Using the Resource Database to Adapt to Session Timeouts

G J Boris Allan, Joseph Allen and Siu Wan, Westat

1. Abstract

If an interview session times out, an obvious method of recovery is to use an error page to trap the error and then launch a call to a recovery URL. Unfortunately, because we have lost contact with the session data, it is problematic to recover the case ID of the interview we just lost (State.KeyValue is null). Also we have to hardcode the value of the recovery URL in the error page because any value stored in the survey database is unavailable as we have lost contact with the session data. Using our resource-database methodology developed for web security we can handle session timeouts and do not have to hardcode URLs – we can modify the URL without changing the package, because we just change the preloaded data.

2. Introduction

There are three common ways to stop and possibly restart any web page:

1. Users have had enough, and there is no Exit feature available on the page[1] so they close the tab by selecting X top right on the browser or selecting X on the relevant browser tab. We will term this "X-out". The tab is lost and there is no currently-visible URL.
2. Users want to refresh the information on a page for some reason such as wanting to re-enter the data typed on the page or to unfreeze a page that seems stuck. Sometimes selecting refresh is accidental. By refreshing, the currently-visible URL is relaunched.
3. Somebody has made a copy of the web-page URL, the page is closed, and later the web page is relaunched using the copy of the URL.

All three forms of restart involve security issues. For example, copying a URL with the intention to restart an interview is a clear way in which malefactors can try to highjack an innocent encounter with the intention to hack. The types of check discussed in an earlier paper deal with these issues.[2] The result for each check is a Quit command, and what to do after the Quit is not specified.

Issuing a Quit is related to our final problem – how to handle a Blaise 5 server session timeout. A server timeout happens when the user does nothing in the interview for more than a specified time (the default for Blaise 5 is 20 minutes). For a Blaise 5 server, "do nothing" does not necessarily mean nothing has been done, rather it means that no data were sent to the server from the web page for the specified time – data might be entered on the page but no next page action was issued.

3. Session data and the survey database

When you start a Blaise 5 web interview, the Blaise 5 server creates a server-session database by copying data from the survey database for the appropriate case. As the interview proceeds, data collected from

[1] However, for methodological reasons, some clients do not want an Exit or Save button on survey pages. Other clients, for other methodological reasons, want such an option available on all master pages.
[2] "Using the Resource Database to control web security", G J Boris Allan and Peter Stegehuis (IBUC 2020).

web pages are reflected in this session database. At the end of an interview, or normally after a `Quit` action, the data in the session database are copied to the survey database.

If you X-out from a web page (say, close the browser by selecting the X upper right) then the server does not know the user has disconnected, and the session is left open to be closed after the timeout period (we usually use the setting that says, on timeout, session data are saved to the survey database). Proactively, our mangement system (MS) could monitor the status of the remote user, and know when connection was lost by an X-out or a closed browser (the Blaise server is non-proactive, and only after it has had no connection from the user for the specified period is the session closed and data saved).

Suppose we have this clean break in connection, and the user tries to get back into the interview legitimately via the MS. It is easy to imagine that closing the browser (or a tab) could happen unintentionally and an attempt made to restart the interview within the timeout period. So here we are: the respondent accidentally has closed the tab, yet wants to continue, and comes back to the MS, going through all the recognized steps.
As all steps are correct with the login, the MS (using the API) starts the interview. The MS tries to set the IsLaunched field to Yes, but cannot because IsLaunched is inaccessible – we cannot start because a locked session is in progress. That is, the session data are still live (have not been deleted) because not enough time has elapsed for a timeout.

On a second login, the management session can know that session data are live, so it would seem obvious that the session data should be saved to the survey database, and a new session started. If the session data are not deleted then a new (second) instance of the interview cannot be started, because the Blaise server thinks the record is "locked". The record in the survey database is not locked, rather the session-data record is locked, and will remain locked until the first instance times out.

There is no way that it is possible using the API to nullify the lock ID attached to a session database instance. We have proposed an addition to the API which will give this flexibility, but until then we use a more elaborate method with the resource database. There are other reasons we would like this flexibility with the lock ID, because the less complex the security mechanisms we have to use the more robust our defences. The integrity of the Blaise database has the highest priority so allowing a user (however well-qualified) to remove a lock would not be implemented. In addition, automatically locking a record is a good default. In some situations this can create issues, and further discussions are underway on this point. Currently, the integrity of the Blaise database is paramount and removing a lock is not permitted.

Another less complex approach we considered was using the API to see the original session instance, and then to call the `Quit` method on the session. The MS (via the API) would then loop waiting for the session to close and, once closed, launch a new session instance.

There was, however, a problem with the time taken for the original session to close because, for any particular session, the time taken to close was unpredictable. In some tests closing took seconds and in other tests closing took minutes. We did not think, given the variation, that this approach was consistent enough to field. If the closing time could be made consistent and short then we thought this approach would solve the restart problem and might be easier to implement than the unlock feature.

If the user (or an intruder) tries to restart the case by using the original URL after an X-out before the session is closed:

1. Whenever we start a new interview session, we compare the previous session ID to the current session ID,

2. If the two IDs are the same, then we can

    2.1. `Quit` and `Save` the session,

    2.2. give a warning,

    2.3. return to the MS.

