Session Data Preservation and Migration—Problems and Solutions

Jason Ostergren and Helena Stolyarova, University of Michigan Institute for Social Research

The Health and Retirement Study (HRS) at the University of Michigan Institute for Social Research is a longitudinal study that originated in 1992, switched to conducting interviews using Blaise 4 in 2002, and moved to Blaise 5 in 2018. Among the challenges HRS has faced since the switch to Blaise 5 is how to handle Blaise 5 session data when an interview is not completed in one sitting. Session data is the working database that maintains the state of the instrument, including the values of temporary data and properties of fields. If that data is lost in a case where an interview was interrupted and has to be resumed later, significant problems can result if the instrument is dependent on session data to resume correctly, as is the case with HRS. There are a number of aspects to this problem, and HRS has tackled new and different ones in each of the three (2018, 2020, and 2022) waves of interviews since adopting Blaise 5. Data migration has been at the center of some of these issues—for example, HRS has had to incorporate careful attention to harmless changes into processes for updating instruments in the field. Most problems have been solved, sometimes with significant help from CBS, but with varying degrees of completeness—for example, it was thought that data migration issues had been solved in 2020, but it turned out that mode switches rendered that solution incomplete. Finally, HRS has determined that it would be useful to retain and preserve session data after interview completion, which has been partially solved as well. This paper will break down why session data has been important to HRS, as well as the various issues and solutions to the problems that have arisen—including testing tools—and what remains to be done.

1. Session Data

A brief discussion of the nature and purpose of session data is in order. Session data is the stored state information from the Blaise 5 session service, which handles active interview sessions. The runtime session database (shortened to "session database" here), where the session data is stored, is regularly updated with data from the active interview while in session. Additionally, the session database persists session data when an interview is interrupted. When an interrupted interview is resumed, provided there is a primary key, Blaise first checks to determine whether session data are available. If session data for the key are found in the session database, the interview is resumed where it left off, with all session data present, including auxiliary data and survey state. This process is at the heart of the issues discussed in this paper. There are a variety of ways that session data might be lost before an interview is resumed, and in a survey like HRS, this causes problems for any interview that is interrupted. Generally speaking, HRS needs session data to be preserved both between sessions and across new datamodel releases (within an HRS "wave" [e.g., HRS 2020]). There are a number of reasons that this is the case.

At first glance, session data may not appear to be a crucial component of the Blaise system for a user to understand. Indeed, in many use cases, there would be no need for a Blaise user to consider it at all. Session data-related issues arise mainly due to the ways external processes interact with Blaise interviews. A simple web survey that is deployed and runs without interruption or intervention until the end of its interview period would be unlikely to be impacted by these issues. The failure points that HRS has experienced occur in transitional processes. One such process happens when particular interviews are stored and transferred within our system—for example, between servers in the building and field laptops. Another such process happens when new versions of the HRS datamodel must be deployed, replacing old versions (rather than multiple versions running side by side).

2. HRS and Session Data

The other element that makes session data a concern to HRS is the complexity of the HRS survey instrument. HRS has design features, legacy code, and operational requirements that turn out to make session data important in a number of ways. A very simple instrument subject to the same transitional processes alluded to earlier may have some problems (e.g., resetting the position to the first question), but those would be among the more manageable ones HRS encounters. The risk was not immediately apparent to HRS. In fact, HRS has come across new problems connected with session data in each of the three "waves" since adopting Blaise 5.

Among the design features and legacy code found in HRS that make session data significant are the following.

HRS has reusable question series for estimation (referred to as "unfolding sequences," or just "unfoldings" hereafter) that are scattered throughout the instrument after questions where a respondent may be unsure about an important amount, such as the value of a house or pension. These are programmed as procedures for legacy reasons and, as such, are made up of temporary data—even the questions defined as Blaise Fields are actually treated as auxfields. Historically, it was possible to close a routing gate after each procedure was complete and preserve the final result. If an interviewer needed to get back in, there was a complicated method of doing so involving erasing answers and moving back and forth. In a theme that will repeat in this paper, the need to support web self-interviews required a change.

HRS also ran into a variety of issues connected with losing session data that were not HRS specific. For example, when resuming a case that had lost session data, the interview would revert to the first field on the route rather than the field at which the interview was suspended. This forced the interviewer—or worse, a self-interview respondent—to step through the dozens or hundreds of questions that had previously been answered. Another such problem was that formerly suppressed signals would be reset and require intervention again.

HRS also ran into some still-mysterious data-loss issues that likely result from complicated programming for arrayed data (in this case, data about children-in-law) reacting badly to the loss of session data upon resume. This highlights the need to consider session data in testing. HRS has a variety of custom tools for testing its instrument and dedicated staff to handle the testing. An enormous amount of time is spent testing each instrument to minimize flaws. However, until HRS discovered the kinds of session data problems described in this paper, there was no provision in those tools to simulate loss of session data; therefore, problems resulting from that eventuality (like the missing children-in-law) were not discovered before the field period began.

One last wrinkle is that there was initially no effort to preserve any session data after the interview was over. When problems were discovered later on, there was a desire to look at session data to troubleshoot and reconstruct missing data, much like is often done with audit trail data. Paradoxically, of course, even if the session data were preserved in general, it would have been partially missing from the problem cases. Nonetheless, while preventing loss of session data during suspend and resume was the highest priority, HRS realized that once that was solved, it would be desirable to retain session data even after the interview was complete for later reference.

3. HRS Session Data Vulnerabilities

To expand on the reasons mentioned earlier for the HRS instrument being vulnerable to loss of session data mid-interview, a little background is required. The design and programming of the HRS instrument, which goes through a process of reevaluation and improvements in each two-year cycle, attempts to balance out a huge amount of content with extensive customization of flow, question text substitutions, and other tricks. The interview typically clocks in at more than two hours—sometimes much more—so a lot of attention is paid to complicated logic to minimize the number of questions and to an equally complicated amount of logic to make question text personalized for each respondent as much as possible. When the decision was made to add a web self-interview mode to the HRS instrument starting in 2018, a comprehensive reevaluation of all this logic was undertaken. For example, it was quickly determined that the self-interview needed to support respondents if they simply clicked next rather than answering a question. In other words, our interviewer-administered mode rarely allows empty answers—the interviewer may probe when instructed or assign DK/RF or other codes as needed. HRS determined that none of these options should be forced on self-interview respondents ("self Rs" from now on).

Furthermore, where previously HRS made extensive use of gates (using conditions that stored data from sequences and removed them from the flow) to close off precisely the kinds of sequences that are now causing problems in session data, now HRS strives to minimize such devices to prevent confusion among self Rs. The reason for this was mainly that having a "previous" button to allow the self R to have control over the ability to return to a previous question was important, but gates that removed previous questions from the flow would make the self R unable to locate their previous answers. Sometimes this could be handled with instructions to the self R, but it was determined that these should not be common occurrences anymore.

