HTTP Data Volume Considerations

This chapter discusses the data volume that is transferred between the server and the browser client at runtime. You should read this chapter carefully when deploying your Application Designer based application for productive usage - especially in scenarios in which limited bandwidth is available.

Application Designer is optimally designed to serve these kinds of applications, but you have to know and understand how a browser requests information from a server in order to configure your deployment in an optimal way.

The following topics are covered:

- The Browser’s Loading Behavior
- How does Application Designer Fit in?
- Configuration Options for Optimized HTTP Usage

The Browser’s Loading Behavior

Browsers load data from the server by specifying a URL and getting back some content (for example, HTML text). This happens, for example, for any static files that are loaded:

- pages (HTML),
- images (GIF, JPG, etc.),
- other resources.

In order to reduce the volume of data exchanged, the browser uses caching. Once the content for a specific URL is loaded, the browser will keep the result in its cache, together with a timestamp that is passed as header information of the loaded content.

How does caching work (now assuming that you use the standard browser configuration)?

- Once the content is loaded into the browser, the browser will not ask again for the content in the browser session, i.e. if you keep your browser up and running, it will use the cached content.
- But if you close the browser and reopen it again, the browser will check whether the cached content is still up to date. It will send a URL to the server passing the timestamp of the cached content as part of the request. The server will check whether the timestamp of the cached content is valid - if so, it only sends back a short "OK" message to the browser, but does not include the content. If the timestamp is outdated, the server will send back the now up to date content.

You see: switching cache on (as it is in the standard configuration) still does mean that there is communication between the browser and the client when starting a new browser. The number of URLs that are sent as requests to the server is still the same - just the responses are smaller.

Imagine a page that internally holds 10 images. If not yet cached, there will be 11 requests towards the server, and 11 responses, each response containing the content (1 HTML page, 11 image byte-streams). Now closing the browser and reopening the page, you will see that again 11 requests are sent to the server.

This time, the server will not send back the content, but just 11 acknowledgement messages that tell the browser that its cached content is still valid.

When improving the data transfer between browser and server, both aspects do play a significant role:

- The number of exchanged requests.
- The number of exchanged bytes.

Note:
A request over satellite-based networks may have a roundtrip time of about 500 ms - even if very few pieces of information are actually exchanged.

How does Application Designer Fit in?

Application Designer pages are kept stable inside the browser. They are not permanently reloaded but exchange their data content with the server. The data content is exchanged in delta mode, i.e. only changed data properties are exchanged.

Result: once pages are loaded, Application Designer is "unbeatably" fast (compared to traditional web frameworks in which pages are constantly resent to the client) because of the clear separation of page loading and data loading. A data roundtrip (which is internally zipped) has a size of 1 to 3 kBytes typically (depending on the page).

What is the price? Application Designer pages contain JavaScript statements and reference to JavaScript libraries (that are shared cross pages, of course), i.e. pages are by nature bigger than plain HTML pages. Of course, most of the code is shared across different pages (by using libraries). However, loading an initial page without any information cached in the browser means:

1. loading the page itself (e.g. 50 kBytes)
2. loading libraries (e.g. 150 kBytes)
3. loading images (e.g. 30 kBytes)

   and then finally:

4. loading page data (1-3 kBytes)

Only step 4 contains dynamic data. All other steps are requests against static content.

Consequence: you have to explicitly keep an eye on steps 1 to 3. From a communication perspective, they are the biggest parts of the initial load, but afterwards need not be reloaded into the browser because they contain static content.

Configuration Options for Optimized HTTP Usage

There are two ways to optimize the HTTP communication:

1. Use compression for all textual information.

2. Make use of browser cache and avoid roundtrips.

For both purposes, Application Designer offers so-called servlet filters. Filters are small programs that you can configure to wrap HTTP requests coming into a server. In the web application (via the *web.xml* file), you configure which filters are used in which way for which requests.

The two Application Designer filters are:

- **com.softwareag.cis.server.filter.CompressionFilter**
  A filter that zips content that is loaded from the web application.

- **com.softwareag.cis.server.filter.HttpHeaderFilter**
  A filter that sets HTTP header attributes for requests. One of the most important attributes that will be used here is the expiration date of content.