To return to the MS, we use a URL that we have stored in the survey database (and thus in the session database).
When a session times out there are no session data available, because in theory the data have been saved to the survey database and the session has been removed from the server. However, there were problems when the session had not been removed before a new survey was opened from the same IP address – this meant a case was launched with the new case ID but with an old session.[3]
Once the session no longer exists, we cannot access the values in the survey database directly but, however, Blaise 5 opens *a generic* error page that reports the type of "error". We suggest that Blaise 5 should open a *specific* session timeout error page. The specific session timeout error page will have a custom action that will enable us to act on the timeout event.
We know of two ways that we can approach this custom action, either launching a hard-coded URI or launching a special survey case that has an exit URI that can be changed as the needs of the project changes.

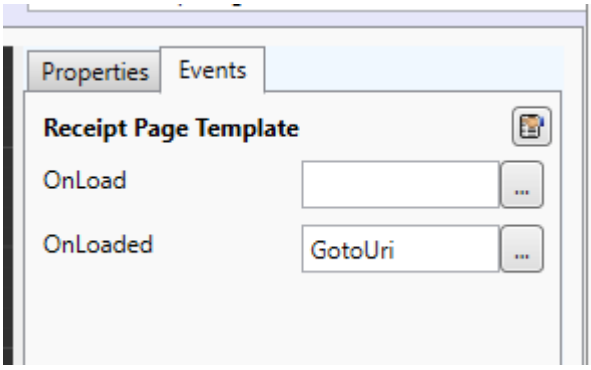
4. Different ways of ending an interview

There are various types of page template that are used to cope with reasons to leave an instrument.

4.1 Receipt Page Templates

This is the way we would like all interviews to end. That is, the interview has gone all the way to the end, and is complete.

Often receipt pages thank the user and, to end the interview, have a button to select for a return to the MS. Other clients like to thank the user, but want to return automatically to the MS and thank them there. We accomplish the return to the MS by use of a redirection of program control to the management site when the page has been loaded (`OnLoaded`):



[3] We think this was remedied in Blaise 5.6.9.2082.

The `OnLoaded` program expression is:

```
<ReceiptPageTemplate OnLoaded="{Action GotoUri({Expression ExitURL.ValueAsText},{Values
        'Target=_parent'})}">
```

That is, when the page has been loaded, go to the URI (ExitURL) that has been mapped from the MgSys block – this is, the address of the MS web site.

4.2 Abort Page Templates

There are times, within the set of rules in the resource database, when there is an instruction to Quit and this action opens an Abort page (calling this action an abort seems a little extreme). In our system, the Default abort page template does not use a field reference but uses a server variable named URL – the idea is that we have extra flexibility because (if necessary) we could change the server variable value depending on the source of the Quit – even assign a hard-coded reference.

The expression is:

```
<AbortPageTemplate OnLoaded="{Action GotoUri({Expression ServerVariables.GetString('URL')},{Values
        'target=_parent'})}">
```

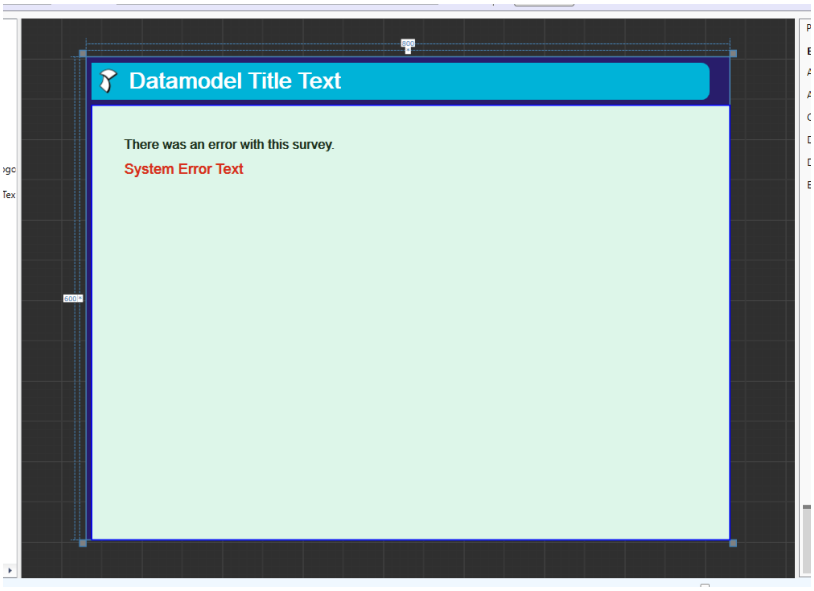
This follows the same format as the receipt page `OnLoaded` event.

4.3 Error Page Templates

If an instrument encounters a “system” error, such as a record locked by another user, then an error is shown. The list of system error codes is long, and includes:

- `InvalidKeyValue,`
- `RecordExists,`
- `ServerAccessDenied,`
- `SessionExpired,`
- `BrowserNotSupported,`
- `SurveyNotActive,`
- `RecordNotFound`

You will probably be aware of these error pages because of the Default error-page template:



In general, we do not change the default error page template because we do not want the user to have any easy way back to the survey MS, as the user could be an intruder testing ways into the instrument (hacking). We do, however, have a special TimeOut error-page template for one specific error – in the list of error codes the code named “SessionExpired”.