There were multiple aspects to addressing this issue. For example, HRS gates normally rely on the interviewer answering a gate "question." This question would have some instruction to the interviewer, even if there was no text to be read to the respondent. The interviewer would then know that once they answered the question, the previous sequence would be inaccessible (at least without some complicated process). Since HRS determined not to force any answers on self Rs and Blaise logic requires some catalyst to properly close a gate, HRS had to remove or replace gates with other mechanisms. Where HRS simply removed gates entirely, as in the case of the aforementioned unfoldings, the loss of session data would cause the code to be reevaluated and the final answers would be emptied out as a result because the session data they relied on was emptied out.

When HRS retained existing gates, it turned to other mechanisms that proved vulnerable to this problem, as well. One such mechanism is the use of the isVisited property to trigger a gate in place of a human-selected answer when, as the name implies, the field is visited (appears on the screen). Since some gates were still required, this was used in a handful of places, including some that turned out to require very complicated logic and even custom browser DEP code. The use of isVisited presented its own problems initially. HRS quickly discovered that it was losing this property data alongside the session data in 2018. We believe that in Blaise versions released after that time, it became possible to persist properties like isVisited by redeclaring them. However, HRS has not explicitly tested this in our process, and we are still under the impression that isVisited is precarious without preserving the session data. The point of all this is that if session data is lost and isVisited is lost with it, gates are unintentionally opened, and the previously gated logic is then reevaluated incorrectly because the gated logic also relied on now missing session data.

4. HRS 2018

With a lot of troubleshooting and experimentation, and with considerable help from CBS, HRS has made progress in chipping away at the kinds of problems outlined above. In 2018, HRS experienced widespread problems with loss of session data. In the case management system used for interviewer-administered cases, session data was being lost on every occasion when an interview was suspended and resumed. This resulted in problems like data loss in unfoldings, as described above. This not only caused a loss of data, but also caused respondents to be reasked many questions they had answered before—or alternatively, it caused an employee to have to painstakingly reenter the previous answers from the audit trail, for example.

This problem turned out to be one of the easiest to solve and simply stemmed from inexperience with session data the first time out. In the case management system that was being used, the case files were always packaged and stored after a suspend and the working folder being cleared. When a case was resumed, those files were brought back to their previous locations. However, because there was not a general awareness of the importance of session data, those files were being deleted in this process. Once we realized this, it was a simple matter to retain the session data files, and this problem was solved.

Along these lines, one key takeaway from this paper is that session data should be considered a normal and necessary component of Blaise 5 operation. At the very least, care should be taken with whether it is being retained up until the point when an interview is truly completed.

5. Harmless Changes

Another concept that needs attention is that of harmless (or harmful) changes. This notion concerns data file compatibility (in particular, for the purposes of this paper, session data file compatibility) in a case where a survey needs to be replaced with an updated version of itself. While changes to a survey that break data file compatibility may be common during the development of a survey, once real data collection starts, serious problems can arise if compatibility is broken. Blaise uses a checksum to determine compatibility and may refuse to launch or install a survey if the checksum fails. To make things slightly more complicated, some changes that cause checksums not to match may still be harmless. For example, changing the number of fields in a survey breaks data file compatibility and alters the checksum, but the change can still be considered harmless if the change was due to fields being added rather than deleted. The Blaise 5 online help specifies a set of general rules governing this situation, as follows:

- No (relevant) items may be removed.
- Items can be added.
- Each item of the new collection must be a harmless extension of the related item (i.e., with the same name) of the old collection.

The online help also notes: “As a rule of thumb, extension of the data definition, such as an additional field, enlarging a string, or expanding answer categories, are fairly harmless.”

In practice, if one needed to harmlessly rename a field, for example, it would be necessary to retain the original field (while taking it off the route, probably with a keep statement) and add a new field with the changed name. Simply renaming the field would be a harmful change because it effectively removed that field from the instrument. In principle, there are ways to make almost any proposed change harmless with enough attention to detail.

6. HRS 2020

HRS was forced to begin thinking seriously about how to handle changes between versions during its second (2020) fielding of a Blaise 5 instrument. One of the (perhaps peculiar) features of HRS is that during the field period, which often lasts most of a year or even more than a year, HRS updates its instrument with fixes for problems identified in the field and with high-value changes requested by investigators. As a result, HRS has a history of releasing multiple versions of its datamodel each wave (an average of one per month over a year is not surprising). Importantly, these are very similar datamodels, usually with logic or wording bugfixes, new Spanish translations, or added question sequences arising from unexpected events (e.g., COVID-19). In other words, they are conceptually identical for the most part and lack the kinds of wholesale rewrites of sections or sequences that can be made during the preproduction phase between field periods. Therefore, HRS has always assumed (and found in practice in our interviewer-administered-only Blaise 4 environment before 2018) that data could be easily transferred between these versions within a "wave." In particular, HRS assumed that there would be no problems when a case is suspended in one datamodel version but must be resumed in a later one (incidentally, the versions may not even be consecutive, since there is occasionally a long lag between suspend and resume, based on the respondent's schedule or wishes).

Unfortunately, HRS discovered that it was not possible to resume a case in a newer datamodel version that was suspended in a previous datamodel with session data intact. By this point, HRS had learned to preserve the session data files as described above, so this problem revealed itself only after having solved the previous one. What is more, it turned out that this newly discovered problem occurred even if there were no harmful changes between the two datamodels in question (as of Blaise release 5.10.6).

To take a step back, HRS at first thought that data incompatibility might be the cause and looked deeper into understanding harmless/harmful change concepts. As implied above, the key requirement for preserving session data between sessions in different datamodel releases is that the new datamodel must not have harmful changes. Because the rules governing harmful or harmless changes can be a little vague, it is necessary to resort to some form of testing or tool to detect harmful changes. For example, Blaise 5 ships with a sample Manipula script for detecting harmful changes, which can be found in the samples folder:

- \Documents\Blaise5\Samples\Specific Features\Manipula\HarmlessChanges\HarmlessChanges.bsol

This script compares two datamodels and reports whether it detects either no changes or only harmless changes, or, as a third possible outcome, it lists all harmful changes. HRS had previously tried this Manipula sample, but it did not work well due to a bug causing a listing of more than a thousand false positives. As a result, HRS was essentially guessing at whether changes were harmful and was not fully engaged in documenting this. When HRS began to run into these datamodel version update problems during the 2020 field period, CBS was called in to help and quickly provided a small code change to make this script work as intended for HRS. CBS also provided sample API code for building a tool to test datamodel updates with harmless changes, along roughly the following lines:

```
private void ApplyHarmless(string oldDMbdixFilename, string newDMbmixFilename)
{
        DataLinkAPI.IDataLink5 dl = DataLinkAPI.DataLinkManager.GetDataLink(oldDMbdixFilename) as
        DataLinkAPI.IDataLink5;
        dl.ApplyHarmlessChanges(newDMbmixFilename);
}
```

HRS was then able to verify compatibility between datamodels and was immediately presented with the next problem: session data was *always* treated as incompatible, regardless of the harmless change determination in the version of Blaise HRS was using for development at the time (5.10.6), as well as earlier versions.

