

SAGE PAYMENT SOLUTIONS

GATEWAY

HTTPS Bankcard Specifications

Revised 12/08/2006

sage

Overview - Bankcard Services

The Gateway offers a wide variety of connectivity options for processing Bankcard transactions, including:

- HTTPS Post – a direct request / response packet processing methodology based on the documentation and specification provided within this document
- XML Web Services – an XML Web Service method of connectivity leveraging HTTP POST, HTTP GET, and SOAP protocols over SSL for a variety of processing, reporting, and other features. Please refer to the specific documentation related to XML Web Services for more information.
- Virtual Terminal – a Browser Based environment for transaction reporting, manual transactions, recurring transactions, integrated shopping cart features, etc.
- Virtual Terminal's Shopping Cart – The integrated shopping cart within the Virtual Terminal.
- Third Party Shopping Cart Vendors – Shopping Cart solutions provided by third party vendors which is integrated to either this specification for transaction processing or the Forms Shopping Cart specification for handing transactions off to the Gateway for completion.

When to Use the HTTPS Processing Method

The HTTPS processing method is best suited for the following environments:

- Website's which will securely acquire, via SSL, all necessary information required for transaction processing and who simply wish to transmit the transaction information for authorization while maintaining the direct interaction with the consumer. In such a case, the consumer will not be aware that the transaction is being transmitted to , as and the gateway are accessed by your application/website behind the scenes.
- Third Party Shopping Carts which securely acquire, via SSL, all necessary information required for transaction processing. These transactions will be processed and the Third Party Shopping Cart will, based upon the authorization response, update inventory, product databases, and generate invoices/confirmations for delivery via email.
- Any other environment where the 'Authorization' of bankcard transactions should occur in a transparent mode to the consumer, including retail applications, proprietary accounting and billing systems, etc.

Bankcard Authorization Specification - Request

Post Location: <https://www.sagepayments.net/cgi-bin/eftBankcard.dll?transaction>

Field Specifications

Field/Data Element	Required	Field Description
M_id	Required	12 Digit Merchant Identification (Virtual Terminal ID) Number Required for Gateway Access
M_key	Required	12 Digit Merchant Key Required for Gateway Access
C_name	Required	Cardholder Name
C_address	Required	Cardholder Billing Address
C_city	Required	Cardholder Billing City
C_state	Required	Cardholder Billing State
C_zip	Required	Cardholder Billing Zip/Postal Code
C_country	Optional	Cardholder Billing Country
C_email	Required	Cardholder Email Address
C_cardnumber	Required	Cardholder Bankcard Account Number
C_exp	Required	Bankcard Expiration Date (MMYY)
T_amt	Required	Transaction Amount (#####0.00)
T_code	Required	Transaction Processing Code <ul style="list-style-type: none"> • 01 = Sale • 02 = AuthOnly • 03 = Force/PriorAuthSale • 04 = Void • 06 = Credit • 11 = PriorAuthSale by Reference* *Requires T_reference
T_ordernum	Optional	Unique 1 to 20 Digit Order Number

T_auth	Optional	Previous (VOICE) Authorization Code, Required for Force Transactions
T_reference	Optional	Unique Reference, Required for Void and Prior Auth Transactions
T_trackdata	Optional	Track 2 Data for POS Applications
C_cvv	Optional	Card Verification Value
T_customer_number	Optional	Customer Number for Purchase Card Level II Transactions
T_tax	Optional	Tax Amount (#####0.00)
T_shipping	Optional	Shipping Amount (#####0.00)
C_ship_name	Optional	Shipping Recipient
C_ship_address	Optional	Shipping Recipient Address
C_ship_city	Optional	Shipping Recipient City
C_ship_state	Optional	Shipping Recipient State
C_ship_zip	Optional	Shipping Recipient Zip/Postal Code
C_ship_country	Optional	Shipping Recipient Country
C_telephone	Optional	Customer Telephone Number
C_fax	Optional	Customer Fax Number
T_recurring	Optional	1=Add as a Recurring Transaction
T_recurring_amount	Optional	Recurring Amount
T_recurring_type	Optional	1=Monthly, 2=Daily
T_recurring_interval	Optional	Recurring Interval
T_recurring_non_business_days	Optional	0=After, 1=Before, 2=That Day
T_recurring_start_date	Optional	Recurring Start Date MM/DD/YYYY
T_recurring_indefinite	Optional	1=Yes Indefinite, 0=No
T_recurring_times_to_process	Optional	Times To Process the Recurring Transaction
T_recurring_group	Optional	Recurring Group ID to Add the Recurring Transaction under

T_recurring_payment	Optional	Merchant initiated recurring transaction
---------------------	----------	--

Bankcard Authorization Specification - Response

Upon successful communication with the Gateway, a response packet will be received which should be parsed with the following specification.

