Portrait
Foundation

Client Event Delivery User Guide

Edition 5.1

10 January 2013

Pitney Bowes
Software

Portrait Foundation
Client Event Delivery User Guide

Acknowledgement of trademarks

Other product names, company names, marks, logos and symbols referenced herein may be the trademarks or registered trademarks of their registered owners.

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc.

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Software Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

To describe how to implement Client Events in a Portrait Application.

Intended audience

Project teams interested in Client Event Delivery.

Related documents

Client Event Delivery Overview.

Extending Applications using .NET *(Portrait NET SDK)*

Extending Applications using C++ and COM *(Portrait SDK)*

Software release

Portrait Foundation 3.1 or later.

Contents

1 Introduction

This document covers the various integration points for Client Event Delivery (CEV), and how a Project can incorporate the functionality.

It is intended for use by project development teams familiar with the various technologies.

Users of this document should be familiar with the Client Events Overview. In addition, teams wishing to customise CEV should also be familiar with extending Portrait using the SDK.

The areas covered are:

- Integrating CEV Overview
- The Subscriber Control
- Communicating with the Control
- JavaScript support
- Event Handlers
- Sending Events.
- Customising CEV behaviour

NOTE: Client Events is NOT enabled by default in the vanilla Contact Centre application. To enable it, follow the instructions in the ClientEventsBase.aspx file, installed as part of the Implementation install. The contents of this aspx file can be seen in the appendix to this document.

2 Integrating CEV Overview.

Client events touch all tiers of a Portrait Application. However, integration can be relatively simple. The minimum steps required are:

- **The browser has to host an ActiveX control** — acts as the agent for events. It both sends commands and receives any events. The vanilla profile already loads the control and so, if the application is being based on that, little extra work is required. If not, it can be used as a template.
- **The control has to be initialised** — requires data that is part of the CEV configuration; to this end, a DAT has been written to extract this data and it is then sent to the control using a CI node. Details of this are shown later.
- **Events must be subscribed to** — a client will only receive an event when the relevant EventType and EventID have been subscribed to by that client. Subscription can happen from within browser script code, or from a Portrait Model.
- **Events must be fired** — can be achieved either by running the SendClientEvent node from within a Portrait Model, or by creating the relevant CEVManager COM object from a code module and calling the SendEvent method.

In addition, the following aspects of CEV operation can be customised by using the Portrait SDK (C++ or .NET):

Event Subscription:

This is the processing that takes place (on the CRM server) whenever a Client Subscribes to, or Unsubscribes from, an event.

Event Resolution:

The is the process that transforms a Send Event command into zero or more physical client addresses destined to receive that event.

Reference Data is used to control this customisation. A new Group has been created, called "ClientEvents". Whenever an event is subscribed to, unsubscribed from or fired then this new group is searched for a Reference Data Item with the name of the EventType. If this is found then the attributes of this RDI can contain the COM ProgIDs of the objects that are to used to customise the behaviour for this event.

NOTE: It is recommended that ALL EventTypes in use are entered into this Ref Data group, even if no values are supplied for the attributes, as you will see errors reported to the event log whenever an attempt to read RefData with a non-existent item is made.

There are some special considerations to take account of, these are detailed later.

3 The Subscriber Control

The ActiveX control that supports Client Events in the browser hosts a number of objects implementing the CEV interfaces:

IamcCevMessage Encapsulates the CEV message (common).

IamcCevListener Encapsulates the listener Messages (common).

IamcCevSender Encapsulates the sender for Messages (common).

IamcCEvents The control interface used by the browser.

_IamcCEventsEvents Interface used by control to fire events to the browser.

In the deployed Portrait Foundation environment, this control is also used to host the first three objects as a normal COM dll.

The control contains mechanisms to detect if it is installed on a non-Portrait machine and modifies its logging and settings behaviour accordingly.

If Portrait Foundation is not detected then trace and error information is sent to the Debug device using OutputDebugString.

Similarly, settings information uses an internal implementation of IamcCevSettings to provide access to the three common objects described above.

3.1 Hosting the control

The control is loaded into an Iframe created solely for the purpose. In normal circumstances this Iframe would be hidden (ie. A size of 0 x 0). However, there is a feature of IE whereby Controls loaded into hidden Iframes do not get windows created (even if they specify they are not windowless). Because the control events will only work when posted from the original thread (the events are triggered using a Windows PostMessage call from the thread handling the socket code) it is essential that a valid window handle is available. For this reason, the control is loaded into an Iframe with a size of 1x1 pixels. To stop any corruption on the display, the drawing code in the Control does nothing.

