

Sign up for our free weekly Web Dev Newsletter.



[articles](#) [quick answers](#) [discussions](#) [features](#) [community](#) [help](#)

Search for articles, questions, tips



Articles » Web Development » Ajax » General

Follow



HTTP Push from SQL Server — Comet SQL



CoperNick, 27 Nov 2012



4.91 (55 votes)

Rate:

This article provides an example solution for presenting data in "real-time" from Microsoft SQL Server in an HTML browser. The article presents how to implement Comet functionality in ASP.NET and how to connect Comet with Query Notification from SQL Server.



Is your email address OK? You are signed up for our newsletters but your email address is either unconfirmed, or has not been reconfirmed in a long time. Please [click here to have a confirmation email sent](#) so we can confirm your email address and start sending you newsletters again. Alternatively, you can [update your subscriptions](#).

[Download source code - 45 KB](#)

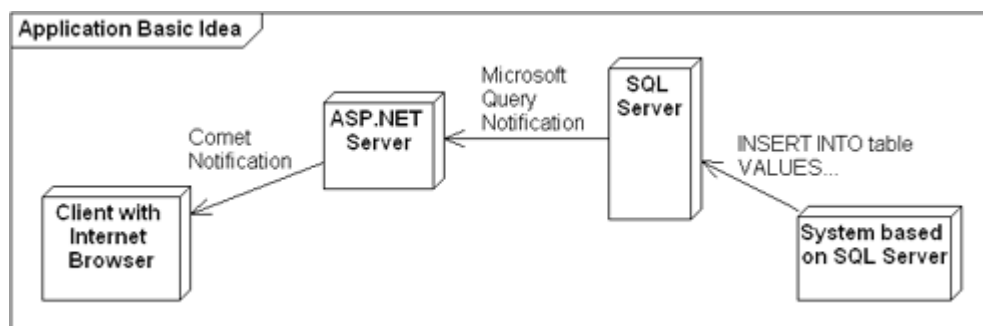
Contents

- [Introduction](#)
- [Background](#)
- [From Comet Idea to Implementation in ASP.NET](#)
- [Long Polling in ASP.NET](#)

- [Query Notification](#)
- [Results](#)

Introduction

This article presents how to implement the functionality for presenting the so-called real-time data using ASP.NET and SQL Server. This functionality is realized by implementing Comet functionality in ASP.NET and connecting this with Query Notification from SQL Server.



The code presented can also be used when adding a web interface to a legacy system based on SQL Server without changing the legacy system.

The presented code can also be used instead of periodical data refresh in existing web interfaces which use an ASP.NET AJAX Timer. After adaptation of my solution, data in the web browser will be refreshed only just after the data is updated in SQL Server. With no artificial delay. This is why I called it real-time data (not to be confused with Real-time computing).

Background

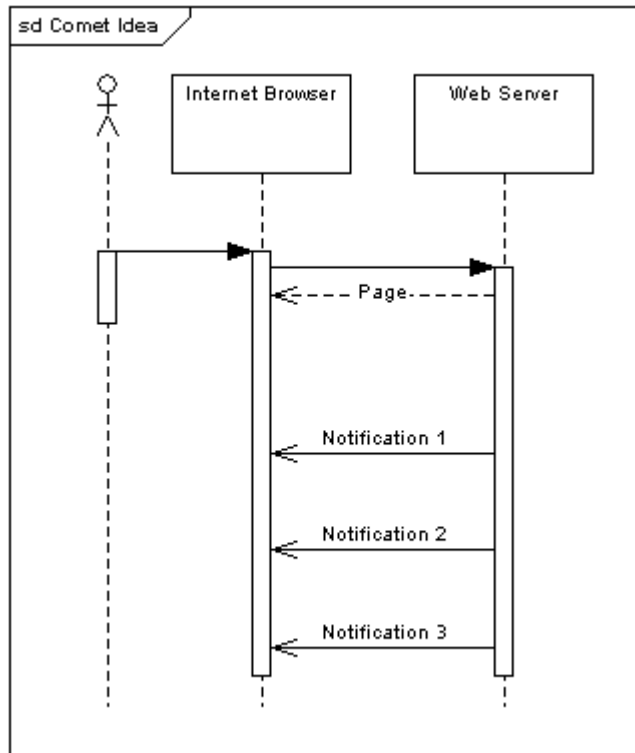
I wanted to implement a SQL Server initiated message to an Internet browser, but I wanted to do this without Java applets, without Flash, and without a Comet-dedicated server. Only JavaScript with AJAX. I also decided not to use advanced Comet message parsing on the client side. My Comet message contains only an event with simple information: "something was changed". I will later explain why I do it like this.

This article is **not** about periodical (for example, every 10 seconds) refresh of data from SQL Server or the page in a browser. In this case, the user will see updated information with a maximum delay of 10 seconds. This article is about cases when a compromise between small refresh time and server resources (CPU, RAM, network) can not be achieved. This article is about cases when a periodic refresh delay is too big, and you can not decrease this time because you will break down your server.

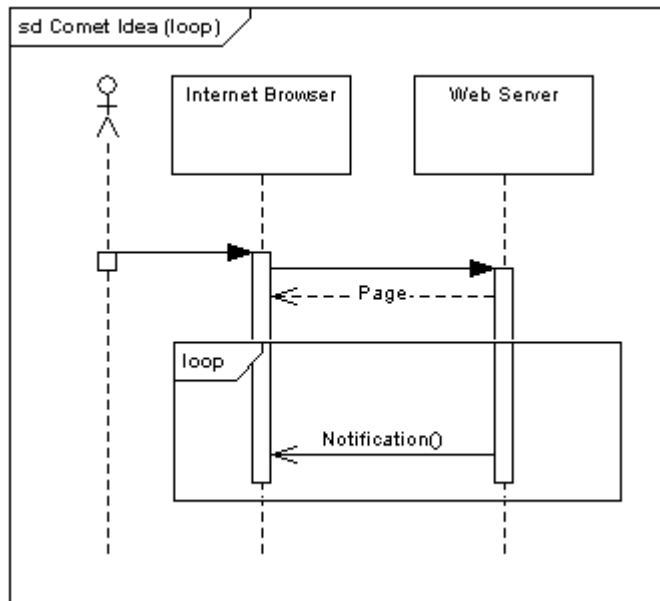
From Comet Idea to Implementation in ASP.NET

Comet Idea

Some ideas behind Comet and Comet implementation called Long Polling are described on [Wikipedia](#). I can only add some UML diagram to visualize the idea.