5. The Timeout error page template

In the resource database Error Page Templates list we have a new template named “TimeOut”, which is first in the list of error templates. This means that Blaise 5 looks at this template first, and decides if the template is applicable for the current session error.

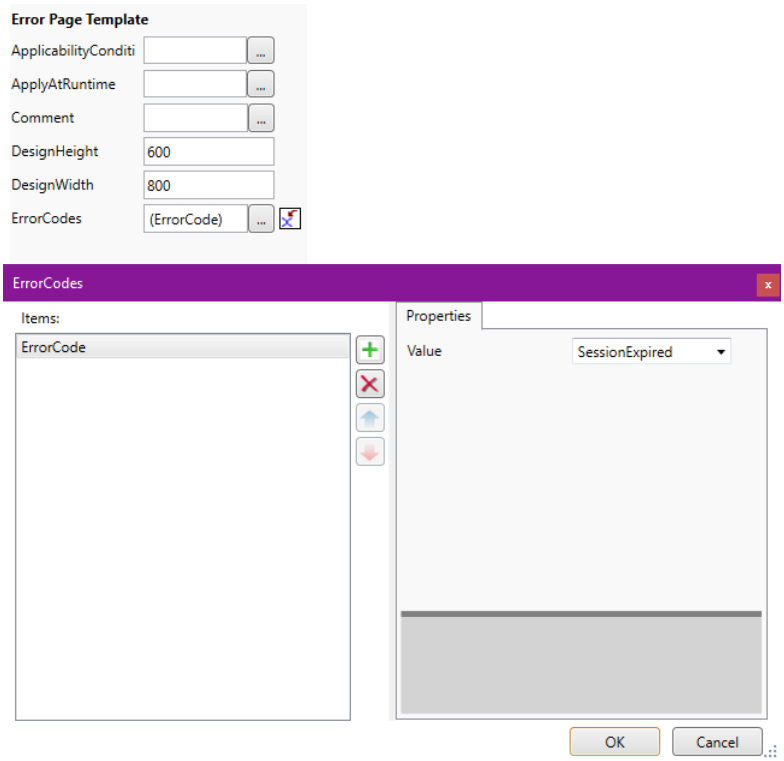
5.1 TimeOut version 1

Looking at the underlying expressions for this version of the Timeout error template, two interesting aspects are `OnLoaded` and `ErrorCodes`:

```
<ErrorPageTemplate
        OnLoaded="{Action GotoUri({'https://www.westat.com',{Values 'target=_parent'})}"
        ErrorCodes="SessionExpired">
```

We cannot use server variables or field references for our `GotoURI` because the session expired and, with the session expiration, the variable references also expired (and we have not found a way to keep them alive). Only errors that are “SessionExpired” are recognized by this template, and so – as TimeOut is first

in the list of error page templates – whenever the session expires this page is selected. Our problem is the hardcoded value of the URL.



5.2 TimeOut version 2

In an attempt to give more control over where we send the user after a session has expired, we needed to start a new session about which we had clear knowledge, and then act on information in that new session (a bit like our action with `Quit`/`Abort`) to send the user to a web page. If we hardcode the web-page location in the `OnLoad` event for TimeOut, whenever the location changes then we have to make the change and reinstall the survey.
The new session we create is back on the Blaise server and is attached to a "dummy" record/case we know as "*". The * case is part of the main survey database, and we can assign (via the API) appropriate values to fields in MgtSys. We do not assign a value to the field that says the case was started correctly, and so the resource database code issues a `Quit` instruction, that generates a Default abort page which then goes to a URI defined in the * case MgtSys block.

To start the survey with * in the current survey database, we use the `OnLoad` event (not `OnLoaded` because the action we want cannot be selected for that event). Here is the expression:

```
<ErrorPageTemplate
        OnLoad="{Action StartSurvey({Expression State.InstrumentId},{Values '*'},'','')}"
        ErrorCodes="SessionExpired">
```

When we load the TimeOut page, we run the `StartSurvey` action for the current survey (which is based on the resource database `State.InstrumentId` variable, not lost with the session). The survey to be started has a key value of *, and the only time the page is relevant is when the Blaise system error is `SessionExpired` and we start the * case.

This technique seemed to work during development but then we put it through more extensive testing. What happened in practice is that the previous session seemed (sometimes) to be still alive and not ended, so that the previous key value was replaced by *, but the rest of the data from the previous key value remained. We decided for the pretest to hardcode the return URI.[4]

[4] After that decision was made, Blaise 5.6.9.8022 was released and in the change log we read that there had been a problem with `Action StartSurvey`, very relevant to our experience. The change log read: `2019.11.25 | HELS | Resource Editor | BL56-993 | BSR-3698 | End current session before StartSurveyAction is executed`. This explains our experience. We are not, however, changing anything until the pretest is in the field.