In order to solve this problem, CBS provided a special unsupported 5.10.10 release to accommodate this need for the then upcoming start of HRS 2022 interviewing. Basically, in that version, the

“ApplyHarmlessChanges” would successfully run against the session database, allowing a suspended case to be resumed in a new datamodel version, provided that only harmless changes were present between the two. This capability should be a normal part of the Blaise 5 feature set from 5.13 on (note that HRS never tested these features in 5.11 or 5.12, but CBS had specifically mentioned 5.13 in this regard, so that would seem to be the best starting point), meaning updates that preserve session data are now a normal supported part of Blaise 5 operation. So far, our testing in 5.13 has borne this out. In addition to the API functionality, a number of options now (as of 5.13) appear in the Blaise 5 Server Manager when updating an existing datamodel that allow for this, as will be discussed later.

7. Testing Harmless Change Compatibility

This new capability (in 5.10.10 and 5.13.x) has allowed HRS to add functions to test session data migration between datamodels in our testing tools. So, to complement efforts by programmers to keep harmful changes in mind while programming (the first line of defense), and to allay any lingering doubt about the aforementioned Manipula script’s accuracy (which is the second line of defense), HRS can actually test suspending a case in one datamodel, updating the datamodel, and attempting to resume using the main HRS case testing tool. Additionally, the staff at SRO, who handle field operations, do a final test using tools with the aforementioned code. As a result, session data problems due to this set of issues have been minimized in the 2022 field period.

Following is an illustrative process (as of Blaise 5.10.10) for testing datamodel migration with harmless changes to preserve session data, which works in a normal server deployment:

1. Copy a preloaded .bdbx into the .bpkg of the initial datamodel (or equivalent process to handle preload).

2. In Server Manager, install the .bpkg of the initial datamodel.

3. In Server Manager, start in browser with normal arguments, test some “unfoldings” and signals, and suspend.

4. Run a tool with the ApplyHarmlessChanges API function as administrator against the deployed .bdix in the \Blaise5\Surveys folder and the .bmix from the new datamodel with the harmless changes.

5. In Server Manager, start in browser with normal arguments. It will then resume in the correct place in the new datamodel with session data intact.

The problem with updated datamodels that was described above had one particularly troublesome twist that will require some extra detail and an alternative set of testing steps to describe. HRS has been handling web self-interview and telephone or face-to-face interviewer-assisted cases using different systems so far, though efforts are being made to eventually merge everything into one. The web self-interview cases run from a server in a conventional way, but the interviewer-assisted cases run in a locked-down laptop environment in Windows DEP standalone mode. A lot of extra steps were involved in making datamodel updates work in that environment.

Following is an illustrative process (as of Blaise 5.10.10) for testing datamodel migration with harmless changes to preserve session data, which works in Standalone mode:

1. Copy a preloaded .bdbx into the .bpkg of the initial datamodel (or equivalent process to handle preload).

2. Copy this .bpkg of the initial datamodel into the same folder as the DEP (and its associated .dlls).

3. Run the special DEP with the commandline argument -RunMode:ThickClient (and other normal arguments), test some “unfoldings” and signals, and suspend.

4. Run a tool with the ApplyHarmlessChanges API function against the deployed .bdix and the .bmix from the new datamodel with the harmless changes.

5. Copy the three files that are updated by the ApplyHarmlessChanges tool from the deploy folder into the same preloaded .bpkg from Step 2.

6. Run the special DEP with the commandline argument -RunMode:ThickClient (and other normal arguments). It will then resume in the correct place in the new datamodel with session data intact.

8. HRS 2022

In the 2022 field period, HRS encountered another new twist on session data preservation. This was connected with a new style of mode switch being supported in 2022. In this approach, cases could be passed from the standard web server deployment into the offline standalone laptop environment. To make this approach work for HRS, SRO uses a heavily modified custom DEP, based on CBS examples and assistance. The sync process in this custom DEP presents issues with preserving session data. In a variation on the previously discussed suspend-resume issue, HRS had cases that switched from web-based self-interview to telephone interview from an offline laptop, and many of these cases had been started before the switch. It was discovered that starting the case prior to the switch caused many calculations and assignments to happen, even if it was suspended on the first screen, and those were reflected in the .bdbx that was being transferred, causing problems. If the case was resumed using only the .bdbx without the session data in that situation, you would get some corrupted data. For example, the aforementioned problem where children-in-law got deleted cropped up as a result of this.

A number of discussions with CBS have resulted in a possible fix coming down the road in Blaise 5.14 or 5.15. Just like the previous changes concerning preserving session data during datamodel updates solved that problem in the HRS 2022 field period, HRS is hopeful that these upcoming fixes will eliminate one more cause of session-data-related difficulties in the HRS 2024 field period.

One clear trend in all of this is that each set of Blaise fixes have unmasked new deeper issues with HRS and session data over time, but it is also the case that the magnitude of the problem keeps shrinking wave-on-wave. Whereas the first issue impacted potentially thousands of cases, the subsequent issue may have affected hundreds of cases, and the most recent is impacting perhaps less than a hundred. Additionally, new versions are bringing more features to address these issues. For example, the Blaise 5 Server Manager presents more options when installing surveys over previously existing ones in Blaise 5.13. Some of those will be discussed below. Overall, our hope would be that questions about what happens to session data will eventually permeate every part of the system.