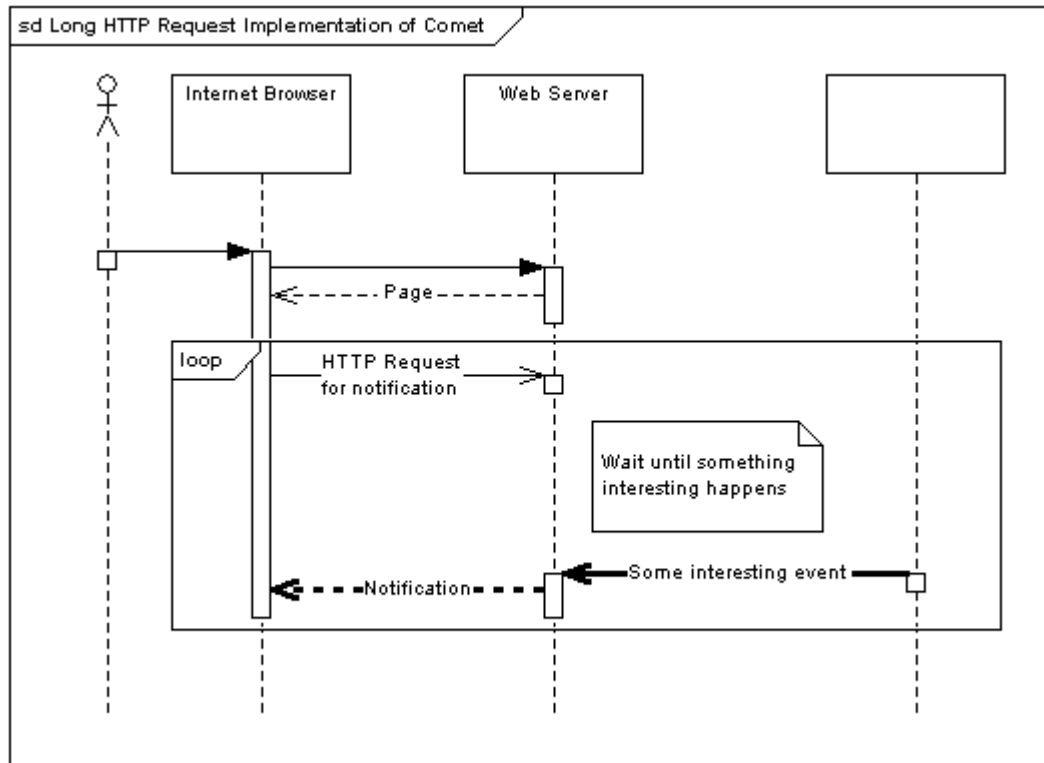
and the same using a loop:



Long-lived HTTP Request

In the previous images, you can see that the Internet browser is notified by the WWW server. Using only pure HTTP without extensions (as those proposed in HTML 5 or similar), it is not simple. Internet browsers were not designed for receiving notifications from a server. We must use some workaround to get this notification.

One of the possibilities is to make some HTTP request and wait for a response. The server will **not** return a response until some event is raised.

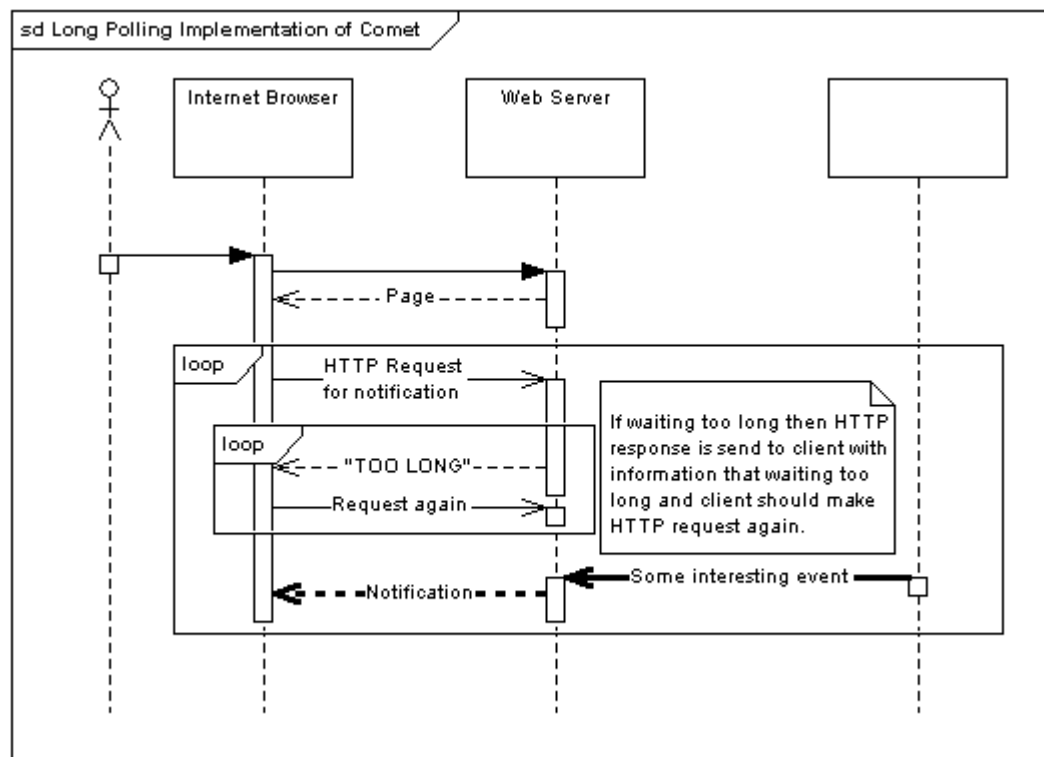


When the client receives a response ("Notification" arrow), it means that the event was raised. This event is our Comet message.

