise, the delete operator cannot be used to delete an instance of the object. Instead, call the Release method to delete an instance of the object.

---

void Release();

Purpose:
This method is used to delete the instance of the object. Remember that the delete operator cannot be used.

---

void SetResourcePath(LPCSTR resourcepath);

Purpose:
Set the value of the resource path object property.

Parameters:
resourcepath - A pointer to a character string that contains the new property value.

---

void SetAlias(LPCSTR alias);

Purpose:
Set the value of the ALIAS object property.

Parameters:
alias - A pointer to a character string that contains the new property value.

---

void SetAction(LPCSTR action);

Purpose:
Set the value of the ACTION object property.

Parameters:
action - A pointer to a character string that contains the new property value.

---

void GetAction(char* buf, int bufsize);

Purpose:
Get the value of the ACTION object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetType(LPCSTR type);

Purpose:
Set the value of the TYPE object property.

Parameters:
type - A pointer to a character string that contains the new property value.

---

void GetType(char* buf, int bufsize);

Purpose:
Get the value of the TYPE object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCard(LPCSTR card);

Purpose:
Set the value of the CARD object property.

Parameters:
card - A pointer to a character string that contains the new property value.

---

void GetCard(char* buf, int bufsize);

Purpose:
Get the value of the CARD object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpYear(LPCSTR expYear);

Purpose:
Set the value of the EXPYEAR object property.

Parameters:
expYear - A pointer to a character string that contains the new property value.

---

void GetExpYear(char* buf, int bufsize);

Purpose:
Get the value of the EXPYEAR object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpMonth(LPCSTR expMonth);

Purpose:
Set the value of the EXPMONTH object property.

Parameters:
expMonth - A pointer to a character string that contains the new property value.

---

void GetExpMonth(char* buf, int bufsize);

Purpose:
Get the value of the EXPMONTH object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetExpDay(LPCSTR expDay);

Purpose:
Set the value of the EXPDAY object property.

Parameters:
expDay - A pointer to a character string that contains the new property value.

---

void GetExpDay(char* buf, int bufsize);

Purpose:
Get the value of the EXPDAY object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCVV2(LPCSTR cvv2);

Purpose:
Set the value of the CVV2 object property.

Parameters:
cvv2 - A pointer to a character string that contains the new property value.

---

void GetCVV2(char* buf, int bufsize);

Purpose:
Get the value of the CVV2 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetAmt(LPCSTR amt);

Purpose:
Set the value of the AMT object property.

Parameters:
amt - A pointer to a character string that contains the new property value.

---

void GetAmt(char* buf, int bufsize);

Purpose:
Get the value of the AMT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetCurrency(LPCSTR currency);

Purpose:
Set the value of the CURRENCY object property.

Parameters:
currency - A pointer to a character string that contains the new property value.

void GetCurrency(char* buf, int bufsize);

Purpose:
Get the value of the CURRENCY object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetTransId(LPCSTR transid);

Purpose:

Set the value of the TRANSID object property.

Parameters:

transid - A pointer to a character string that contains the new property value.

---

virtual void GetTransId(char* buf, int bufsize);

Purpose:
Get the value of the TRANSID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetTrackId(LPCSTR trackid);

Purpose:
Set the value of the TRACKID object property.

Parameters:
trackid - A pointer to a character string that contains the new property value.

void GetTrackId(char* buf, int bufsize);

Purpose:
Get the value of the TRACKID object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void SetUDF1(LPCSTR udf1);

Purpose:
Set the value of the UDF1 object property.

Parameters:
udf1 - A pointer to a character string that contains the new property value.

---

void GetUDF1(char* buf, int bufsize);

Purpose:
Get the value of the UDF1 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf2(LPCSTR udf2);

Purpose:
Set the value of the udf2 object property.

Parameters:
udf2 - A pointer to a character string that contains the new property value.

---

void Getudf2(char* buf, int bufsize);

Purpose:
Get the value of the udf2 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf3(LPCSTR udf3);

Purpose:
Set the value of the udf3 object property.

Parameters:
udf3 - A pointer to a character string that contains the new property value.

---

void Getudf3(char* buf, int bufsize);

Purpose:
Get the value of the udf3 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf4(LPCSTR udf4);

Purpose:
Set the value of the udf4 object property.

Parameters:
udf4 - A pointer to a character string that contains the new property value.