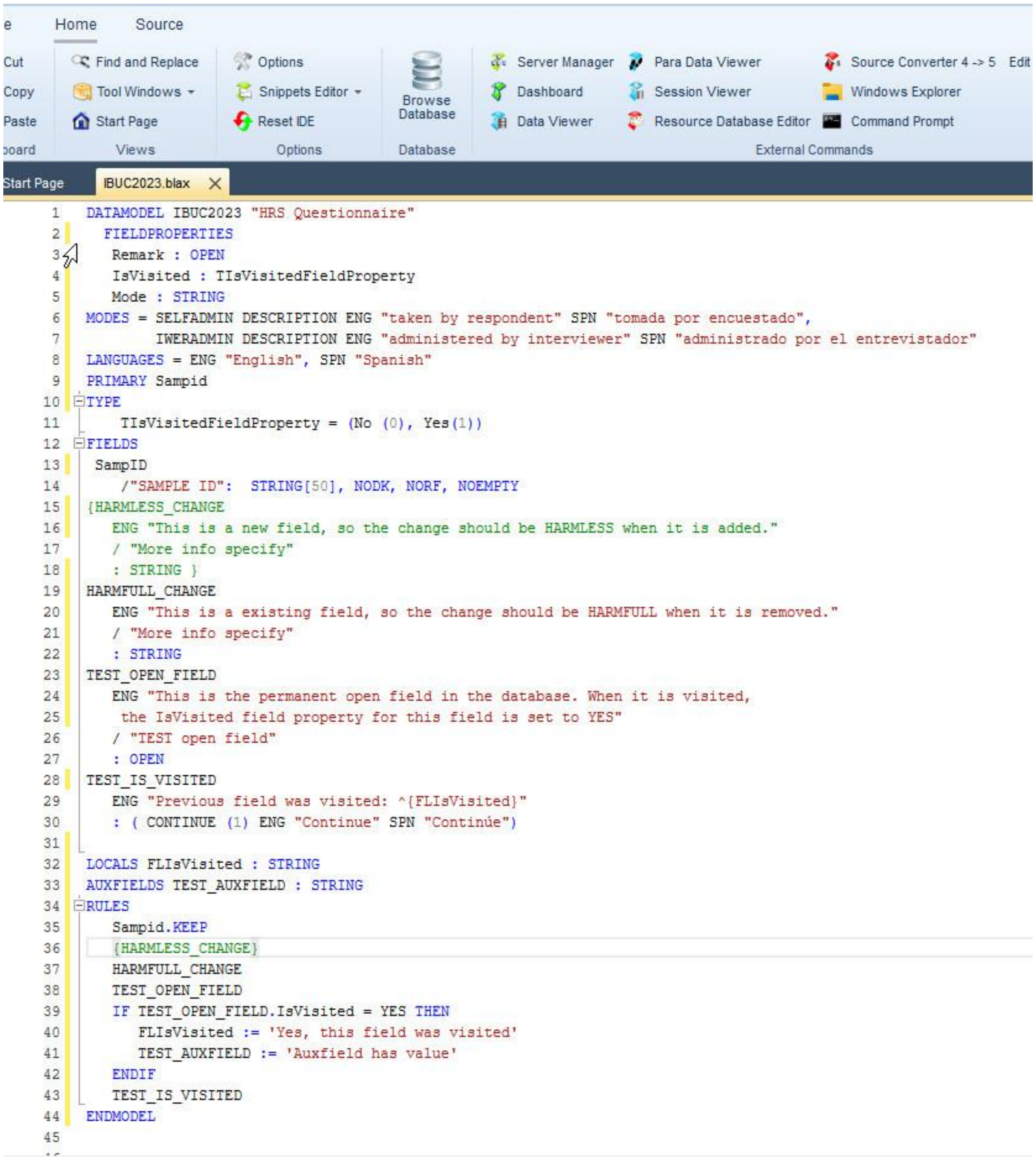
9. Preserving Session Data After Completion

One last concern that fits by virtue of being about session data but is not a bug or unsolved issue of any sort is the preservation of session data after a case is completed. So far, this paper has discussed mainly preservation of session data on suspend and resume, first within the same datamodel, then across different datamodels, and finally through the custom DEP sync process. All of those were concerned with successfully completing the interview when multiple sessions were involved. This additional issue of long-term preservation of session data is for all interviews. To some extent, what HRS wants from this is already handled by audit trail data—that is, the ability to go back later and review a case that had problems, determine what went wrong, and recover missing answers, as well as other related needs. Having session data available for later review would expand on that since it contains *all* the working data, such as auxfields. This could make later investigations easier by providing a more direct way to see what might have caused a particular problem.

HRS did not realize that session data was deleted upon case completion until well into the 2020 field period. In response, SRO was able to develop an approach for HRS that uses SQL Server triggers (the self-administered mode of HRS stores data in SQL Server) to copy the session data to another database right before any delete command is executed. This has worked well for the parts of HRS 2022 using SQL Server. In the future, CBS has indicated that new options may be added to natively prevent deletion of session data, if desired.

10. Examples

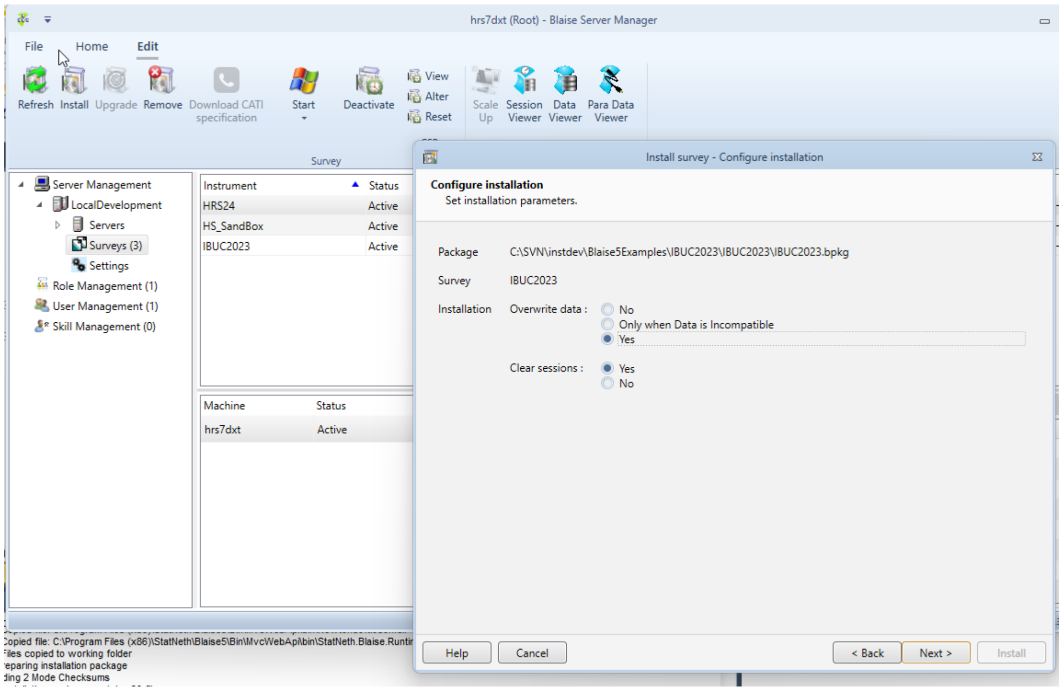
To illustrate harmful/harmless changes, we use a simple datamodel. Note that there is a field HARMLESS_CHANGE that is off the route for now. We plan to add it later on to demonstrate that this change is harmless.



```
DATAMODEL IBUC2023 "HRS Questionnaire"
  FIELDPROPERTIES
   Remark : OPEN
   IsVisited : TIsVisitedFieldProperty
   Mode : STRING
MODES = SELFADMIN DESCRIPTION ENG "taken by respondent" SPN "tomada por encuestado",
        IWERADMIN DESCRIPTION ENG "administered by interviewer" SPN "administrado por el entrevistador"
LANGUAGES = ENG "English", SPN "Spanish"
PRIMARY Sampid
TYPE
    TIsVisitedFieldProperty = (No (0), Yes(1))
FIELDS
 SampID
    /"SAMPLE ID":  STRING[50], NODK, NORF, NOEMPTY
{HARMLESS_CHANGE
   ENG "This is a new field, so the change should be HARMLESS when it is added."
   / "More info specify"
   : STRING }
HARMFULL_CHANGE
   ENG "This is a existing field, so the change should be HARMFULL when it is removed."
   / "More info specify"
   : STRING
TEST_OPEN_FIELD
   ENG "This is the permanent open field in the database. When it is visited,
    the IsVisited field property for this field is set to YES"
   / "TEST open field"
   : OPEN
TEST_IS_VISITED
   ENG "Previous field was visited: ^{FLIsVisited}"
   : ( CONTINUE (1) ENG "Continue" SPN "Continúe")

LOCALS FLIsVisited : STRING
AUXFIELDS TEST_AUXFIELD : STRING
RULES
   Sampid.KEEP
   {HARMLESS_CHANGE}
   HARMFULL_CHANGE
   TEST_OPEN_FIELD
   IF TEST_OPEN_FIELD.IsVisited = YES THEN
      FLIsVisited := 'Yes, this field was visited'
      TEST_AUXFIELD := 'Auxfield has value'
   ENDIF
   TEST_IS_VISITED
ENDMODEL
```

