This page last changed on Jan 30, 2013 by steve.maryka.

Notification Architecture

ICEpush provides a web-based asynchronous notification mechanism in support of pushing updates from a server-based web application to a client browser connected to that application. The mechanism includes both client and server constituents, and leverages long polling over a blocking connection to deliver notifications asynchronously from the server to the client.

The core mechanism is purely for push notification, and carries no application data payload. An integration layer must be implemented around the core to provide payload processing. This integration will be specific to the client and server technologies used in the application's implementation.

The mechanism, in it's simplest form, is illustrated below with both block and sequence diagrams.

The core constituents include:

Push Client	A JavaScript implementation that is included in the page requiring push. The Push Client is responsible for managing the blocking connection, and triggering payload processing when notifications are received.
Push Service	An environment-specific library (Java, .net, PHP, ...) integrated into the web application to manage the blocking connection and listen for notification events. When notifications occur, the Push Service fulfills the blocked notification request.

Technology-specific integration constituents include:

Payload Processing Business Logic	A JavaScript callback function associated with the notification capable of retrieving the push payload, processing it, and affecting changes to the page DOM to modify the presentation.
Push Business Logic	Framework and/or application specific business logic for pushing updates. This logic must be able to trigger push notifications based on some state change in the application, and then deliver the associated payload when requested.

Core Features

In order to achieve true asynchronous notifications from the server to the client using only standard HTTP communications, it is necessary to invert the protocol and use long polling, where a connection is held open in anticipation of a notification event that will complete the request/response cycle over that connection. There are a number of ramifications associated with these long-held connections that must be addressed by the core notification mechanism.

Connection Sharing

For a browser client viewing a single push-enabled page, two connections are required - one blocking connection for the push notifications, and one non-blocking connection for all other Ajax and user-initiated requests. As you extend beyond this simplest use case to multiple browser tabs open to the same application, or multiple push-enabled applications being viewed from the same browser, it is not feasible to open multiple blocking connections for each of the page views, as this will lead to connection exhaustion at the browse (IE allows only two connections to the same domain). Under these more advanced use cases it is necessary for the core notification mechanism to perform connection sharing to prevent connection exhaustion. For a single web application, connection sharing can be achieve entirely on the client side, but as you move into multi-application and clustered deployments, server-side connection sharing also becomes necessary. ICEpush provides full-featured client-side connection sharing, and basic server-side connection management for single and multi-application deployments.

Heartbeating

Because long-held connections over the Internet can be unreliable, server-side connection heartbeating is used to keep the connection healthy.

ARP

Holding HTTP connections open at the server can have performance and scalability ramifications at the server. Typical HTTP processing maintains a thread per connections, which can lead to thread exhaustion. Asynchronous request processing can alleviate performance problems associated with blocking connections, but ARP mechanisms are typically proprietary and server-specific.

In the Java domain, the Servlet 3.0 specification unifies ARP support, and ICEpush can be configured use it.

Graceful Degradation

When multiple push applications are detected in a configuration without an EPS, the last push-enabled application page loaded will gain the blocking connection, and notifications are suspended for other application. This is sufficient for development and testing, but not suitable for Enterprise deployments.

Push Clients, Push IDs and Groups

Push Clients are simply JavaScript callback functions that are registered with the ICEpush core. Each registered callback has a unique `pushid` assigned by the core and is used in the protocol to uniquely identify that Push Client. Multiple clients can occur in a single page, or across multiple pages in a web application. Multiple browser tabs/windows can also contain multiple pages with multiple Push Clients, but the protocol maintains unique `pushids` across all those clients within a browser instance.

While ICEpush supports pushing notifications to a specific `pushid`, protocol-assigned `pushids` may not represent the best way to organize notification triggers within the application. For this reason the ICEpush protocol supports pushing to named groups. Group naming strategies are applications-specific but can be used to create notifications spokes akin to applications server scopes. Specifically:

Window Scope: Group name derived from `pushid`.
Session Scope: Group name derived from session id.
Application Scope: Fixed group name.