Field/Data Element	Length	Start Position	Description
STX	1	1	ASCII 02, Start of Message Indicator
Approval Indicator	1	2	<ul style="list-style-type: none"> • A = Approved • E = Front-End Error / Non-Approved • X = Gateway Error / Non-Approved
Code	6	3	Approval or Error Code
Message	32	9	Approval or Error Message
Front-End	2	41	Front-End Indicator (Internal Use Only)
CVV Indicator	1	43	<ul style="list-style-type: none"> • M = Match • N = CVV No Match • P = Not Processed • S = Merchant Has Indicated that CVV2 Is Not Present • U = Issuer is not certified and/or has not provided Visa Encryption Keys
AVS Indicator	1	44	<ul style="list-style-type: none"> • X = Exact; match on address and 9 Digit Zip Code • Y = Yes; match on address and 5 Digit Zip Code • A = Address matches, Zip does not • W = 9 Digit Zip matches, address does not • Z = 5 Digit Zip matches, address does not • N = No; neither zip nor address match • U = Unavailable • R = Retry • E = Error • S = Service Not Supported • “ “ = Service Not Supported
Risk Indicator	2	45	<ul style="list-style-type: none"> • 01 = Max Sale Exceeded • 02 = Min Sale Not Met • 03 = 1 Day Volume Exceeded • 04 = 1 Day Usage Exceeded • 05 = 3 Day Volume Exceeded • 06 = 3 Day Usage Exceeded • 07 = 15 Day Volume Exceeded • 08 = 15 Day Usage Exceeded • 09 = 30 Day Volume Exceeded • 10 = 30 Day Usage Exceeded • 11 = Stolen or Lost Card • 12 = AVS Failure
Reference	10	47	Unique Reference Code
Field Separator	1	57	ASCII 28, Field Separator
Order Number	Variable	N/A	Order Number
Field Separator	1	N/A	ASCII 28, Field Separator

Recurring Indicator	1	N/A	1 = Added as a Recurring Transaction
Field Separator	1	N/A	ASCII 28, Field Separator
ETX	1	N/A	ASCII 03, End of Message Indicator

Bankcard Settlement/Release Specification - Request

Post Location: <https://www.sagepayments.net/cgi-bin/eftBankcard.dll?settlement>

Field Specifications

Field/Data Element	Required	Description
M_id	Required	12 Digit Merchant Identification Number (Virtual Terminal ID) Required for Gateway Access
M_key	Required	12 Digit Merchant Key Required for Gateway Access
B_count	Optional	Batch Item Count for Verification (-1 for Inquiry)
B_net	Optional	Batch Net for Verification (0.00 for Inquiry)

Bankcard Settlement/Release Specification - Response

Upon successful communication with the Gateway, a response packet will be received which should be parsed with the following specification.

Field/Data Element	Length	Start Position	Description
STX	1	1	ASCII 02, Start of Message Indicator
Approval Indicator	1	2	<ul style="list-style-type: none">• A = Approved• E = Front-End Error / Non-Approved• X = Gateway Error / Non-Approved
Batch Number	6	3	Batch Number
Message	32	9	Approval or Error Message
Batch Item Count	6	41	Batch Transaction Count
Batch Net	12	47	Batch Net
Reference	10	59	Unique Reference Code
Field Separator	1	69	ASCII 28, Field Separator
ETX	1	70	ASCII 03, End of Message Indicator

Error Codes/Messages

A wide variety of error messages and error codes may be returned. The actual error codes and error messages returned for switched transactions (transactions with a response indicator of 'E') varies depending on which front-end network is utilized for transaction routing. For assistance with these error codes and messages, please contact technical support. For gateway generated error codes (transactions with a response indicator of 'X'), the following table is provided.