After installing and deploying this survey, we are able to view session data from Blaise Server Manager.

At the first installation, we choose the options to overwrite the data and clear sessions.



Instrument	Status
HRS24	Active
HS_SandBox	Active
IBUC2023	Active

Machine	Status
hrs7dxt	Active

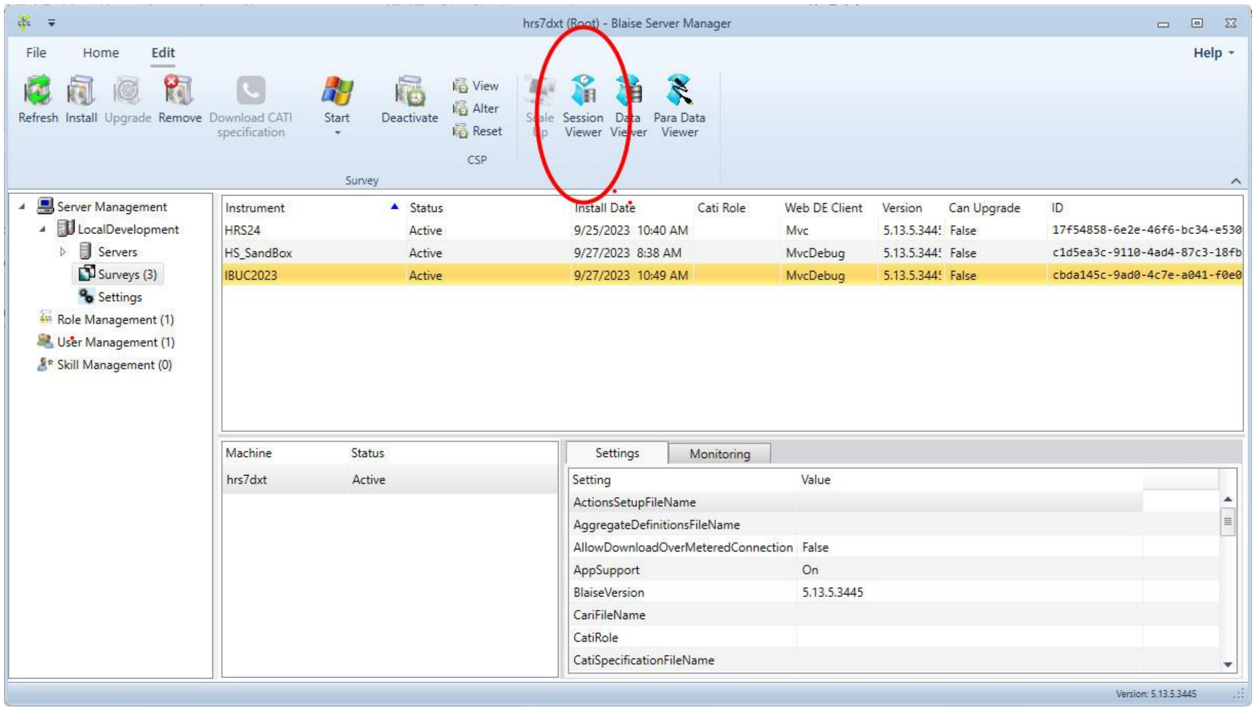
Install survey - Configure installation

Configure installation
Set installation parameters.

Package	C:\SVN\instdev\Blaise5Examples\IBUC2023\IBUC2023\IBUC2023.bpkg
Survey	IBUC2023
Installation	Overwrite data : No / Only when Data is Incompatible / [✓ Yes]
	Clear sessions : [✓ Yes] / No

```
opied file: C:\Program Files (x86)\StatNeth\Blaise5\Bin\MvcWebApi\bin\StatNeth.Blaise.Runtir
iles copied to working folder
eparing installation package
ding 2 Mode Checksums
```

After answering a series of questions, we are able to view the data in fields and auxfields through Session Viewer. You can also do it from the Blaise Control Center.