This long-lived HTTP request to the server can be very long, maybe even infinite. Implementing this without introducing some timeout is impractical: think about fighting with all possible timeouts and with a "dead" request on the server because of a broken connection. Let us set some limits on how long the request can be...

Long Polling

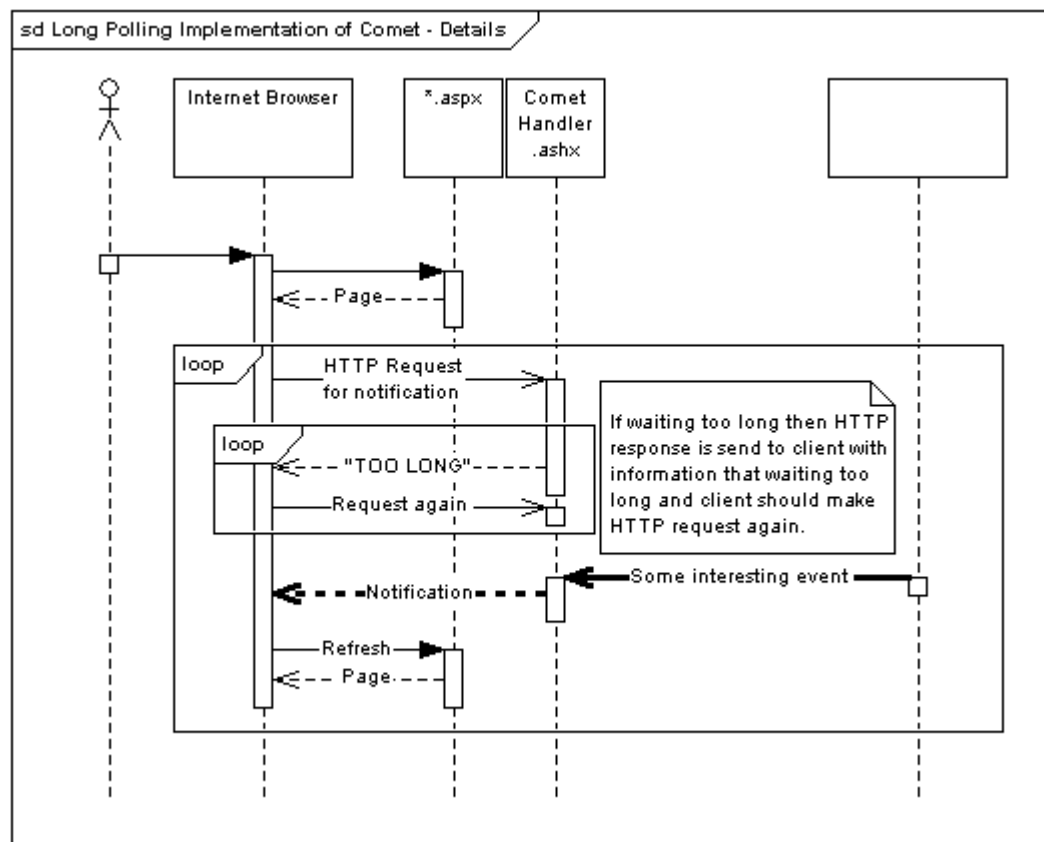
To prevent network timeouts and other problems with infinite requests, instead of infinite waiting, we can wait long but not too long. In the case of a request waiting too long, a special response is generated telling the client that there is no notification and that it must request for a notification again.



This pair of "TOO LONG" response and "Request again" is repeated in a loop periodically. It prevents timeouts, and we can call it "Comet Keep-alive". The number of keep-alives, of course, can be 0 when an event comes before the first "TOO LONG" message.

Implementation of Long Polling Comet in ASP.NET

Parsing Comet messages and then using DOM manipulation on the page can not only be hard for JavaScript beginners (like me), it also moves some logic (logic of how the page looks like) from ASP.NET to JavaScript. I recognized it as undesirable because in this case, to change or add some content to the Comet message, you must also change how the Comet JavaScript will parse this message. Because of that reason, I decided to use Comet only for a simple notification that something has changed. After receiving this message, the client will refresh data by making a postback to the current ASPX page (see the "Refresh" and "Page" arrows).



Long Polling in ASP.NET

Why not an ASP.NET AJAX Timer

ASP.NET AJAX Timer can be used for periodic page refresh (or partial page refresh). Instead of this, I decided to use **asp:Timer** to make the long-lived requests described and shown in the [Long Polling](#) paragraph. It worked fine and simple until I wanted to stop watching real-time refreshing and click some button. The postback sent by the button click was blocked. It was queued and executed after the long-lived "tick" from **asp:Timer**. Aborting the current postback (using the technique described in the "Canceling an Asynchronous Postback" article on MSDN) does not work as I needed: it "has no effect on what's going on in the server". See the "Canceling Server Tasks with ASP.NET AJAX" article on MSDN for details, or take a look at the "Canceling Async Postback" [thread](#) on the ASP.net forum. If we use Session, our postback will be queued and executed after (still running) the cancelled postback!