The Iframe is created by using the following code in the application .hta (in this case ContactCentre.hta). It is important that the border is set to 0, as this ensures that a 1x1 Iframe will have part of the control visible and thereby create a window.

```
<iframe name="CEV" src="ClientEventsBase.aspx" application="yes" scrolling="no"
    frameBorder=0 height="1px" width="1px">
</iframe>#
```

The aspx file ClientEventsBase.aspx loads the Control using the standard HTML OBJECT tag.

```
<OBJECT id="idCEvent"
  data=
     "data:application/x-oleobject;base64,ep9qcMEHm0WE41pgjK+tzQADAADYEwAA2BMAAA=="
  classid="clsid:706A9F7A-07C1-459B-84E3-5A608CAFADCD" name="idCEvent"
  VIEWASTEXT
  codebase="/portrait_client/controls/cevsubscriberU.dll#version=2.5.36.0">
</OBJECT>
```

This file also contains some JavaScript used to drive the control and handle any events that are sent. These are used by the code in CTISendMessage.aspx to

process a command received from a Portrait Model (see table 1). See appendix A for the complete ClientEventsBase.aspx listing.

In addition, there are a number of JavaScript functions supplied in a JavaScript include file supplied as part of the ADK (ClientEvents.js).

3.2 Initialising the Control

The control has to be initialised. The data required for this is the target address of the Event Publisher(s). This information is held in the Portrait Foundation environment as configuration data (managed using the Portrait Management Console). Therefore, this data needs to be extracted and passed to the browser in order that it can perform its initialisation.

This is achieved by using exactly the same process as Telephony to perform its own initialisation and is handled in the same model "Initialise PTC". A ClientEvents DAT is run to extract the required data and is formatted into an xml command sent to the browser using a CI node. This technique is detailed later (see the next section, Communicating with the Control).

Following a successful "Initialise" call, the client is "logged on" to Client events and can receive any events subscribed to.

4 Communicating with the Control

4.1 Locally, from the browser elements

The Iframe is called CEV and is a child of the topmost element. All communication is performed by referencing the path (usually through **parent.winMaster.top.CEV**). This is the same mechanism as used by the PTC.

As the Iframe also includes the ClientEvents support javascript file, all functions are available.

4.2 Remotely, from a Portrait Model

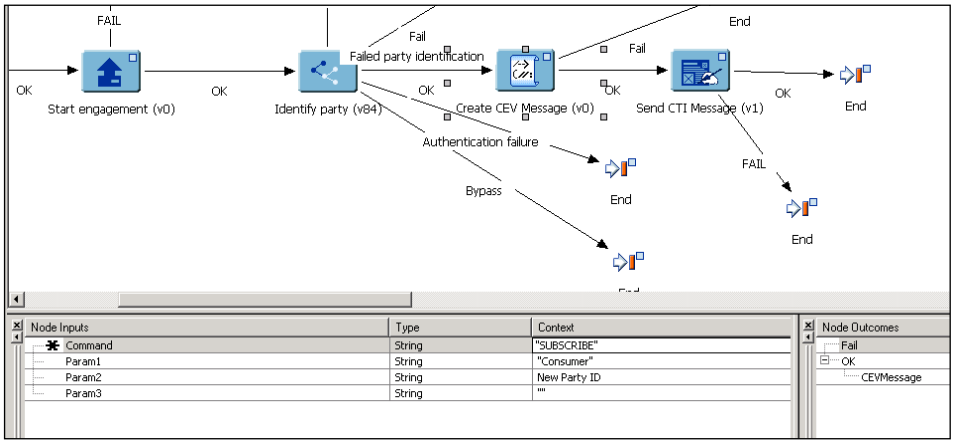
This is achieved via a Custom Interaction node. In fact, it uses the same node as PTC and is called Send CTI Message. This sends some XML to the browser and is processed by **CTISendMessage.aspx**. This file contains some JavaScript which passes the xml received to the relevant elements within the browser. In the case of ClientEvents, this is the **CEVCommand** function implemented in ClientEventsBase.aspx. The commands implemented and their parameters are as follows:

Table 1 – Portrait Model Comands

Command	Parameters	Description
INIT	Param1	"Ipaddress,port number" Address of the Event Publisher. This is sent to a client browser during the initialisation of the application framework model. When this is executed, the Control is initialised using its "Initialise" method. This results in a "Hello" command being sent to the Publisher, following which the Client is registered in the database.
SUBSCRIBE	Param1 Param2 Param2 (optional)	Event type to subscribe to. Event ID Handler name for event. This must be an available JavaScript function of the form detailed below. If not supplied, the default handler is used, which is a message box.
UNSUBSCRIBE		Event type to subscribe to. Event ID Removes the Client subscription from the event(s).
UNSUBSCRIBEALL		Unsubscribes the client from all events subscribed to using SUBSCRIBE.

The xml is created using a Script node (**Create CEV Message**), with the resultant output (**CEVMessage**) being an input to the **Send CTI Message** CI Node. The parameters above (param1 etc.) are mapped to the script node inputs, named parram1 etc.). The command is mapped to the input COMMAND.

In the following picture (from ProcessModeller) you can see the Create CEV Message script node, followed by the Send CTI Message CI node to cause the client to subsribe to an event. This is taken from a modified "Select Party from Take Call" model; it is subscribing the client to receive an event whenever the consumer that has been selected is changed.



5 JavaScript Support

ClientEvents.js provides the following functions:

```
function cevSubscribe( eventType, eventID, handler )
function cevUnsubscribe( eventType, eventID )
function cevUnsubsribeAll()
```

These are the functions that will be called whenever the commands detailed above are initiated from the Portrait Model. They can be called directly from client side JavaScript as well, should that prove easier.

6 Event Handlers

Event handlers are JavaScript functions of the form:

```
Function xxxHandler( eventType, eventID, eventPayload )
```

Any events triggered where no handler has been defined will cause a message box to be displayed, containing the details of the event.

Where a handler is defined, it can be one of the following:

- A string, the contents of which is treated as the name of the function.
- A function reference.

For example:

```
Function eventHandler( type, id, payload )
{
…
}

// subscribe to event as string
cevSubscribe( "type", "id", "eventHandler" );
…
// subscribe to event as reference
cevSubscribe( "type", "id", eventHandler );
```

When the subscribe is performed using the model nodes described above, the handler is always passed as a string.

7 Sending Events

The CRM server(s) host the Client Events Manager object. One of the methods on this is SendEvent.

An event is raised whenever this method is invoked, which can be done in two ways:

1 Including the SendClientEvent node in a Portrait Foundation model

2 Writing code to create the Manager Object and calling the method directly.

7.1 SendClientEvent node

The node has the following inputs defined:

EventType (string) The name of the event

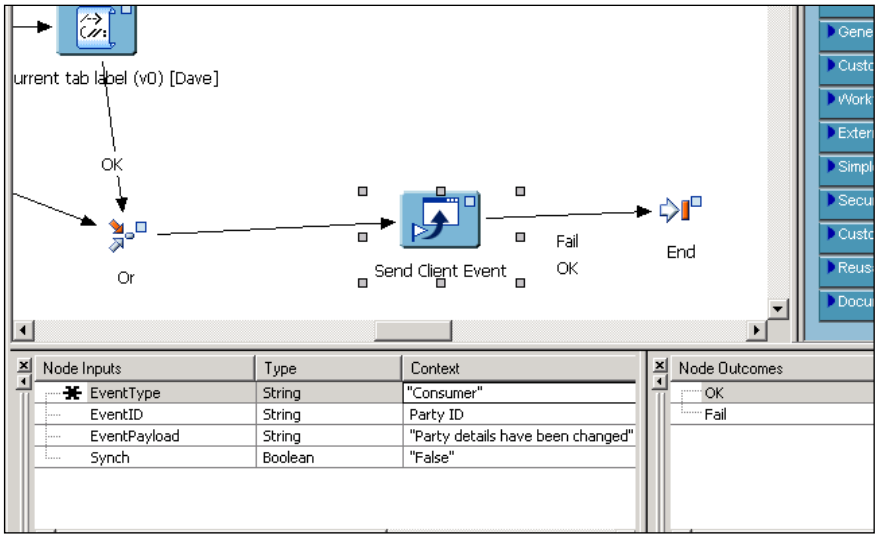
EventID (string) The ID of the event

EventPayload (string): Any extra data relevant to the event (typically this is only understood by the relevant Event Handler).