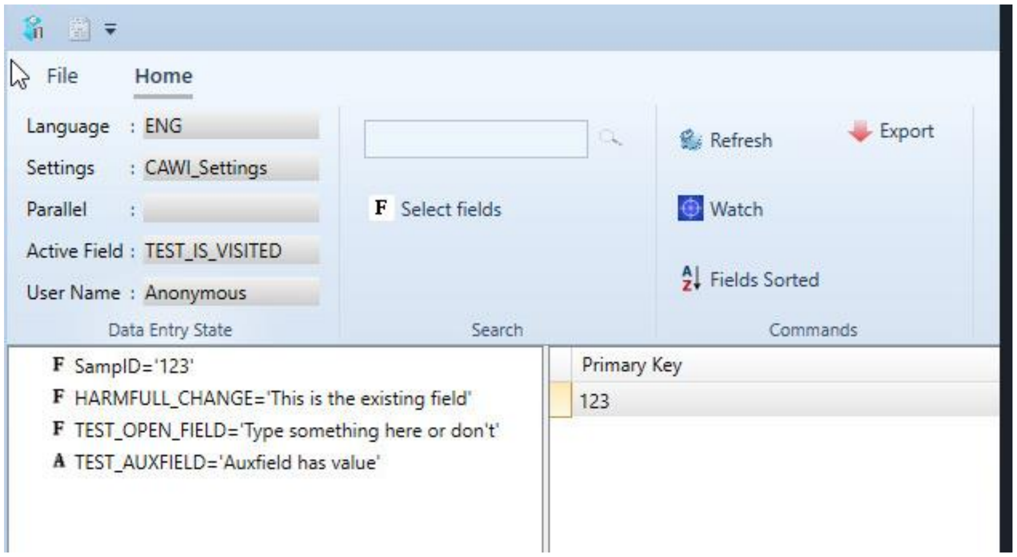


Instrument	Status	Install Date	Cati Role	Web DE Client	Version	Can Upgrade	ID
HRS24	Active	9/25/2023 10:40 AM		Mvc	5.13.5.344!	False	17f54858-6e2e-46f6-bc34-e530
HS_SandBox	Active	9/27/2023 8:38 AM		MvcDebug	5.13.5.344!	False	c1d5ea3c-9110-4ad4-87c3-18fb
IBUC2023	Active	9/27/2023 10:49 AM		MvcDebug	5.13.5.344!	False	cbda145c-9ad0-4c7e-a041-f0e0

Machine	Status
hrs7dxt	Active

Setting	Value
ActionsSetupFileName	
AggregateDefinitionsFileName	
AllowDownloadOverMeteredConnection	False
AppSupport	On
BlaiseVersion	5.13.5.3445
CariFileName	
CatiRole	
CatiSpecificationFileName	

Provided that your survey has a Primary Key, defined in your datamodel, you can view your session data:

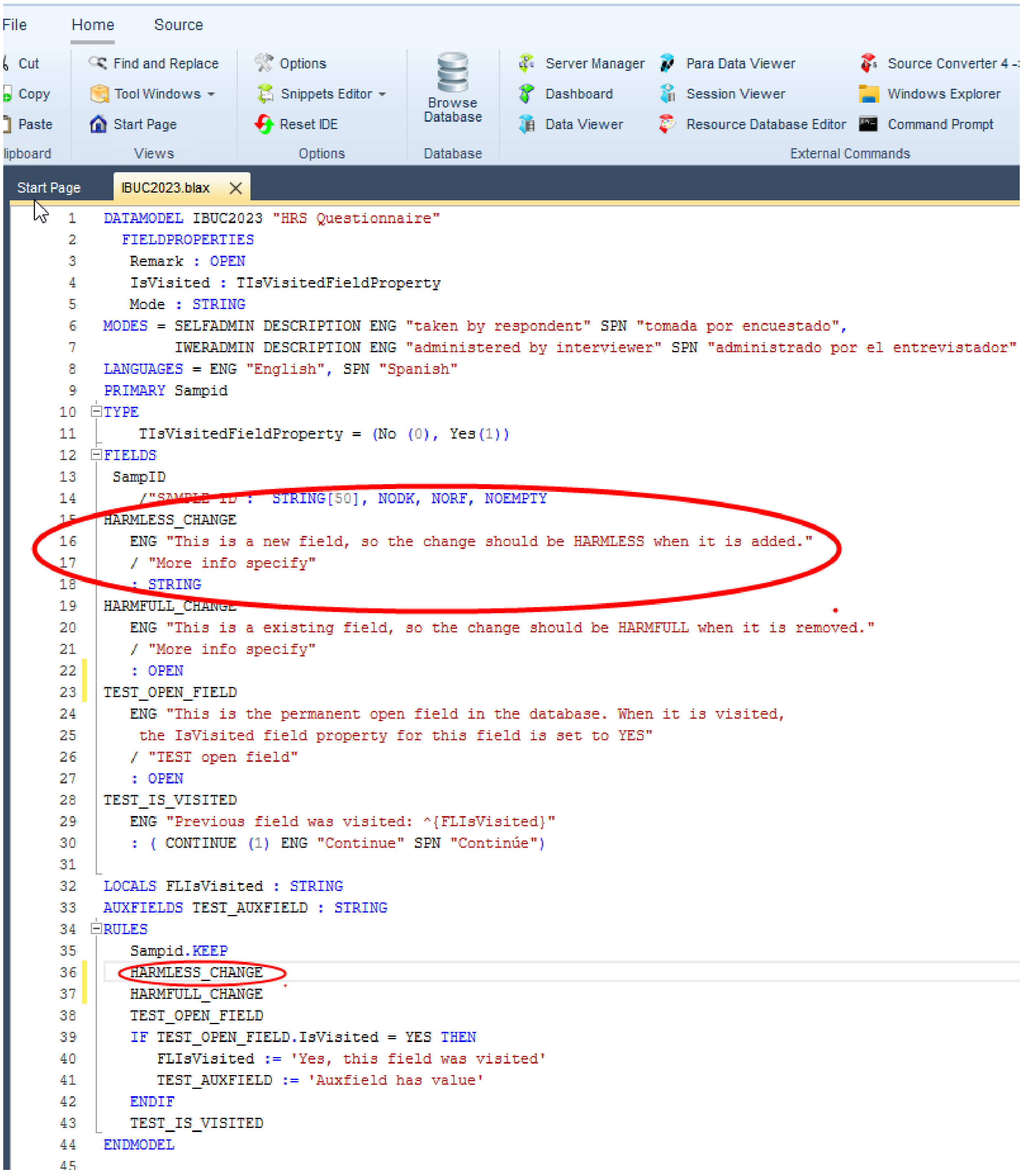


Language : ENG
Settings : CAWI_Settings
Parallel :
Active Field : TEST_IS_VISITED
User Name : Anonymous

F SampID='123'
F HARMFULL_CHANGE='This is the existing field'
F TEST_OPEN_FIELD='Type something here or don't'
A TEST_AUXFIELD='Auxfield has value'

