Portrait
Foundation

Client Event Delivery Overview

Edition 5.1

10 January 2013

Pitney Bowes
Software

Portrait Foundation
Client Event Delivery Overview

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 provide an overview of the infrastructure for delivery of events to Clients of Portrait Foundation.

Intended audience

Portrait Foundation developers interested in an overview of the Client Event Delivery mechanism.

Related documents

Client Event Delivery Developers Guide

Software release

Portrait Foundation 3.1 or later.

Contents

1 Introduction

Portrait Client Event Delivery provides a mechanism to push information out to Portrait Foundation clients. This functionality is not readily available in a browser-based implementation because web technology essentially uses a "pull" approach, whereby the client requests information from the server. In some cases, it is acceptable for the client to poll the server for relevant information, for example by periodically refreshing a web page. However, this approach is expensive from a resource usage perspective and means that information is only available to a client the next time it refreshes a page.

The type of information expected to be pushed includes:

- Notification that page content has been updated. For example, Portrait user Fred may have updated a record currently being viewed by Bob. Bob's browser needs to receive a notification of the update so that it can take appropriate action, such as displaying a "Refresh" button or automatically refreshing a page.
- Notification of events from an external application, possibly one that shares the user's desktop. For example, an external application may inform Portrait Foundation that it has just identified a customer, and Portrait Foundation may be required to display the relevant customer details.
- Notification of any other events that may affect the user. For example this could include "system" level information such as database availability (or lack of it).

This document provides an overview of just such a Client Event mechanism for Portrait Foundation. The design is based on the following fundamental tenets:

- The Clients will subscribe/unsubscribe to the mechanism, allowing it to receive only those events it is interested in.
- The Events will be generated mainly (but not exclusively) on the CRM server(s).
- The subscribe/unsubscribe mechanism will need to be customisable.
- Event delivery will not be guaranteed.

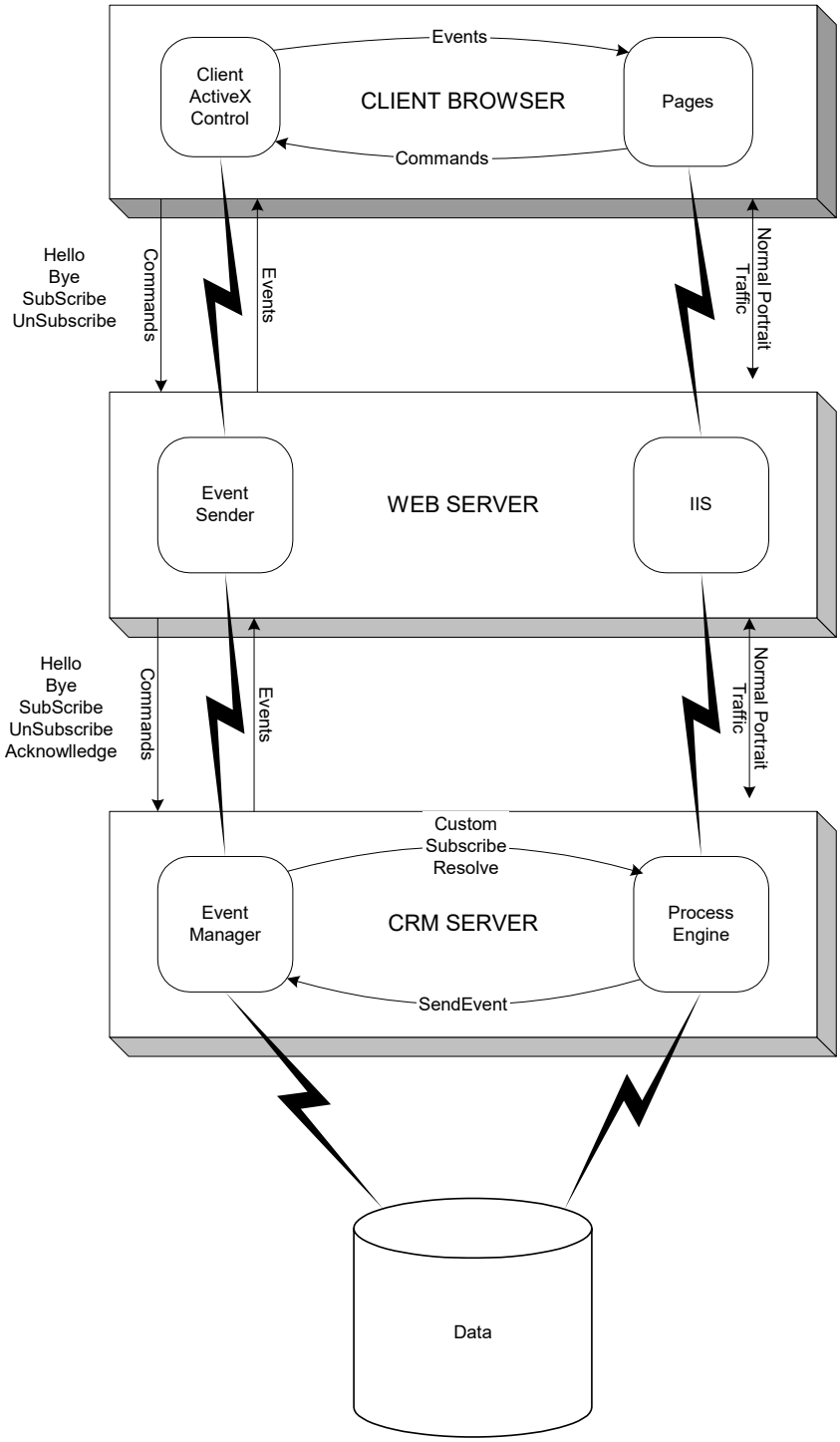
The browser controls the process and receives events through an ActiveX control that is hosted within the Browser application. The vanilla ContactCentre application achieves this by placing it within a hidden Iframe.