OK, no more bad solutions. In the next paragraphs, I will show my implementation, describing arrow by arrow from the sequence [diagram](#).

Page request - first arrow

The three first [arrows](#) are simple. The user enters the URL or clicks a link, and the page is generated with a **GridView** filled with data using **.DataBind()**, and this page is returned to the client - nothing unusual.

[Hide](#) [Copy Code](#)

```
private void RefreshData()
{
    // . . .

    int lastRecId;
    List<string> data = MessageDal.GetMessageData(out lastRecId, ...);

    // . . .

    Session["LastRecId"] = lastRecId;
    GridView1.DataSource = data;
    GridView1.DataBind();
}
```

"HTTP Request for notification" arrow

The arrow described as "[HTTP Request for Notification](#)" is implemented in JavaScript using AJAX with jQuery. Using jQuery was much simpler for me (JavaScript beginner) than using **XMLHttpRequest** directly or using the Microsoft AJAX library.

When the page is loaded, the **longPolling()** function is called.

[Hide](#) [Copy Code](#)

```
$(document).ready(function(){
    longPolling(); // Start the initial request
});
```

The **longPolling()** function makes an AJAX request to *CometAsyncHandler.ashx*, **IHttpAsyncHandler**, which simulates some kind of a page, that is calculated as a very long time. This time is specified in seconds in the request parameter (i.e., **waitTime=60**).

[Hide](#) [Copy Code](#)

```
function longPolling()
{
$.ajax({
    type: "GET",
    url: "CometAsyncHandler.ashx?waitTime=60", // one minute
    cache: false,
    success: function(data){
        isPolling--;
        if(data == "NEWDATAISAVAILABLE")
            RefreshData(); // this function is generated by
                           // using RegisterFunctionToPostBack()
        else if( data == "TOOLONG-DOITAGAIN" )
            setTimeout("longPolling()", 0 );
    }
});
```

```

        else
            addLongPollingError("error",
                "Error on server side. Received data: \"" +
                data + " \""");
    },
    error: function(XMLHttpRequest, textStatus, errorThrown){
        isPolling--;
        addLongPollingError("error",
            textStatus + " (" + errorThrown + ")");
    }
});
}

```

This request is handled on the server side by the **CometAsyncHandler** class derived from **IHttpAsyncHandler**. On the ASP.NET server side, we check if there is new data. If we have new data, then an HTTP response is immediately generated with information: **"NEWDATAISAVAILABLE"**. If there is no new data, then we register to receive Query Notifications (implemented in **WaitMessageDataAsync()**) and just wait for new data. (How the registration is made will be explained later.)

[Hide](#) [Shrink](#) ▲ [Copy Code](#)

```

public class CometAsyncHandler : IHttpAsyncHandler, IReadOnlySessionState
{
    public static List<CometAsyncResult> AllWaitingClients =
        new List<CometAsyncResult>();
    public static object AllWaitingClientsSync = new object();
    private static bool threadForTimeoutsWorking = false;

    // . . .
    // . . .
    // . . .

    public IAsyncResult BeginProcessRequest(HttpContext context,
        AsyncCallback cb, object extraData)
    {
        context.Response.ContentType = "text/plain";

        // Get wait time from request
        int waitTime;
        ParseRequest(context.Request, out waitTime);

        // Get last seen record ID from Session
        int lastRecId = (int)context.Session["LastRecId"];

        // . . .

        CometAsyncResult result = new CometAsyncResult(
            context, cb, waitTime, lastRecId);
        lock (AllWaitingClientsSync)
        {
            // register to Query Notification or complete
            // request synchronously in case if there is
            // already new data:
            if (!MessageDal.WaitMessageDataAsync(lastRecId))

```



```

    {
        // if not waiting (there is new data)
        // result is to be completed synchronously
        result.IsCompleted = true;
        result.CompletedSynchronously = true;
        result.Result = true; // new data is available
        WriteResponseToClient(result);
        return result;
    }
    else
    {
        // asynchronous (normal case):
        AllWaitingClients.Add(result);

        if (AllWaitingClients.Count == 1)
            StartClientTimeout();
    }
}
return result;
}

// . . .
// . . .
// . . .
}

```

"TOO LONG" response

To prevent very long waiting (or infinite waiting), we create a "while" thread that checks all waiting (not responded) clients whether they are waiting too long. If a given client is waiting too long, it is removed from the list and the **Callback()** associated with the client is called. This callback is the **AsyncCallback cb** parameter from the **BeginProcessRequest()** method.

Following is a part of **StartClientTimeout()** (modified for presentation and contains only the main idea):

Hide Copy Code

```

while( AllWaitingClients.Count > 0)
{
    // Call Callback() to all timeouted requests and
    // remove from list.
    lock (AllWaitingClientsSync)
    {
        DateTime now = DateTime.Now;
        AllWaitingClients.RemoveAll(
            delegate(CometAsyncResult asyncResult)
            {
                if (asyncResult.StartTime.Add(asyncResult.WaitTime) < now)
                {
                    asyncResult.Result = false; // timeout
                }
            }
        );
    }
}

```

```

        asyncResult.Callback(asyncResult);
        return true; // true for remove from list
    }

    return false; // not remove (because not timed out)
});
}

// This sleep causes that some timeouted clients are removed with delay
// Example: if timeout=60s, sleep=1s then timeouted client can be removed after 60,7s.
// In some cases this can be considered as bug. TODO: Change it to WaitOne() and
// calculate proper sleep time.
Thread.Sleep(1000);
}

```

After calling **Callback()** (which is the same as the **AsyncCallback cb** parameter from the **BeginProcessRequest()** method), the **EndProcessRequest()** method is called by the ASP.NET framework. In this method, we have a chance to finish generating the HTTP response.