Synch (Boolean): True means that the node outcome will reflect failures during the resolution process (note that the actual event sending is always asynchronous). False means that the whole Send Event processing takes place asynchronously.

There are two outcomes, **OK** and **Fail**, there is no output associated with either. **Fail** is only generated when Synch is true and the resolution failed in some way.

The following picture is from a modified "Amend Personal Details" model, which fires the event when the details have been changed.



7.2 Writing code

The CEVManager object can be created explicitly within code. However, please note that, as it is hosted by the Portrait ServiceHost framework, attempts to create an instance should **always** use the ObjectBroker.

The following snippet is from a C# application:

```
// get ObjectBroker
OBJECTBROKERPROXYLib.IAmcOBProxy OB = new OBJECTBROKERPROXYLib.AmcOBProxyClass();
// create instance of the Cev Manager
Object omanager;
OB.GetObjectInstance( "AIT.AMC.ClientEvents.Manager", out omanager );
IamcCevManager manager = (IamcCevManager)omanager;
// fire an event
manager.SendEvent( 0,  "System", "", "Payload" );
```

8 Customising CEV Behaviour

Typically, whenever a component is built to customise Client Events processing for one or more event types, both objects would be provided as they provide a 'matched pair'. However, this is not enforced as there may be times when the process needs to be 'intercepted' rather than 'replaced'.

Whenever the Reference Data Item Attribute is missing, or is an empty string, then the default is used even if the other "matching" object has been customised.

If you want to call the default implementation from a customised one you should perform the following:

Create an instance of the CE Settings objects. This has a ProgID of 'AIT.Portrait.ClientEvents.Settings'. You can call the 'GetSubscription' or 'GetResolver' methods to create instances of the default objects.

NOTE: There is an inextricable link between Registration (the process of "logging on" and "logging off") and Subscription/Resolution. It is for this reason that it is recommended to use the default implementations from within your customised objects to maintain this link. If you do not, it is possible for the database data to lose integrity. If there are compelling reasons to proceed then the registration Object should be customised as well. This is catered for by configuration but is not supported in the SDK directly. Users should contact Portrait Support for assistance in this area.

8.1 Event subscription:

This is the processing that takes place (on the CRM server) whenever a Client Subscribes to, or Unsubscribes from, an event.

The interface that must be implemented is IamcCevSubscriber. The reader is referred to the relevant SDK guide in order to create the necessary boilerplate objects from the wizards provided.

The interface has two methods:

```
HRESULT Subscribe( [in] BSTR client_id,
                   [in] BSTR eventtype,
                   [in] BSTR eventID );
```

Registers the client for the specified event type and (optionally) event id.

```
HRESULT Unsubscribe( [in] BSTR client_id,
                     [in] BSTR eventtype,
                     [in] BSTR eventID );
```

client_id is the unique GUID for the client. This MUST be preserved as it will be required in any address resolution (see below).

eventtype and **eventID** describe the event the processing is being performed for.

What processing is performed here is entirely up to the user, however, it is not recommended to completely replace the default behaviour as links to the registration process will be difficult to maintain. It is always best to use the current database mechanism to store the subscription information.

8.2 Event resolution:

The is the process that transforms a Send Event command into zero or more physical client addresses destined to receive that event.

The interface that must be implemented is IamcCevSubscriber. The reader is referred to the relevant SDK guide in order to create the necessary boilerplate objects from the wizards provided.

The interface has the following method:

```
HRESULT Resolve( [in] BSTR eventType,
                 [in] BSTR eventID,
                 [out, retval] BSTR* addresses,
                 [out, retval] long* numAddresses );
```

eventType and **eventID** describe the event the processing is being performed for.

addresses is the result of the resolution. It takes the following format:

ggggggg/ss/aaa1/aaa2;[more addresses] where:

ggggggg — the client GUID

ss — the current suspect count

aaa1 — address1

aaa2 — address2

numAddresses — the number of addresses found

The manager will automatically handle the 'chunking' of the output if there are a large number of results.