---

void Getudf4(char* buf, int bufsize);

Purpose:
Get the value of the udf4 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void Setudf5(LPCSTR udf5);

Purpose:
Set the value of the udf5 object property.

Parameters:
udf5 - A pointer to a character string that contains the new property value.

---

void Getudf5(char* buf, int bufsize);

Purpose:
Get the value of the udf5 object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetResult(char* buf, int bufsize);

Purpose:
Get the value of the RESULT object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

virtual void GetRef(char* buf, int bufsize);

Purpose:
Get the value of the REF object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

virtual void GetCardBalance(char* buf, int bufsize);

Purpose:
Get the value of the CardBalance object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetErrorMsg(char* buf, int bufsize);

Purpose:
Get the value of the ErrorMsg object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

---

void GetRawResponse(char* buf, int bufsize);

Purpose:
Get the value of the RawResponse object property.

Parameters:
buf - A pointer to the character array that will receive the property value.
bufsize - The length of the buf parameter.

short PerformTransaction(Ie24CardPipeStatus* pStatus);

Purpose:
Performs the transaction.

Parameters:
pStatus - pointer to the Ie24CardPipeStatus object that will receive asynchronous status messages. If NULL is used, then no asynchronous status messages will be delivered.

Returns:
0 if the transaction was successful; -1 otherwise.

---

function Createe24CardPipe

Purpose:
This is the factory function used to create an instance of the e24CardPipe C++ component. This is the only way to create the component. The new operator cannot be used to create an instance of the component.

Definition:
Ie24CardPipe* Createe24CardPipe();

---

typedef SCFACTORY

Purpose:
This is the typedef of the function pointer to the Createe24CardPipe function. It can be used by applications that load this DLL dynamically using LoadLibrary, and get the function pointer using GetProcAddress.

Definition:
typedef Ie24CardPipe* (WINAPI *SCFACTORY)();

24: Card Management Java Class Object

The e24CardPipe Java Class Object and Source Code can be used in a variety of development environments for a wide range of desktop to web-based applications that are platform independent. The e24CardPipe Java Class Object is a royalty-free distributable component source code that any developer can use to enable their Internet e-Commerce applications or websites for Card Management transactions. The menu below provides you with documentation on usage and sample applications.

- **Class Hierarchy**
- **Class e24CardPipe**
- **Class NotEnoughDataException**

Class Hierarchy

- class java.lang.Object
  - class **e24CardPipe**
  - class java.lang.Throwable (implements java.io.Serializable)

    class java.lang.Exception

    class NotEnoughDataException

Class e24CardPipe

java.lang.Object

|

+--e24CardPipe

---

public final class **e24CardPipe**

extends java.lang.Object

This is the class. It is the primary class used for accessing the Card Management backend system. The class is marked as final because it should not be subclassed. The class is thread-safe via synchronized methods.

Field Summary

static int	**FAILURE** Indicates failure.
static int	**SUCCESS** Indicates success.

Constructor Summary

- e24CardPipe()

Method Summary

void	**clearAllFields**() Clears all of the property values to empty strings.
void	**clearResults**() Clears all of the result property values to empty strings.
java.lang.String	**getAction**() Get the value of the Secure-Bank action.

java.lang.String	**getType**() Get the value of the Secure-Bank type.
java.lang.String	**getAmt**() Get the value of the Amount of the transaction.
java.lang.String	**getCard**() Get the value of the Credit card number.
java.lang.String	**getCurrencyCode**() Get the value of the currency code.
java.lang.String	**getCvv2**() Get the CVV2 code.
java.lang.String	**getDebugMsg**() Get the debug message.
java.lang.String	**getErrorMsg**() Get the description of the error if the transaction was not successful.
java.lang.String	**getExpDay**() Get the value of the Credit card expiration day.
java.lang.String	**getExpMonth**() Get the value of the Credit card expiration month.
java.lang.String	**getExpYear**() Get the value of the Credit card expiration year.
java.lang.String	**getRef**() Get the value of the Reference number.
java.lang.String	**getCardBalance**() Get the value of the Card Balance amount.
java.lang.String	**getResult**() Get the value of the Transaction response.
java.lang.String	**getTrackId**() Get the value of the Tracking ID.
java.lang.String	**getTransId**() Get the value of the Transaction ID.