This control instance can then be used to send commands to Portrait Foundation or receive events from it. These events are fired as normal ActiveX Control events and are handled in JavaScript code. Project teams can implement bespoke handlers for their events or use the default mechanism (which displays a Message Box).

The control can be controlled directly from JavaScript code (to Subscribe to/Unsubscribe from events) or this can be achieved using a Script and CI node within a model.

2 Architecture

The Client Events infrastructure touches all tiers of a Portrait Installation. The following diagram illustrates this:



2.1 Client Browser

All aspects of Client Events is handled though the inclusion of an ActiveX control in the browser. Within the Client Events Architecture, this control is called an ***Event Subscriber***.

This control provides methods allowing the browser environment to control the use of Client Events:

- Initialise
- Subscribe
- Unsubscribe

In addition, received events are fired using the normal Activex Events mechanism. This allows for event handling to be programmed using one of the script technologies (usually JavaScript).

The control module also hosts a number of other COM objects that are used to abstract technical detail from the overall logic. These components are also used on the Portrait Servers. Currently, these objects are:

Message Object – encapsulates messages and its formatting.

Listener Object – encapsulates the communications technology used to listen for messages.

Sender Object – encapsulates the communications technology used to send messages.

Every instance of a client control will allocate a GUID to identify itself uniquely. This GUID is used in all communications so that the Client can verify that messages it receives are intended for it.

2.2 Web Server

The Web Server(s) host a component called the ***Event Publisher***.

This is, in fact, functionality that has been added into the existing Web Channel Service. It is completely stateless.

In essence, it listens (using an instance of the Listener object described above) for messages and processes them accordingly.

There are two types of message that can be received:

- A Command from the Client (Hello, Bye, Subscribe, Unsubscribe).
- A Command from the Event Manager (see below). (Ping, SendEvent).

In both cases, the message is effectively relayed on to the Event Manager for the former and to the actual Client ActiveX controls for the latter.

2.3 The CRM Server

The CRM Server(s) host a component called the ***Event Manager***.

This is a COM component that runs as a module within the ServiceHost architecture. It dispatches a single COM interface, which is used to control the Client Events Infrastructure.

Its clients are the Publisher (described above) and components wishing to Send Events to clients (models or processes).

All of its processing is delegated to specific implementations (implemented using .NET technology using COM Interop). All of these implementations are configured allowing projects to build custom versions should that prove necessary.

The supplied implementations use two database tables to manage client registrations (ie. Hello/Bye) and Event Subscriptions (Subscribe/Unsubscribe).

This database access is achieved using stored procedures.

2.4 Firing Events

Events are characterised by two attributes, Type and ID. In addition, they can carry a payload, which is a single string that means something only to the relevant Event Handler.

Type is the event name (a string) and the ID is some other qualifying data. For example, Type could be "Consumer" and ID the Party ID. Any one subscribing to that event would receive it whenever and Event was triggered using those attributes. In this example it could be fired whenever the Consumers details were modified. The user would then be aware that the details for a consumer they were viewing may be out of date.

Events are fired by executing a method on an instance of the Manager object. This is achieved in two main ways:

- By including the Send Event Node in a Portrait Model.
- By writing code to create a Manager instance and calling the SendEvent method.