Error Code	Response Message
000000	INTERNAL SERVER ERROR
900000	INVALID T_ORDERNUM
900001	INVALID C_NAME
900002	INVALID C_ADDRESS
900003	INVALID C_CITY
900004	INVALID C_STATE
900005	INVALID C_ZIP
900006	INVALID C_COUNTRY
900007	INVALID C_TELEPHONE
900008	INVALID C_FAX
900009	INVALID C_EMAIL
900010	INVALID C_SHIP_NAME
900011	INVALID C_SHIP_ADDRESS
900012	INVALID C_SHIP_CITY
900013	INVALID C_SHIP_STATE
900014	INVALID C_SHIP-ZIP
900015	INVALID C_SHIP_COUNTRY
900016	INVALID C_CARDNUMBER
900017	INVALID C_EXP
900018	INVALID C_CVV

900019	INVALID T_AMT
900020	INVALID T_CODE
900021	INVALID T_AUTH
900022	INVALID T_REFERENCE
900023	INVALID T_TRACKDATA
900024	INVALID T_TRACKING_NUMBER
900025	INVALID T_CUSTOMER_NUMBER
910000	SERVICE NOT ALLOWED
910001	VISA NOT ALLOWED
910002	MASTERCARD NOT ALLOWED
910003	AMEX NOT ALLOWED
910004	DISCOVER NOT ALLOWED
910005	CARD TYPE NOT ALLOWED
911911	SECURITY VIOLATION
920000	ITEM NOT FOUND
920001	CREDIT VOL EXCEEDED
920002	AVS FAILURE
999999	INTERNAL SERVICE ERROR

Samples

The following samples demonstrate the basics of creating the HTTPS POST and parsing the gateway response. For additional information and support, please contact Technical Support at 1-877-470-4001 for assistance.

Active Server Pages (ASP) Sample

The bankcard authorization specification is built to accept SSL Posts with the data provided in accordance with the bankcard authorization specification. Building the request string is relatively simple. The difficult part is how to perform the SSL post from within the ASP environment. Fortunately, Microsoft's suite of XML DOM (Document Object Model) components includes the XMLHTTP object. This object was originally designed to provide client-side access to XML documents on remote servers through the HTTP protocol. It exposes a simple API which allows you to send requests and get the resultant XML, HTML or binary data.

If you've been keeping up with versions of Internet Explorer, you probably have some version of MSXML installed. If you need to install it, you can either install the latest version of MDAC or the MSXML component itself. If you don't have it installed, you may get the error: Invalid ProgID when you execute `Server.CreateObject("Microsoft.XMLHTTP")`.

```
<%  
  
    'Bankcard Example Using ASP and Microsoft XMLHTTP Object  
  
    '=====  
  
    'Note: This sample uses only the minimum fields required,  
  
    ' consult the current authorization specification for  
  
    ' additional fields.  
  
    '=====  
  
    Option Explicit  
  
    Dim obj  
  
    Dim request_string  
  
    Dim response_string  
  
    'Create Request String Per Bankcard Authorization Spec  
  
    '=====  
  
    request_string = "M_id=999999999999"
```

```
request_string = request_string & "&" & "M_key=AAAAAAAAAAAA"
request_string = request_string & "&" & "C_name=" & "John Doe"
request_string = request_string & "&" & "C_address=" & "1234 AnyStreet"
request_string = request_string & "&" & "C_city=" & "AnyTown"
request_string = request_string & "&" & "C_state=" & "TX"
request_string = request_string & "&" & "C_zip=" & "12345"
request_string = request_string & "&" & "C_country=" & "US"
request_string = request_string & "&" & "C_email=" & "none@none.com"
request_string = request_string & "&" & "C_cardnumber=" & "4111111111111111"
request_string = request_string & "&" & "C_exp=" & "0105"
request_string = request_string & "&" & "T_code=" & "01"
request_string = request_string & "&" & "T_amt=" & "1.00"
```

'=====

'Create Object

'=====

```
Set obj = Server.CreateObject("Microsoft.XMLHTTP")
```

'Post Request

'=====

```
obj.Open "POST", "https://www.sagepayments.net/cgi-bin/efBankcard.dll?transaction", False
```

```
obj.Send request_string
```