[Hide](#) [Copy Code](#)

```

public void EndProcessRequest(IAsyncResult result)
{
    WriteResponseToClient((CometAsyncResult) result);
}

public void WriteResponseToClient(
    CometAsyncResult cometAsyncResult)
{
    if (cometAsyncResult.Result)
        cometAsyncResult.Context.Response.Write(
            "NEWDATAISAVAILABLE");
    else
        cometAsyncResult.Context.Response.Write(
            "TOOLONG-DOITAGAIN"); // timeout - client must make request again
}

```

So to each timed out client (time out thread sets its result to **false**), a **"TOOLONG-DOITAGAIN"** response is returned. This response is handled by the JavaScript code [fragment](#) that made the AJAX/Comet request.

[Hide](#) [Copy Code](#)

```

// . . . part of <a href="#LongPollingFunction%22">LongPolling()</a> function
else if( data == "TOOLONG-DOITAGAIN" )
    setTimeout("longPolling()", 0 );
// <a href="#LongPollingFunction%22">. . .</a>

```

"Request again" arrow

The code above will cause that, after the "too long" message, the current function will be called again. This will cause the client to make the "[HTTP Request for notification](#)" once again.

"Notification" arrow

When a Query Notification comes from SQL Server to the ASP.NET server (see bold [arrow](#)), the `ProcessAllWaitingClients()` method is called. This method will iterate through the waiting clients list, setting the `Result` fields to `true` and calling the callback (passed earlier as a parameter to the `BeginProcessRequest()` method).

[Hide](#) [Copy Code](#)

```
public static void ProcessAllWaitingClients()
{
    // . . .
    foreach (CometAsyncResult asyncResult in AllWaitingClients)
    {
        asyncResult.Result = true; // New data available
        asyncResult.Callback(asyncResult);
    }
    AllWaitingClients.Clear();
    // . . .
}
```

The callback will execute `EndProcessRequest()` in the same way as in the case of the timed out thread. The difference lies in the fact that `Result` is set to `true` in this case. So during HTTP response generation, "NEWDATAISAVAILABLE" is written.

This response is handled by the same JavaScript code [fragment](#) that made the AJAX/Comet request.

[Hide](#) [Copy Code](#)

```
// . . . part of <a href="#LongPollingFunction%22">LongPolling()</a> function
if(data == "NEWDATAISAVAILABLE")
    RefreshData(); // this function is generated
                  // by using RegisterFunctionToPostBack()
// <a href="#LongPollingFunction%22">. . .</a>
```

In this case, the `longPolling()` function is not executed again, so the long polling loop is not stopped. Instead of complicated data, we only have information about new data.

Page Refresh

After receiving the Comet message, we make a partial AJAX refresh by sending a postback to `asp:UpdatePanel1` (`UpdatePanel1`).

[Hide](#) [Copy Code](#)

```
function RefreshData()
{
    __doPostBack('UpdatePanel1','')
}
```

This function is generated by the **RegisterFunctionToPostBack()** method.

[Hide](#) [Copy Code](#)

```
// I decided to generate JavaScript Refresh() function, but you can
// write it by yourself and include in "LongPolling.js"
//
// Thanks to:
// http://geekswithblogs.net/mnf/articles/102574.aspx
// http://www.xefteri.com/articles/show.cfm?id=18 How postback works in ASP.NET
// and thanks to Dave Ward hint for calling __doPostBack("UpdatePanel1", "") ,
public bool RegisterFunctionToPostBack(string sFunctionName, Control ctrl)
{
    // call the postback function with the right ID
    // __doPostBack("'" + UniqueIDWithDollars(ctrl) + @"', '');
    string js = "    function " + sFunctionName + @"()
    {
        " + ctrl.Page.ClientScript.GetPostBackEventReference(ctrl, "") + @"
    }";
    ctrl.Page.ClientScript.RegisterStartupScript(this.GetType(), sFunctionName, js, true);
    return true;
}
```

So instead of writing a parser for the Comet message in JavaScript and making DOM operations on the page, we just trigger the ASP.NET engine for a partial refresh of the page.

Query Notification

In this part of the article, I will try to show how to trigger Comet events using a Query Notification from SQL Server.

SqlDependency Class

To receive "Query Notifications", we can use the **SqlDependency** class. In the MSDN documentation of **SqlDependency**, you can read that you need to associate a **SqlDependency** object to the **SqlCommand** object and subscribe to the **OnChange** event. Then you must guess, that after these steps, you must execute this command. When executing the command, you will get some data. The **OnChange** event is raised when data from the command changes.

Table

In our case, we are interested in new rows from the table **TestTable**. Obviously, notifications can be received about any kind of update.

[Hide](#) [Copy Code](#)

```
CREATE TABLE [dbo].[TestTable](
    [RecId] [int] IDENTITY(1,1) NOT NULL,
    [Text] [nvarchar](400) NULL,
    [Time] [datetime] NOT NULL CONSTRAINT [DF_TestTable_Time] DEFAULT (getdate()),
```

```

CONSTRAINT [PK_TestTable] PRIMARY KEY CLUSTERED ( [RecId] ASC )
WITH (
    PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
) ON [PRIMARY]

```

We can insert data in this table using a simple **INSERT**.

[Hide](#) [Copy Code](#)

```

INSERT INTO TestTable (Text)
VALUES(N'Hello World!')

```

Step 1 - Check if we need to wait for changes

When the ASP.NET server is asked by the browser for notification about new data, the ASP.NET server checks if there is new data. If so, the browser will receive notification about new data without using a "Query Notification".

In our case, we are watching only insertions, so the query is very simple. We just check **MAX(RecId)**.