java.lang.String	**getUdf1**() Get the value of the user-defined field 1.
java.lang.String	**getUdf2**() Get the value of the user-defined field 2.
java.lang.String	**getUdf3**() Get the value of the user-defined field 3.
java.lang.String	**getUdf4**() Get the value of the user-defined field 4.
java.lang.String	**getUdf5**() Get the value of the user-defined field 5.
short	**performTransaction**() Performs the transaction based on the property values that have been set.
void	**setAction**(java.lang.String action) Set the Secure-Bank action.
void	**setType**(java.lang.String type) Set the Secure-Bank type.
void	**setAmt**(java.lang.String amt) Set the Amount of the transaction.
void	**setCard**(java.lang.String card) Set the Credit card number.
void	**setCurrencyCode**(java.lang.String currency) Set the currency code.
void	**setCvv2**(java.lang.String cvv2) Set the CVV2 code.
void	**setExpDay**(java.lang.String expday) Set the Credit card expiration day.
void	**setExpMonth**(java.lang.String expmonth) Set the Credit card expiration month.
void	**setExpYear**(java.lang.String expyear) Set the Credit card expiration year.

Void	**setResourcePath**(java.lang.String resourcepath) Set the resource path.
void	**setAlias**(java.lang.String alias) Set the terminal alias.

void	**setRef**(java.lang.String ref) Set the Reference number.
void	**setCardBalance**(java.lang.String cardbalance) Set the Card Balance amount.
void	**setSSL**(int ssl) Set the SSL Flag (0 = off; 1 = on).
void	**setTrackId**(java.lang.String trackid) Set the Tracking ID.
void	**setTransId**(java.lang.String transid) Set the Transaction ID.
void	**setUdf1**(java.lang.String udf1) Set the user-defined field 1.
void	**setUdf2**(java.lang.String udf2) Set the user-defined field 2.
void	**setUdf3**(java.lang.String udf3) Set the user-defined field 3.
void	**setUdf4**(java.lang.String udf4) Set the user-defined field 4.

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

Field Detail

SUCCESS

public static final int **SUCCESS**

Indicates success.

FAILURE

public static final int **FAILURE**

Indicates failure.

Constructor Detail

e24CardPipe

public **e24CardPipe**()

Method Detail

clearAllFields

public void clearAllFields()

Clears all of the property values to empty strings.

---

clearResults

public void **clearResults**()

Clears all of the result property values to empty strings.

---

getAction

public java.lang.String **getAction**()

Get the value of the action.

Returns:

The value of the action.

---

getType

public java.lang.String **getType**()

Get the value of the type.

Returns:

The value of the type

---

getAmt

public java.lang.String **getAmt**()

Get the value of the Amount of the transaction.

Returns:

The value of the Amount of the transaction.

---

getCard

public java.lang.String **getCard**()

Get the value of the Credit card number.

Returns:

The value of the Credit card number.

---

getCurrencyCode

public java.lang.String **getCurrencyCode**()

Get the value of the currency code.

Returns:

The value of the currency code.

---

getCvv2

public java.lang.String **getCvv2**()

Get the CVV2 code.

Returns:

The value of the CVV2 code.

---

getDebugMsg

public java.lang.String **getDebugMsg**()

Get the debug message.

Returns:

The value of the debug message.

---

getErrorMsg

public java.lang.String **getErrorMsg**()

Get the description of the error if the transaction was not successful.

Returns:

The description of the error if the transaction was not successful.

---

getExpDay

public java.lang.String **getExpDay**()

Get the value of the Credit card expiration day.

Returns:

The value of the Credit card expiration day.

---

getExpMonth

public java.lang.String **getExpMonth**()

Get the value of the Credit card expiration month.

Returns:

The value of the Credit card expiration month.

---

getExpYear

public java.lang.String **getExpYear**()

Get the value of the Credit card expiration year.

Returns:

The value of the credit card expiration year.

---

getRef

public java.lang.String **getRef**()

Get the value of the reference number.

Returns:

The value of the reference number.

---

getCardBalance

public java.lang.String **getCardBalance**()

Get the value of the card balance amount.

Returns:

The value of the card balance amount.

---

getResult

public java.lang.String **getResult**()

Get the value of the transaction response.

Returns:

The value of the transaction response.

---

getTrackId

public java.lang.String **getTrackId**()

Get the value of the tracking ID.

Returns:

The value of the tracking ID.

---

getTransId

public java.lang.String **getTransId**()

Get the value of the transaction ID.