'Obtain Response

'=====

```
response_string = obj.responseText
```

'Parse Response and Print Out Results Per Spec

```
'=====
Response.Write "Full Packet Response: " & response_string & "<BR>"
Response.Write "Approval Indicator: " & mid(response_string,2,1) & "<BR>"
Response.Write "Approval/Error Code: " & mid(response_string,3,6) & "<BR>"
Response.Write "Approval/Error Message: " & mid(response_string,9,32) & "<BR>"
Response.Write "Front-End Indicator: " & mid(response_string,41,2) & "<BR>"
Response.Write "CVV Indicator: " & mid(response_string,43,1) & "<BR>"
Response.Write "AVS Indicator: " & mid(response_string,44,1) & "<BR>"
Response.Write "Risk Indicator: " & mid(response_string,45,2) & "<BR>"
Response.Write "Reference: " & mid(response_string,47,10) & "<BR>"
Response.Write "Order Number: " & mid(response_string, _
(InStr(1, response_string, Chr(28)) + 1), InStr((InStr(1, response_string, Chr(28)) + 1), _
response_string, Chr(28)) - (InStr(1, response_string, Chr(28)) + 1)) & "<BR>"
```

'Destroy Object

```
'=====
Set obj = Nothing
%>
```

ASP.Net Sample

The bankcard authorization specification is built to accept SSL Posts with the data provided in accordance with the bankcard authorization specification. Building the request string is relatively simple. The difficult part is how to perform the SSL post from within the ASP.Net environment. Fortunately, Microsoft's .Net includes the HttpWebRequest object. It exposes a simple API which allows you to send a HTTP request and get the resultant data.

```
Imports System.Net
Imports System.IO
Imports System.Web

Dim request As String

'Create Request String Per Bankcard Authorization Spec
request = "M_id=999999999999"
request = request & "&" & "M_key=AAAAAAAAAAAA"
request = request & "&" & "C_name=" & "John Doe"
request = request & "&" & "C_address=" & "1234 AnyStreet"
request = request & "&" & "C_city=" & "AnyTown"
request = request & "&" & "C_state=" & "TX"
request = request & "&" & "C_zip=" & "12345"
request = request & "&" & "C_country=" & "US"
request = request & "&" & "C_email=" & "none@none.com"
request = request & "&" & "C_cardnumber=" & "4111111111111111"
request = request & "&" & "C_exp=" & "0105"
request = request & "&" & "T_code=" & "01"
request = request & "&" & "T_amt=" & "1.00"

'Create Request Object
Dim webRequest As HttpWebRequest =
    webRequest.Create("https://www.sagepayments.net/cgi-
    bin/efBankcard.dll?transaction")

webRequest.Method = "POST"
webRequest.ContentType = "application/x-www-form-urlencoded"

'Write Request Bytes To Stream
Dim outByte() As Byte = System.Text.Encoding.ASCII.GetBytes(request)
webRequest.ContentLength = outByte.Length

Dim outputStream As Stream = webRequest.GetRequestStream()
outputStream.Write(outByte, 0, outByte.Length)

'Read Response Bytes From Stream
Dim webResponse As HttpWebResponse = webRequest.GetResponse()
Dim statusCode As Integer = webResponse.StatusCode

Dim sr As New StreamReader(webResponse.GetResponseStream(),
    System.Text.Encoding.ASCII)

Dim response As String = sr.ReadToEnd()
```