[Hide](#) [Copy Code](#)

```

// Query for making decision whether data was changed or we must wait.
// In our case we are interested in new records so we select MAX.
private const string queryForCheck = @"
SELECT MAX(RecId)
FROM dbo.TestTable";

// . . .

// 1. First query, to check if we need to wait for changes
using (SqlCommand cmd = new SqlCommand(queryForCheck, conn))
{
    int max = Convert.ToInt32(cmd.ExecuteScalar());
    if (max > lastRecId) // if max > last seen recId
        return false; // No async! New data available right now!
}

```

Step 2 - Run dependency

When there is no new data, we create and setup a new **SqlDependency**, associate it with a **SqlCommand**, and execute the command.

[Hide](#) [Shrink](#) [Copy Code](#)

```

// This query follows rules of creating query for "Query Notification"
// and is filtered by record ID, because (in this case) we expect only
// "INSERT" of new records. We are not observing old records. To be
// compatible with Query Notification we must use schema name ("dbo"

// in our case) before table! For other compatibility issues you must
// search MSDN for "Creating a Query for Notification" or go to
// http://msdn.microsoft.com/en-us/library/ms181122.aspx
// And don't Look at this: (old and incomplete list):
// "Special Considerations When Using Query Notifications" at
// http://msdn.microsoft.com/en-us/library/aewzkxxh%28VS.80%29.aspx
private const string queryForNotification = @"
SELECT RecId
FROM dbo.TestTable
WHERE RecID > @recId";

// . . .

// 2. Second query, to run dependency
SqlDataReader reader;
using (SqlCommand qnCmd = new SqlCommand(queryForNotification, conn))
{
    qnCmd.Parameters.AddWithValue("@recId", lastRecId);

    // Setup dependency which will be used to wait for changes
    depend = new SqlDependency(qnCmd);
    depend.OnChange += Depend_OnChangeAsync; // callback 1

    // Execute second query to run dependency (Query Notif.),
    // and to get content of our table data, that will be used
    // by Query Notification as a starting point for observing
    // changes.
    reader = qnCmd.ExecuteReader();
}

```

Step 3 - Handle rare cases

When executing a command to receive notification, it can be too late. Just before execution, data can be changed (inserted in our case) and we will not receive a notification. To prevent inserting new data between "Step 1" and "Step 2", you can put them in a transaction. This will block insertion of new data. I prefer to avoid blocking table in transaction in this case, because we can simply check if new data was inserted between those two steps.

[Hide](#) [Copy Code](#)

```

// 3. Make sure that nothing has changed between point 1. and point 2.
// (just before firing query notification)
bool newData = reader.HasRows;
reader.Close();
if (newData)
{
    // very rare case - data changed before

```

```

// firing Query Notif. (SqlDependency)
// TODO: test this case by making some Sleep() between 1. and 2.

// We have new data and we decide not to receive notification:
depend.OnChange -= Depend_OnChangeAsync;
depend = null;

return false; // No async! New data available right now!
}

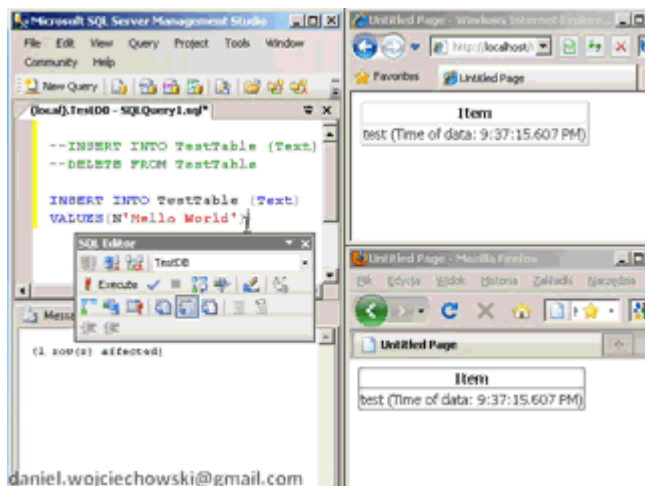
```

Receive notification

When there is no new data, we successfully register to "Query Notification". When somebody or something inserts data to **TestTable**, then our **Depend_OnChangeAsync()** will be called. This method will call our **ProcessAllWaitingClients()** (discussed previously) which will deliver notification to clients.

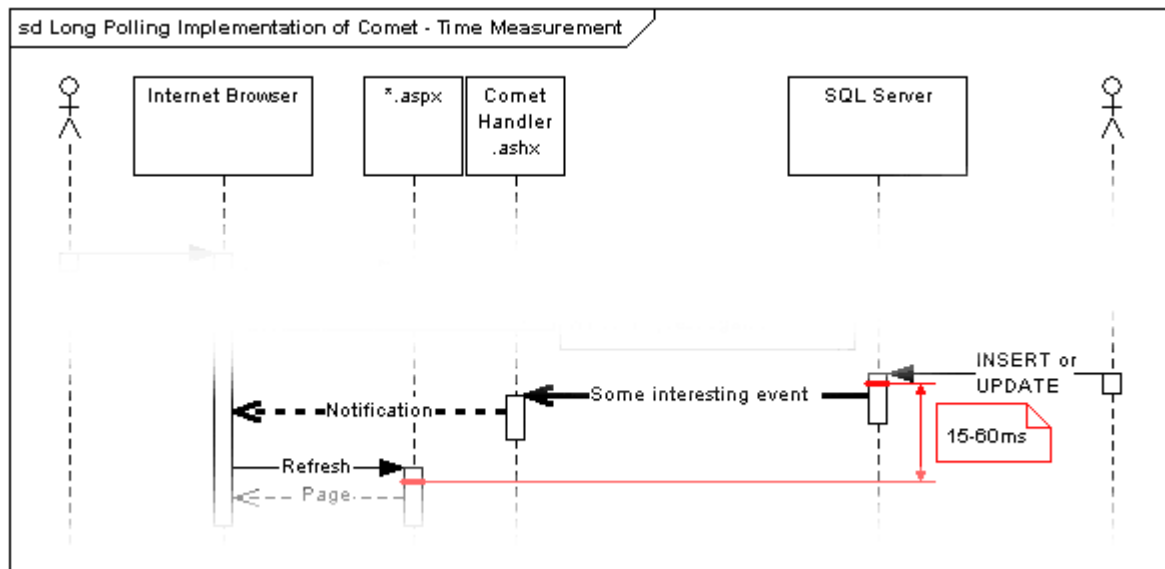
Results

Precise time measurement is not important here. The most important thing is that time is not related to the polling interval, because there is no polling interval (as in a constant polling solution). If you buy a faster server, you will be faster. But let's make a time measurement just for fun.