Returns:

The value of the transaction ID.

---

getUdf1

public java.lang.String **getUdf1**()

Get the value of the user-defined field 1.

Returns:

The value of the user-defined field 1.

---

getUdf2

public java.lang.String **getUdf2**()

Get the value of the user-defined field 2.

Returns:

The value of the user-defined field 2.

---

getUdf3

public java.lang.String **getUdf3**()

Get the value of the user-defined field 3.

Returns:

The value of the user-defined field 3.

---

getUdf4

public java.lang.String **getUdf4**()

Get the value of the user-defined field 4.

Returns:

The value of the user-defined field 4.

---

getUdf5

public java.lang.String **getUdf5**()

Get the value of the user-defined field 5.

Returns:

The value of the user-defined field 5.

---

performTransaction

public short **performTransaction**()

throws NotEnoughDataException

Performs the transaction based on the property values that have been set.

Returns:

e24CardPipe.SUCCESS if successful; otherwise e24CardPipe.FAILURE.

Throws:

NotEnoughDataException - Thrown if no property values have been set.

---

setAction

public void **setAction**(java.lang.String action)

Set the action.

Parameters:

action - The new value of the action.

setType

public void **setType**(java.lang.String type)

Set the type.

Parameters:

action - The new value of the type.

---

setAmt

public void **setAmt**(java.lang.String amt)

Set the amount of the transaction.

Parameters:

amt - The new value of the amount of the transaction.

---

setCard

public void **setCard**(java.lang.String card)

Set the credit card number.

Parameters:

card - The new value of the credit card number.

---

setCurrencyCode

public void **setCurrencyCode**(java.lang.String currency)

Set the currency code.

Parameters:

currency - The new value of the currency code.

---

setCvv2

public void **setCvv2**(java.lang.String cvv2)

Set the CVV2 code.

Parameters:

cvv2 - The new value of the CVV2.

---

setExpDay

public void **setExpDay**(java.lang.String expday)

Set the credit card expiration day.

Parameters:

expday - The new value of the credit card expiration day.

---

setExpMonth

public void **setExpMonth**(java.lang.String expmonth)

Set the credit card expiration month.

Parameters:

expmonth - The new value of the credit card expiration month.

---

setExpYear

public void **setExpYear**(java.lang.String expyear)

Set the credit card expiration year.

Parameters:

expyear - The new value of the Credit card expiration year.

---

setAlias

public void **setAlias**(java.lang.String alias)

Set the terminal alias.

Parameters:

alias - The new value of the terminal alias.

---

setRef

public void **setRef**(java.lang.String ref)

Set the Reference number.

Parameters:

ref - The new value of the Reference number.

---

setCardBalance

public void **setCardBalance**(java.lang.String cardbalance)

Set the Card Balance amount.

Parameters:

ref - The new value of the Card Balance amount.

---

setSSL

public void **setSSL**(int ssl)

Set the SSL Flag (0 = off; 1 = on).

Parameters:

ssl - The new value of the SSL flag.

---

setTrackId

public void **setTrackId**(java.lang.String trackid)

Set the Tracking Id.

Parameters:

trackid - The new value of the Tracking ID.

---

setTransId

public void **setTransId**(java.lang.String transid)

Set the Transaction ID.

Parameters:

transid - The new value of the Transaction Id.

---

setUdf1

public void **setUdf1**(java.lang.String udf1)

Set the user-defined field 1.

Parameters:

udf1 - The new value of the user-defined field 1.

---

setUdf2

public void **setUdf2**(java.lang.String udf2)

Set the user-defined field 2.

Parameters:

udf2 - The new value of the user-defined field 2.

---

setUdf3

public void **setUdf3**(java.lang.String udf3)

Set the user-defined field 3.

Parameters:

udf3 - The new value of the user-defined field 3.

---

setUdf4

public void **setUdf4**(java.lang.String udf4)

Set the user-defined field 4.

Parameters:

udf4 - The new value of the user-defined field 4.

---

setUdf5

public void **setUdf5**(java.lang.String udf5)

Set the user-defined field 5.

Parameters:

udf5 - The new value of the user-defined field 5.

setResourcePath

public void **setResourcePath**(java.lang.String resourcepath)

Set the resource path for the Gateway server.

Parameters:

resourcepath - The resource path.