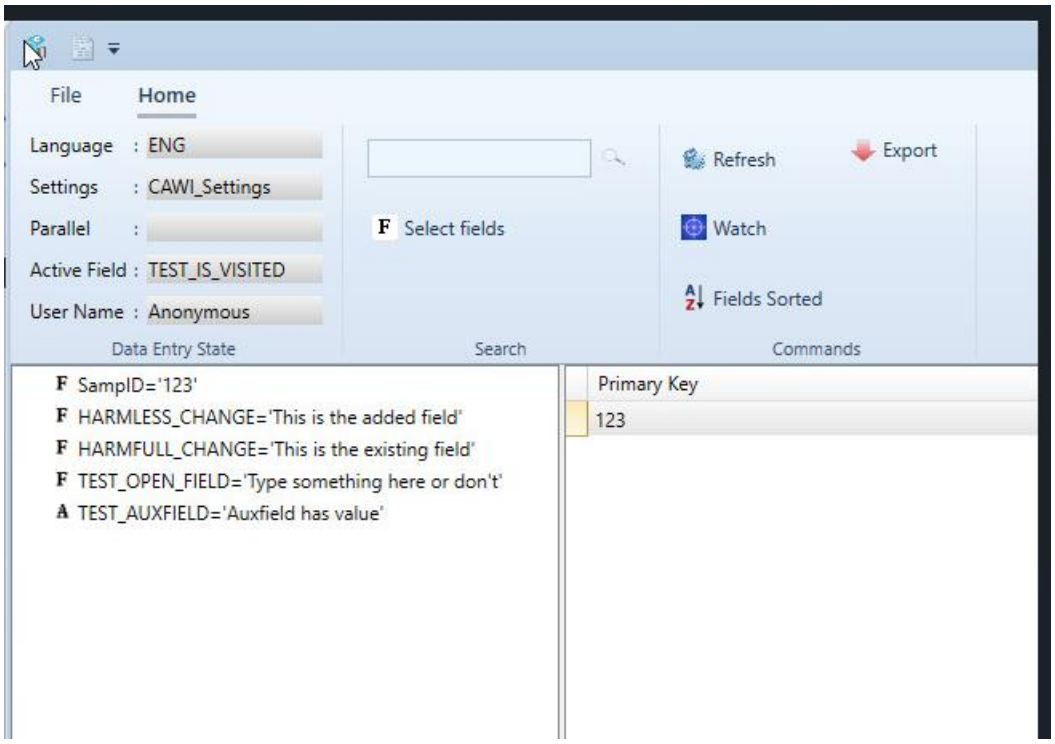
Primary Key
123

Now, suppose we want to add a new field to a datamodel that is already deployed in the field.



```
DATAMODEL IBUC2023 "HRS Questionnaire"
  FIELDPROPERTIES
   Remark : OPEN
   IsVisited : TIsVisitedFieldProperty
   Mode : STRING
MODES = SELFADMIN DESCRIPTION ENG "taken by respondent" SPN "tomada por encuestado",
        IWERADMIN DESCRIPTION ENG "administered by interviewer" SPN "administrado por el entrevistador"
LANGUAGES = ENG "English", SPN "Spanish"
PRIMARY Sampid
TYPE
    TIsVisitedFieldProperty = (No (0), Yes(1))
FIELDS
 SampID
    /"SAMPLE ID :  STRING[50], NODK, NORF, NOEMPTY
HARMLESS_CHANGE
   ENG "This is a new field, so the change should be HARMLESS when it is added."
   / "More info specify"
   : STRING
HARMFULL_CHANGE
   ENG "This is a existing field, so the change should be HARMFULL when it is removed."
   / "More info specify"
   : OPEN
TEST_OPEN_FIELD
   ENG "This is the permanent open field in the database. When it is visited,
    the IsVisited field property for this field is set to YES"
   / "TEST open field"
   : OPEN
TEST_IS_VISITED
   ENG "Previous field was visited: ^{FLIsVisited}"
   : ( CONTINUE (1) ENG "Continue" SPN "Continúe")

LOCALS FLIsVisited : STRING
AUXFIELDS TEST_AUXFIELD : STRING
RULES
   Sampid.KEEP
   HARMLESS_CHANGE
   HARMFULL_CHANGE
   TEST_OPEN_FIELD
   IF TEST_OPEN_FIELD.IsVisited = YES THEN
      FLIsVisited := 'Yes, this field was visited'
      TEST_AUXFIELD := 'Auxfield has value'
   ENDIF
   TEST_IS_VISITED
ENDMODEL
```

This change would be harmless. After installing a new survey over the old one, the new field is on the route and the session data is preserved. Note in the following screenshot that the auxfield data is also preserved. Prior to Blaise 5.10.10 this would not have worked.