Click [here](#) or [here](#) to enlarge. Press Esc key to stop GIF animation.

Time measurement is started using a SQL **INSERT (DEFAULT (getdate()))** with an accuracy of 3.33 milliseconds). After that, a query notification is fired, and ASP.NET is notified. Then, a Comet notification is sent to the browser. The browser makes a refresh call. The server receives the refresh call. On the server side, time is measured again. This time, **DateTime.Now** (with an accuracy of 15 milliseconds) is used to measure time. The time measured is usually from 15 to 60 milliseconds on localhost (Intel Core 2 Duo 2.13GHz, 3 GB RAM).



Why did I do the measurement on localhost? Because I'm not interested in the network speed, I am only interested in the speed of my code. If you wish to calculate the speed on your network, add some "ping" or "pong" speed to each arrow from the diagram.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Share

[TWITTER](#)[FACEBOOK](#)

About the Author

CoperNick

Software Developer

Poland 

Follow
this Member



I graduated from the [Jagiellonian University](#). After defence of master thesis ([link to presentation](#)) I have started professional programming in C and C++ using OpenGL. After that, my Company gave me a chance to switch to .Net framework and C# and I have betrayed my passion for computer graphics. Now I'm C#, SQL and sometimes ASP.NET programmer.

You may also be interested in...



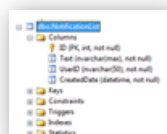
Push Messages in RESTful WCF Web Application with Comet Approach



Convolutional Neural Networks and Their Visualization Using Keras



ASP.NET and Comet: Bringing Sockets Back



Real Time Notifications using SignalR and SQL Dependency



VRC++ Object Oriented Scripting Language :: Vincent Radio {Adrix.NT}

	ID	TID	Name
1	1	1	Ashish
2	2	3	Rakesh
3	3	2	Himangal

CTE In SQL Server

Comments and Discussions

[Add a Comment or Question](#)[Email Alerts](#)

Spacing

Relaxed ▼

Layout

Open All ▼

Per page

25 ▼

[Update](#)[First](#) [Prev](#) [Next](#)

My vote of 5

hongseokwon

24-Feb-16 2:11



Good Push Solution.
Thanks you.

[Reply](#) · [Email](#) · [View Thread](#)

Just what I was looking for after hours and hours

Member 9411300

28-Dec-15 1:16



Love this. Perfect for me. Thank you SOOOO Much!

[Reply](#) · [Email](#) · [View Thread](#)

Hmm... not able to run

saberbladez

1-Jul-15 5:12



Hi I can't seem to compile the program after unzipping..

[Reply](#) · [Email](#) · [View Thread](#)

Error:"Connect Again"

asif rehman baber

14-May-15 22:17



Dear Expert...thanks for your great article...it is working fine when i just ran your code. However when i just wanted to incorporate this in my project it throws an error like "Error Undefined" with a button having text "Connect Again..."

can you guide me to get rid of this issue? the above error is written in LongPooling.js file.

[Reply](#) · [Email](#) · [View Thread](#)

Nice work

Member 11685060

12-May-15 19:52



good work sir I appereciate it.
<http://onlinepctuts.blogspot.com/>

[Reply](#) · [View Thread](#)

VB version

 burhanville

18-Jul-14 6:18



Can anybody provide me VB version of this comet sql project

[Reply](#) · [Email](#) · [View Thread](#)

What is the difference between SignalR & Comet library

 Tridip Bhattacharjee

9-Jul-14 11:54



signalr & comet both used to push message from server side to client side. so i like to know the what is the difference between two. if anyone know then please share the knowledge.

which one is better in terms of performance & flexibility?

looking for good discussion. thanks

tbhattacharjee

[Reply](#) · [Email](#) · [View Thread](#)

Re: What is the difference between SignalR & Comet library

 CoperNick


10-Jul-14 17:59



As I understand the SignalR is a Comet library. SignalR is a Comet library in the same way like C# is programming language. So you can't find differences.

[Reply](#) · [Email](#) · [View Thread](#)

Re: What is the difference between SignalR & Comet library

 Tridip Bhattacharjee

11-Jul-14 10:21



which one has maximum flexibility? also tell me why u use comet not signalr? thanks



tbhattacharjee

[Reply](#) · [Email](#) · [View Thread](#)

Re: What is the difference between SignalR & Comet library

 CoperNick

16-Jul-14 11:24



This is how I understand this:



SignalR Is a library, it is some ready to use solution and has some Comet implementation included.

Comet is some general term, so you must implement Comet or download some Comet library

Comet library is some library that you can download (or even create by yourself). Examples of comet libraries: [nodejs.org\[^\]](#), [ape-project.org\[^\]](#), [Portal\[^\]](#), [SOCKET.IO](#), [dojotoolkit.org\[^\]](#), [SignalR\[^\]](#)

Similar question on Stack Overflow: [What is the difference between Comet Technology and SignalR\[^\]](#)

[Reply](#) · [Email](#) · [View Thread](#)



SignalR works, but...



Dewey

19-Jun-14 6:58



I can also see the beauty in your low overhead solution!

Good work, have a 5!