Class NotEnoughDataException

java.lang.Object

|

+--java.lang.Throwable

|

+--java.lang.Exception

|

+--**NotEnoughDataException**

All Implemented Interfaces:

java.io.Serializable

public class **NotEnoughDataException**

extends java.lang.Exception

This exception is thrown if ProcessTransaction method of the e24CardPipe class is invoked without setting the values of any properties.

See Also:

- Serialized Form

Constructor Summary

NotEnoughDataException()
The empty constructor

NotEnoughDataException(java.lang.String msg)
The constructor with info.

Methods inherited from class java.lang.Throwable

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Methods inherited from class java.lang.Object

- clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

Constructor Detail

NotEnoughDataException

public NotEnoughDataException()

The empty constructor

NotEnoughDataException

public **NotEnoughDataException**(java.lang.String msg)

The constructor with info.

Parameters:

msg - The text message that describes the exception.

25: CF_eCardPipe

CF_e24CardPipe allows ColdFusion developers to integrate real-time automated credit card processing capabilities into their applications and websites. It is a simple inline tag that requires a handful of variables. When called, the CF_ e24CardPipe tag initiates a transaction-processing pipeline and returns the results of the credit card transaction.

File name: e24CardPipe.cfm

Install the CF_e24CardPipe

Copy the e24CardPipe

Copy the file e24CardPipe.cfm to the CustomTags directory located under the ColdFusion Installation directory.

Example: **c:\cfusion\CustomTags**

Implement the CF_e24CardPipe

CF_e24CardPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “TranPortal Raw Interface” section for more information about implementing the CF_e24CardPipe tag.

Tag Properties

<CF_e24CardPipe
id="TranPortal ID"
password="TranPortal Password"
action="A Valid Transaction Action"
type="A Valid Payment Instrument Type"
card="Card Number"
expyear="Card Expiration Year (YYYY)"
expmonth="Card Expiration Month (MM)"
expday="Card Expiration Day (DD)"
CVV2="Card CVV2 Code"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"
transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion application or website once the CF_e24CardPipe Tag has completed its execution. When the CF_e24CardPipe tag has finished execution, the variable **Success** will return a value of 0 (Failure) or 1 (Success). If you receive a **Success** value of 0, you should evaluate the **Error** result.

The results are dependent upon:

- SUCCESS (0 or 1)
- ERROR
- RESULT
- REF
- CARDBALANCE
- TRANSID
- TRACKID
- UDF1

- UDF2
- UDF3
- UDF4
- UDF5

Test the CF_e24CardPipe Tag

A very simple application page has been provided to test your tag. It indicates how to initiate the tag and sends a formatted transaction result to the browser.

Security Concerns - The CF_e24CardPipe Cold Fusion Tag uses Secure Socket Layer communications to transmit tag properties to the Transaction Processing Network. All transmitted data is safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

ACI Worldwide Inc.

26: CFX_e24CardPipe

CFX_e24CardPipe allows ColdFusion developers to integrate real-time automated credit card processing capabilities into their applications and websites. It is a simple inline tag that requires a handful of variables. When called the CFX_ e24CardPipe tag initiates a transaction processing pipeline and returns the results of the credit card transaction.

File name: e24CardPipe.dll

Install the CFX_e24CardPipe

Installation Procedures

1. Copy the file e24CardPipe.dll to the ColdFusion Bin directory located under the ColdFusion Installation directory. Example: c:\cfusion\bin\
2. Go to your ColdFusion Documentation. This is normally located at: http://www.yourdomain.com/cfdocs/
3. Select your ColdFusion Administrator and select the **Tags** button on your left.
4. In the form field to add a new CFX tag type the following: **CFX_ e24CardPipe** and click the Add button once.
5. Verify the location of the e24CardPipe.dll specifies **Server Library**. Example: c:\cfusion\bin\e24CardPipe.dll
6. Verify that the Procedure is **ProcessTagRequest**.

Implement the CFX_e24CardPipe

CFX_e24CardPipe operates in a similar fashion to an object, OCX/ActiveX control, or JavaBean. You provide the tag with a handful of properties and it performs the methods and events needed to process a real-time automated credit card transaction. The tag provides the transaction results as variables, which can be modified.

Implementation Procedures

Refer to the “TranPortal Raw Interface” section for more information about implementing the CFX_e24CardPipe tag.