PHP Sample

If you wish to perform a bankcard transaction from within a PHP environment then we recommend using the CURL library. It can be used in two forms, the preferred, internal method and the external executable. In order to use the internal method the version of PHP installed on your machine needs to have been compiled with `--with-curl` (see PHP.net for details). If your vendor has not provided this and you cannot compile and install PHP yourself then you can use the curl executable from within PHP. This is less efficient and you must take care when exposing a command line in this fashion. All that is needed for this method is to have a working curl executable installed on the server. While other methods exist for doing an HTTPS POST in PHP, this is a relatively simple method that we will support.

```
<?
```

```
//set the URL that will be posted to.

$eftsecure_url = "https://www.sagepayments.net/cgi-bin/eftbankcard.dll?transaction";

//make your query.

$data = "m_id=" . $mid; //your eftsecure merchant id.

$data .= "&m_key=" . $mkey; // your eftsecure merchant key;

$data .= "&T_amt=" . urlencode($amount); // The amount to be charged. Always encode

// any data given to you over the web. it is more

// secure and reliable.

$data .= "&C_name=" . urlencode($cardholder_name);

$data .= "&C_address=" . urlencode($billing_address);

$data .= "&C_state=" . urlencode($billing_state);

$data .= "&C_city=" . urlencode($billing_city);

$data .= "&C_cardnumber=" . urlencode($cardnumber);

$data .= "&C_exp=" . urlencode($exp_Date);

$data .= "&C_zip=" . urlencode($billing_zip);

$data .= "&C_email=" . urlencode($email_addr);

$data .= "&T_code=02"; // transaction type indicator
```

```

//now we will make the transaction. the first method is the preferred internal method

//using the built in CURL functions.

$ch = curl_init(); //initialize the CURL library.

curl_setopt($ch, CURLOPT_URL, $ftsecure_url); // set the URL to post to.

curl_setopt($ch, CURLOPT_POST, 1); // tell it to POST not GET (you can GET but POST is
//preferred)

curl_setopt($ch, CURLOPT_POSTFIELDS, $data); // set the data to be posted.

curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1); // this tells the library to return the
// data to you instead of writing it to
// a file

$res = curl_exec($ch); // make the post.

curl_close($ch); // shut down the curl library.

echo $res . "<br>\n"; // this is the response.

// or you can also use the curl executable directly.

$curl = "/usr/bin/curl"; // the location of the curl executable.

$fp=popen("$curl -data \"$data\" $ftsecure_url","r"); // this allows you to get the
// data directly from curl

while(!feof($fp)) $res .= fread($fp, 1024); // read all the output.

pclose($fp); // close the connection to curl.

```

```
echo "$res" . "<br>\n"; // this is the response.

// once you have the response then you need to parse its different components.

echo "<br><hr><br>\n";

echo "Approval Indicator: " . $res[1] . "<br>"; //A is approved E is declined/error.

echo "Approval/Error Code: " . substr($res, 2, 6) . "<br>\n";

echo "Approval/Error Message: " . substr($res, 8, 32) . "<br>\n";

echo "Front-End Indicator: " . substr($res, 40, 2) . "<br>\n";

echo "CVV Indicator: " . $res[42] . "<br>\n";

echo "AVS Indicator: " . $res[43] . "<br>\n";

echo "Risk Indicator: " . substr($res, 44, 2) . "<br>\n";

echo "Reference: " . substr($res, 46, 10) . "<br>\n";

echo "Order Number: " . substr($res, strpos($res, chr(28)) + 1,
    strpos($res, chr(28) - 1)) . "<br>\n";
```

?>

Perl Sample

If you wish to perform a bankcard transaction from within a Perl environment then we recommend using the LWP library. This package has several dependencies but is fairly simple to perform. You will need the LWP, NET::HTTP and Crypt::SSLeay modules to do an HTTPS POST. URI and CGI modules can be not completely necessary depending on what it is you are trying to do and your proficiency at perl. For the sake of simplicity and because it is assumed that most people will be using this in a web-based application we will use them for this example.

```
use LWP::UserAgent; # you must have installed and use these modules for an https post.

use HTTP::Request;

use URI::Escape;

use Crypt::SSLeay;

use CGI;

$cgi = CGI::new(); # create a cgi object.

$data = "m_id=" . $mid;

$data .= "&m_key=" . $mkey;

$data .= "&T_amt=" . uri_escape($amount, "0-9a-zA-Z"); # you should encode all data

$data .= "&C_name=" . uri_escape($customer_name, "0-9a-zA-Z"); # given to you by the customer.

$data .= "&C_address=" . uri_escape($billing_address, "0-9a-zA-Z");

$data .= "&C_state=" . uri_escape($billing_state, "0-9a-zA-Z");

$data .= "&C_city=" . uri_escape($billing_city, "0-9a-zA-Z");

$data .= "&C_cardnumber=" . uri_escape($cardnumber, "0-9a-zA-Z");

$data .= "&C_exp=" . uri_escape($exp_date, "0-9a-zA-Z");

$data .= "&C_zip=" . uri_escape($billing_zip, "0-9a-zA-Z");

$data .= "&C_email=" . uri_escape($email_address, "0-9a-zA-Z");

$data .= "&T_code=02";
```

```

$ua = new LWP::UserAgent;

my $req = new HTTP::Request('POST', # create the HTTP Request object and set its parameters.
    'https://www.sagepayments.net/cgi-bin/efbankcard.dll?transaction');

$req->content_type("application/x-www-form-urlencoded"); # add the appropriate headers.

$req->content_length("" . length($data));

$req->content($data); # this is the data to be sent.

$res = $ua->request($req); # make the request.

print $cgi->header(); # this is to send out the response HTTP headers

print "Request failed." and die unless $res->is_success; # check for failure

print $res->content . "<br><br><hr><br>\n"; #this is the response

$res = $res->content; # for simplicity lets just handle the content of the response.

print "Approval Indicator: " . substr($res,1, 1) . "<br>"; #A is approved E is declined/error.

print "Approval/Error Code: " . substr($res, 2, 6) . "<br>\n";

print "Approval/Error Message: " . substr($res, 8, 32) . "<br>\n";

print "Front-End Indicator: " . substr($res, 40, 2) . "<br>\n";

print "CVV Indicator: " . substr($res, 42, 1) . "<br>\n";

print "AVS Indicator: " . substr($res, 43, 1) . "<br>\n";

print "Risk Indicator: " . substr($res, 44, 2) . "<br>\n";

print "Reference: " . substr($res, 46, 10) . "<br>\n";

print "Order Number: " . substr($res, index($res, chr(28)) + 1,
    rindex($res, chr(28)) - index($res, chr(28)) - 1) . "<br>\n";