[Reply](#) · [Email](#) · [View Thread](#)



Sometimes its not working



Arul5

30-Jan-14 13:07



Hi,

Whether this code Supports live chat with multi users at a time..

modified 30-Jan-14 6:27am.

[Reply](#) · [Email](#) · [View Thread](#)



Not working properly in my web application.



Member 10478313

20-Jan-14 11:57



I have downloaded your code and it worked completely i.e both auto notifications and on button_click event when i run it. But when i integrated it in my web application only button_click event works and auto notifications when data is inserted into table is not working. Can you please help me out. Thank you in advance.

[Reply](#) · [Email](#) · [View Thread](#)



Re: Not working properly in my web application.



CoperNick

20-Jan-14 12:14

Mayby the structure of your web pages is different. Mayby you should add some subfolder in this line:



url: "CometAsyncHandler.ashx?waitTime=60"

[Reply](#) · [Email](#) · [View Thread](#)



Re: Not working properly in my web application.

Member 10478313

20-Jan-14 12:36

It shows url: "CometAsyncHandler.ashx?waitTime=10". Also, i have used datatables to fetch result from my sql queries rather than the lists which you have used. I am new to this platform so can you please elaborate more on "Mayby you should add some subfolder in this line:" and what should be the ideal structure of the web pages for this thing to workout?

[Reply](#) · [Email](#) · [View Thread](#)



Re: Not working properly in my web application.

CoperNick

20-Jan-14 14:12

I'm talking about something like this: "Folder/SubFolder/CometAsyncHandler.ashx?waitTime=10". But after your mail it looks like a problem with query: <http://msdn.microsoft.com/en-us/library/ms181122.aspx>

"[...] ideal structure of the web pages for this thing to workout?" Ideal structure?
I don't know.

[Reply](#) · [Email](#) · [View Thread](#)



Thank you

Member 10478313

8-Jan-14 14:34

Your code helped me to solve my problem. Thank you. Cheers 😊

[Reply](#) · [Email](#) · [View Thread](#)



Mysql ?

Member 10479155

20-Dec-13 18:07

how this project using for mysql?

[Reply](#) · [Email](#) · [View Thread](#)



performance issue

LoveJenny

31-May-13 5:12

if you use SQL profile , you will find out what was happend in detail.

▲ SQL server will always run "select xxx from xxx" and this is how SQLDependency class work.



recommend to use trigger or other notification server like MSMQ.

[Reply](#) · [Email](#) · [View Thread](#)



Re: performance issue

CoperNick

31-May-13 10:19



Query Notification is not implemented as SELECT in loop.



Notifications are started by executing query command so this is true that "select xxx from xxx" will be executed "always", but SELECT will not be executed in loop but once. It is something opposite to polling - It is using SQL Server Service Broker, to get notifications. I get the notification (through all application layers) to web browser in 60ms (max). Query Notifications it is not implemented as loop. If it would be implemented as a loop then that SELECT query would have to be fired at least 20 times per second in my case - did you observe 20 SELECTs per second?

[Reply](#) · [Email](#) · [View Thread](#)



Implementing the same without using web pages

nicksaily

27-Mar-13 13:34



Hi,

I would like to know, how to implementing the same without using web pages like (html, aspx, php etc) and a request from a tool like REST Client.

Problem requirement I need to achieve is from messagedal file when there is no new records, the server has to wait for the changes and and returns the result. But in my case it is not happening and it immediately return if no record changes.

Please give a solution or guidance to achieve my requirements.

Thanks in advance...

[Reply](#) · [Email](#) · [View Thread](#)



nice work

prem parihar

2-Feb-13 17:56



nice work. I really need it. thanx

[Reply](#) · [Email](#) · [View Thread](#)

5.00/5 (1 vote)



My vote of 5

Member 10510822

30-Nov-12 16:27

 excellent

[Reply](#) · [Email](#) · [View Thread](#)


Nice

 **Vitaly Obukhov**

30-Nov-12 13:15



I had the similar implementation of this solution.

One day client have asked me to develop dashboard page by which customers could track their requests within short delays.

We have discarded solution with classical request polling at once.

I have decided to implement my own long-polling solution since I didn't knew about sql server notifications feature yet or similar frameworks. Also we were running short of time 😊

Basically we had asp.net mvc 3 & sql server 2005 on win 2003 R2 server and jquery for client side. So the workflow was following:

1. Customer opens page which contains client side long-polling logic implementation via jquery ajax & timers.
2. JS script sends request to separated mvc controller which has session state marked as "read only" to disable locking of another customer's requests.
3. Controller's action finds customer session in container and locks workflow via manualreseteventslim.
4. 3rd party app (from another server) modifies special table's data on sql server.
5. CLR trigger on this table calls another one controller.
6. This controller's action lookups session container using id data from trigger and unblocks workflow via manualreseteventslim.
7. First controller's action checks reason of unlocking (timeout or data modification) and responses to client with related data (if exists).
8. Finally client side JS script checks response reason code, handles data in related way and sends request to first controller again.

Well, there were a lot of tricky places in details but this worked just fine 😊

After this I have read about SignalR 😊

[Reply](#) · [View Thread](#)


My 5

 **Epavesi**

28-Nov-12 12:38



My 5

[Reply](#) · [Email](#) · [View Thread](#)


Last Visit: 18-Dec-18 1:37 Last Update: 19-Dec-18 0:14

[Refresh](#)

1 2 3 Next »

 General
  News
  Suggestion
  Question
  Bug
  Answer
  Joke
  Praise
  Rant
  Admin

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Cookies](#) | [Terms of Use](#) | Mobile
 Web02 | 2.8.181215.1 | Last Updated 27 Nov 2012

▼ | تحديد اللغة

Layout: [fixed](#) | [fluid](#)

Article Copyright 2010 by CoperNick
 Everything else Copyright © [CodeProject](#), 1999-2018