Tag Properties

<CFX_e24CardPipe
resourcepath="Resource Path."
alias="Terminal Alias."
action="A Valid Transaction Action"
type="A Valid Payment Instrument Type"
card="Credit Card To Debit"
expyear="Card Expiration Year (YYYY)"
expmonth="Card Expiration Month (MM)"
expday="Card Expiration Day (DD)"
CVV2="Card CVV2 Code"
amt="Amount of transaction NNNNN.NN"
currency="ISO Currency code"
transid="Transaction ID"
trackid="Merchant Tracking ID">
udf1="User Defined Field 1">
udf2="User Defined Field 2">
udf3="User Defined Field 3">
udf4="User Defined Field 4">
udf5="User Defined Field 5">

Tag Results

The following variables will be available to your Cold Fusion Application or Website once the CFX_e24CardPipe Tag has completed its execution.

SC_RESULT
SC_REF
SC_CARDBALANCE
SC_TRANSID
SC_TRACKID
SC_UDF1
SC_UDF2
SC_UDF3
SC_UDF4
SC_UDF5
SC_SUCCESS- A boolean value indicating if the transaction was successful or not. "YES" if successful; "NO" otherwise.
SC_ERROR- Descriptive text of the error if the transaction was not successful.

Retrieving the Results

To retrieve the tag results use a ColdFusion routine similar in respect to the following code:

```
<CFOUTPUT>
<CFIF sc_Success>
Success!<br>
Result is #sc_result#.<br>
Ref is #sc_ref#.<br>
CardBalance is #sc_cardbalance#.<br>
TransId is #sc_transid#.<br>
TrackId is #sc_trackid#.<br>
<CFELSE>
Failure!<br>
The error is #sc_error#
</CFIF>
</CFOUTPUT>
```

Test the CFX_e24CardPipe Tag

A very simple application page has been provided to test your tag. It indicates how to initiate the tag and sends a formatted transaction result to the browser.

Security Concerns - The CF_e24CardPipe ColdFusion Tag uses Secure Socket Layer Communications to transmit tag properties to the Transaction Processing Network. Therefore, all of your transmitted data is extremely safe and secure. It is important that you implement SSL in your ColdFusion application or website when collecting a consumer's financial data.

27: ASP e24CardPipe

The e24CardPipe ActiveX/COM Object provides developers with a set of properties and methods to perform real-time credit card transactions securely across the Internet. This object can be used in an Active Server Pages environment. Please note, since ASP projects do not support asynchronous communications and processing, the e24CardPipe object will not return status messages. A simple ASP demonstration is provided below.

File name: e24CardPipe.dll

```
<%Dim MyObj

Set MyObj = Server.CreateObject("e24CardPipe.e24CardPipe.1")

MyObj.ResourcePath = "c:\resource path\"

MyObj.Alias = "Terminal Alias"

MyObj.Action = "10"

MyObjType = “SV”

MyObj.Card = "4444333322221111"

MyObj.ExpYear = "2005"  'YYYY

MyObj.ExpMonth = "12"  'MM

MyObj.ExpDay = “31” ‘DD

MyObj.CVV2 = "123"

MyObj.Amt = "10.00"

MyObj.Currency = "840"

MyObj.TrackId = strTrackID 'TODO create a new trackid for each transaction
```

```
MyObj.Udf1 = "Your UserDefined Field 1"

MyObj.Udf2 = "Your UserDefined Field 2"

MyObj.Udf3 = "Your UserDefined Field 3"

MyObj.Udf4 = "Your UserDefined Field 4"

MyObj.Udf5 = "Your UserDefined Field 5"

'perform the transaction

Dim TransVal,varResult, varRef, varTransId, varTrackId,

varErrorMsg,varRawResponse

TransVal = MyObj.PerformTransaction 'returns 0 for success and -1 for failure

varRawResponse = MyObj.RawResponse

varResult = MyObj.Result

varRef = MyObj.Ref

varCardBalance = MyObj.CardBalance

varTransId = MyObj.TransId

varErrorMsg = MyObj.ErrorMsg%>

<%Response.Write("<P>Transval  = " &Transval)

Response.Write("<P>varRawResponse  = " & varRawResponse )

Response.Write("<P>varResult  = " & varResult )

Response.Write("<P>varRef  = " & varRef )

Response.Write(“<P>varCardBalance = " & varCardBalance )

Response.Write("<P>varTransId  = " &  varTransId)

Response.Write("<P>varErrorMsg  = " & varErrorMsg)%>
```