```

ColdFusion Sample

If you wish to perform a bankcard transaction from within a ColdFusion environment then we recommend using the following script code below and the latest ColdFusion version from Marcomedia. Currently ColdFusion URL encodes the form keys as well as the data and this will cause incorrect parsing of variables POSTed to the gateway. To fix this, developers should set the cfhttpparam attribute "encoded" to "false". Future updates to the ColdFusion runtime might correct this behavior so you should try it first without this attribute.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>Untitled</title>

</head>

<body>

<cfhttpurl="https://www.sagepayments.net/cgi-bin/efitBankcard.dll?transaction" port="443" method="post">

<cfhttpparam NAME="M_id" value="" type="formfield" >

<cfhttpparam NAME="M_key" value="" type="formfield" >

<cfhttpparam NAME="C_name" value="John Doe" type="formfield" >

<cfhttpparam NAME="C_address" value="1234 Any Street" type="formfield">

<cfhttpparam NAME="C_city" value="AnyTown" type="formfield" >

<cfhttpparam NAME="C_state" value="TX" type="formfield" >

<cfhttpparam NAME="C_zip" value="12345" type="formfield" >

<cfhttpparam NAME="C_country" value="US" type="formfield" >

<cfhttpparam NAME="C_email" value="none@none.com" type="formfield" >

<cfhttpparam NAME="C_cardnumber" value="4111111111111111" type="formfield" >

<cfhttpparam NAME="C_exp" value="1005" type="formfield" >

<cfhttpparam NAME="T_code" value="01" type="formfield" >

<cfhttpparam NAME="T_amt" value="1.00" type="formfield" > </cfhttp>
```

```
<cfset response_string = cfhttp.FileContent>

<cfset ApprovalIndicator= mid(response_string,2,1)> <cfset App_Error_Code=mid(response_string,3,6)>

<cfset App_Error_Message=mid(response_string,9,32)>

<cfset FrontEndIndicator=mid(response_string,41,2)>

<cfset CVVIndicator=mid(response_string,43,1)>

<cfset AVSIndicator=mid(response_string,44,1)>

<cfset RiskIndicator=mid(response_string,45,2)>

<cfset Reference=mid(response_string,47,10)>

<cfoutput>

Status Code: #cfhttp.statuscode#<br>

Approval Indicator: #ApprovalIndicator#<br> Approval/Error Code:

#App_Error_Code#<br> Approval/Error Message: #App_Error_Message#<br> Front End Indicator: #FrontEndIndicator#<br> CVV
Indicator:

#CVVIndicator#<br> AVS Indicator: #AVSIndicator#<br> Risk Indicator:

#RiskIndicator#<br>

Reference: #Reference#<br>

<hr>

</cfoutput>

</body>

</html>
```