Appendix A ClientEventsBase.aspx

```
<%@ Page language="c#" Codebehind="ClientEventsBase.aspx.cs" AutoEventWireup="false"
                                                Inherits="AIT.Portrait.Web.Applications.Cont
                                                actCentre.ClientEventsBase" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<!—
    THIS FILE CONTAINS THE BASE IMPLEMENTATION FOR CLIENT EVENTS
    IT IS **NOT** ACTIVATED BY DEFAULT
    TO ACTIVATE IT, READ THE COMMENTS DESCRIBED BELOW
-->
<html>
  <head>
    <title>ClientEventsBase</title>
    <meta name="GENERATOR" Content="Microsoft Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name=vs_defaultClientScript content="JavaScript">
    <meta name=vs_targetSchema content="http://schemas.microsoft.com/intellisense/ie5">
 </head>

    <SCRIPT language="JavaScript" src="/portrait_client/includes/generic.js"></SCRIPT>
    <SCRIPT language="JavaScript"
                                                src="/portrait_client/includes/clientevents.
                                                js"></SCRIPT>
    <SCRIPT language="JavaScript">
        function InitObjects()
        {
        }
        function StartPage()
        {
        }
        function PageBeforeUnload()
        {
        }

        function sysHandler( eventType, eventID, eventPayload )
        {
            AMCAlert( "System Message: " + eventPayload );
        }

        function CEVCommand( xml )
        {
            //##########################################
            // REMOVE THE NEXT LINE TO ACTIVATE CLIENT EVENTS
            return;            try
            {
                var xmldoc = new ActiveXObject("Microsoft.XMLDOM");
                var result = xmldoc.loadXML( xml );
                if ( result )
                {
                    var node = xmldoc.selectSingleNode( "//request/name" );
                    if ( node != null )
                    {
                        // get data from message
                        var xnode = xmldoc.selectSingleNode(
                                                "//request/parameters/param1" );
```

```
            var xnode2 = xmldoc.selectSingleNode(
                                "//request/parameters/param2" );
            var xnode3 = xmldoc.selectSingleNode(
                                "//request/parameters/param3" );
            var p1 = "";
            var p2 = "";
            var p3 = "";
            if ( xnode != null )
            {
                p1 = xnode.text;
            }
            if ( xnode2 != null )
            {
                p2 = xnode2.text;
            }
            if ( xnode3 != null )
            {
                p3 = xnode3.text;
            }
            // execute the command
            switch( node.text )
            {
                case "INIT":
                {
                    var addresses = p1.split( "," );
                    if ( typeof(addresses[0]) == "string" &&
                                typeof(addresses[1]) == "string" )
                    {
                        idCEvent.Initialise( addresses[0], addresses[1] );
                        cevSubscribe( "System", "", sysHandler );
                    }
                    else
                    {
                        throw "Data Error: " + node.text + "[" + p1 + "]";
                    }
                }
                break;

                case "SUBSCRIBE":
                {
                    cevSubscribe( p1, p2, p3 );
                }
                break

                case "UNSUBSCRIBE":
                {
                    cevUnsubscribe( p1, p2 );
                }
                break

                case "UNSUBSCRIBEALL":
                {
                    cevUnsubscribeAll();
                }
                break
            }
        }
    }
    else
```

```
            {
                throw "Cannot load xml";
            }
        }
        catch( e )
        {
            AMCAlert( "An error has occurred in Client Events processing\nWhilst this
                                    is not catastrophic,\n you should inform
                                    your System Administrator\n [" + e + "]\n ["
                                    + xml + "]");
        }
    }

</SCRIPT>


<body>

<!—
    ################################################
    REMOVE THE COMMENT TAGS AND THESE 3 LINES TO ACTIVATE CLIENT EVENTS
    ################################################
    <OBJECT id="idCEvent" data="data:application/x-
                                    oleobject;base64,ep9qcMEHm0WE41pgjK+tzQADAAD
                                    YEwAA2BMAAA=="
        classid="clsid:706A9F7A-07C1-459B-84E3-5A608CAFADCD" name="idCEvent"
                                    VIEWASTEXT
        codebase="/portrait_client/controls/cevsubscriberU.dll#version=2.5.36.0">
    </OBJECT>

    <script language="VBScript">
        Function idCEvent_OnEvent( eventType, eventID, eventPayload )
            OnClientEvent eventType, eventID, eventPayload
        End Function
    </script>
-->

</body>
```

</html>