Both filters and the way to use them are explained below:

- Filter for Compressing Content

- Filer for Setting HTTP Header Attributes

Filter for Compressing Content

Have a look at the following example configuration (*/WEB-INF/web.xml* file):

```
<web-app id="cis">
    ...
    ...
    ...
    <filter>
        <filter-name>Compress</filter-name>
        <filter-class>com.softwareag.cis.server.filter.CompressionFilter</filter-class>
    </filter>
    ...
    ...
    ...

    <filter-mapping>
        <filter-name>Compress</filter-name>
        <url-pattern>*.html</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>Compress</filter-name>
        <url-pattern>*.bmp</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>Compress</filter-name>
        <url-pattern>*.js</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>Compress</filter-name>
        <url-pattern>/servlet/StartCISPage*</url-pattern>
    </filter-mapping>
    ...
    ...
    ...
```

```
    <servlet id="Connector">
    ...
    ...
    ...
```

First of all, the filter is defined in the `<filter>...</filter>` section. Then, URL patterns are defined for which the filter is to be used. In the example, the compression filter compresses any requests that are HTML pages, non-compressed images (bitmaps), JavaScript library files and calls against the Application Designer servlet to start new Application Designer pages.

The example represents a reasonable filter configuration for running Application Designer: uncompressed content (text, bitmaps) is filtered, i.e. is zipped; compressed content (e.g. JPG images, PDF files) are passed through without being compressed again.

Of course, compression has its price on the server side. A compressed request requires some time for executing the compression, but it is typically worth it.

Filer for Setting HTTP Header Attributes

Each request to which a server responds holds a certain number of HTTP header attributes that surround the content of the response. One important HTTP header parameter is the expiration date of a page.

Let us come back to the caching discussion previously in this section. A browser holds cached content associated with the timestamp of loading. If closing and reopening a browser, it will still send content requests to the server to check whether its content is valid, and will receive acknowledgement messages or refreshed content.

When now using the header attribute `max-age`, you can tell the browser that a content has a certain stability: for example, you tell the browser that when loading a page today, it will not expire in the next 24 hours. When now being closed and reopened, the browser will only send HTTP requests to the server that are expired. Now, with defining expiration dates, the browser will really reduce the number of URLs that are sent to the server when being reopened.

```
...
...
...
<filter>
    <filter-name>HttpHeader</filter-name>
    <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
    <init-param>
                <param-name>Cache-Control</param-name>
                <param-value>max-age=86400</param-value>
    </init-param>
</filter>
...
...
...
<filter-mapping>
    <filter-name>HttpHeader</filter-name>
    <url-pattern>*.gif</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>HttpHeader</filter-name>
    <url-pattern>*.jpg</url-pattern>
```

```
</filter-mapping>
...
...
...
```

In the filter definition, you see that a header attribute `max-age` is set for filtered HTTP responses. The value is set to 86400 seconds (this is 24 times 3600s - one day). The filter is applied to GIF files and JPG files.

Now it depends on some environment conditions how to configure the filter. Ad hoc candidates for filtering are:

- All graphic files. Typically they do not change too frequently and they do not have a direct impact on the consistency of an application.
- Static content, e.g. PDF documents.

Of course you might also think about filtering other files such as:

- JavaScript libraries.
- Generated Application Designer HTML pages.

But be aware of the following: if you upgrade your server software (for example, you modify some of your pages), you will have to explicitly tell your users to clear their cache. Otherwise, they will keep their old pages until they are expired.

The Application Designer recommendation for long term caching via the expiration date is:

- You must use it for any static images.
- You should be very careful with any content-related parts of your application.

Be aware of the fact that you can define the filter several times inside your *web.xml* definition:

```
...
...
...
<filter>
    <filter-name>HttpHeaderLongTerm</filter-name>
    <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
    <init-param>
                <param-name>Cache-Control</param-name>
                <param-value>max-age=86400</param-value>
    </init-param>
</filter>
<filter>
    <filter-name>HttpHeaderShortTerm</filter-name>
    <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
    <init-param>
                <param-name>Cache-Control</param-name>
                <param-value>max-age=3600</param-value>
    </init-param>
</filter>
```