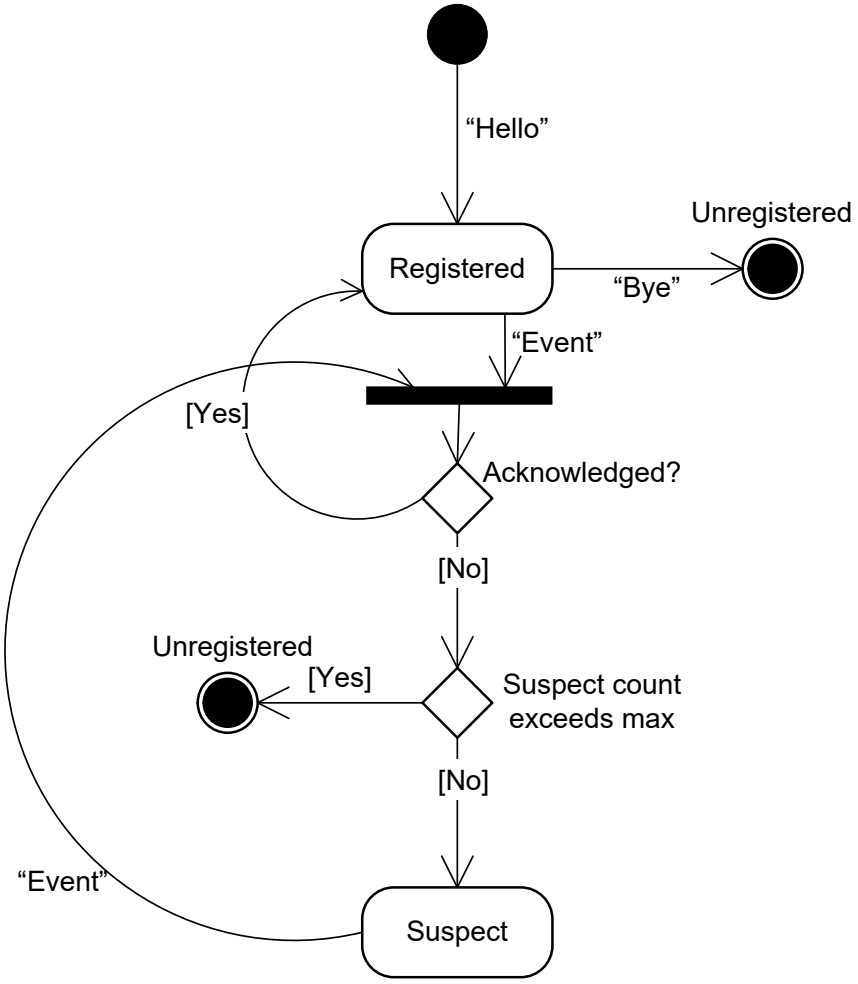
Whenever an event is triggered, the first process undertaken by the manager is to resolve the event into zero or more clients who have subscribed to that event Type/ID. The user can modify default behaviour by adding the event type as a Reference Data item in the **ClientEvents** Reference Data Group. This will have two attributes (for Subscriber and Resolver) which are the ProgIDs of objects to be used instead of the default (for that event type ONLY).

2.5 Client Lifetime Management.

This only applies to default registration processing. If that is overridden using Configuration then it only applies if the new implementation passes the call on.

Whenever a client issues a Hello command, the Registration interface is used to perform the logon. Firstly, this will check that any existing entries for that client address are valid (ie. There must be multiple browser instances on that machine), if not then any stale records are removed. In this way the database is self managing and should require no maintenance.

The following state diagram illustrates this (**as perceived by the Manager**):


"Hello"
Registered
Unregistered
"Bye"
"Event"
[Yes]
Acknowledged?
[No]
Unregistered
[Yes]
Suspect count
exceeds max
[No]
"Event"
Suspect

3 Implementation

This implementation of Client Event Delivery uses UDP as the communications protocol. This is only "known" by the Listener and Sender objects mentioned earlier.

In order to abstract address information, which is held in SysConfig, addresses are held as two strings (Address1, and Address2). This allows for any protocol to be used instead, as long as it can make use of these two attributes.

For UDP, Address1 is the Computer IP Address (or name) and Address2 is the port number to use.

3.1 Lifecycle

When the browser is loaded, it loads the ActiveX control but does nothing more with it.

As the application framework model starts up, it will run a model which will cause the control to be initialised. This is achieved by sending data (via a CI node) to the browser, which then calls the control directly. In the UDP world, the control needs to know the address where the Event Publishers have been implemented (both IP Address and Port Number). This is available in Client Events Config, extracted using a DAT and passed to the CI node via a script which reformats it into xml.

On Initialisation, the Control will issue a "Hello" command to log on. If this is successful, the client is ready to start subscribing to and receiving events.

Subscribe/Unsubscribe commands can be issued using JavaScript code calling the control directly or using the same Script node and CI node above. As such, the process can be controlled from wherever makes sense in the project implementation.

JavaScript handlers can be written to perform bespoke processing when events are received or they can use the default mechanism of displaying a Message Box. When the Browser is closed, the control is unloaded and this will automatically send a "Bye" command to logoff.

The following screenshot show the default handler displaying an event following "Amend Personal Details"