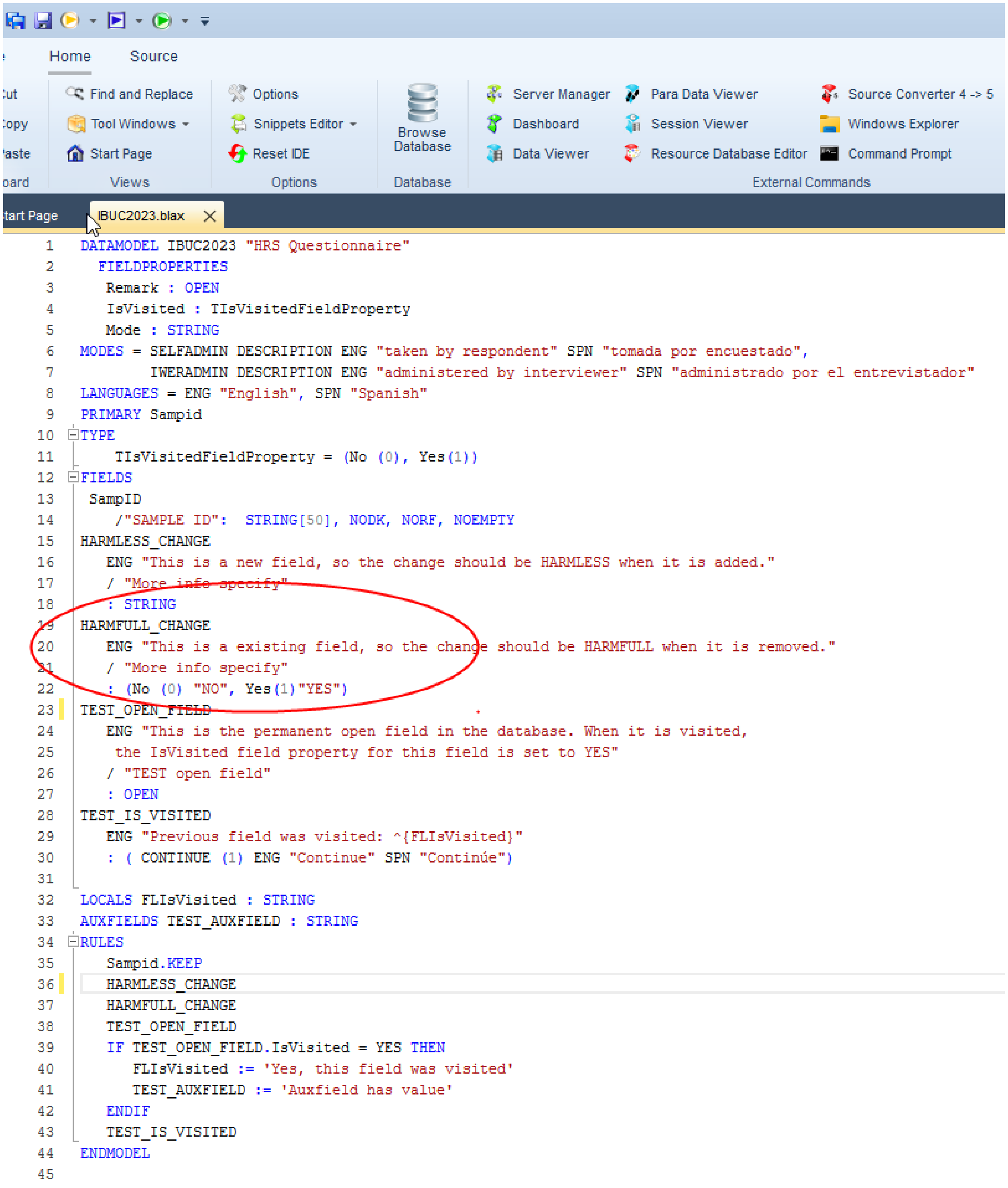


Language : ENG
Settings : CAWI_Settings
Parallel :
Active Field : TEST_IS_VISITED
User Name : Anonymous

F SampID='123'
F HARMLESS_CHANGE='This is the added field'
F HARMFULL_CHANGE='This is the existing field'
F TEST_OPEN_FIELD='Type something here or don't'
A TEST_AUXFIELD='Auxfield has value'

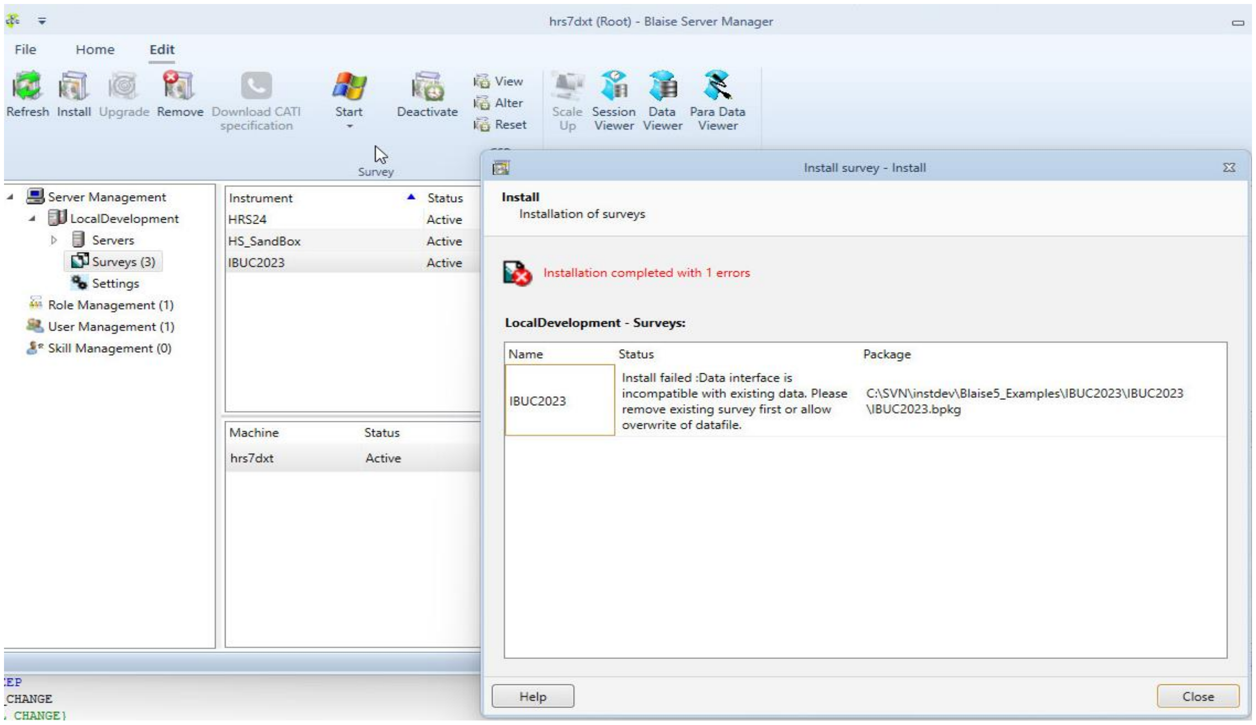
Primary Key
123

Now, suppose there is need to change a field type in the survey that is already deployed. Here, we are changing the type from OPEN to ENUMERATION for HARMFULL_CHANGE field.



```
DATAMODEL IBUC2023 "HRS Questionnaire"
  FIELDPROPERTIES
   Remark : OPEN
   IsVisited : TIsVisitedFieldProperty
   Mode : STRING
MODES = SELFADMIN DESCRIPTION ENG "taken by respondent" SPN "tomada por encuestado",
        IWERADMIN DESCRIPTION ENG "administered by interviewer" SPN "administrado por el entrevistador"
LANGUAGES = ENG "English", SPN "Spanish"
PRIMARY Sampid
TYPE
    TIsVisitedFieldProperty = (No (0), Yes(1))
FIELDS
 SampID
    /"SAMPLE ID":  STRING[50], NODK, NORF, NOEMPTY
HARMLESS_CHANGE
   ENG "This is a new field, so the change should be HARMLESS when it is added."
   / "More info specify"
   : STRING
HARMFULL_CHANGE
   ENG "This is a existing field, so the change should be HARMFULL when it is removed."
   / "More info specify"
   : (No (0) "NO", Yes(1)"YES")
TEST_OPEN_FIELD
   ENG "This is the permanent open field in the database. When it is visited,
    the IsVisited field property for this field is set to YES"
   / "TEST open field"
   : OPEN
TEST_IS_VISITED
   ENG "Previous field was visited: ^{FLIsVisited}"
   : ( CONTINUE (1) ENG "Continue" SPN "Continúe")

LOCALS FLIsVisited : STRING
AUXFIELDS TEST_AUXFIELD : STRING
RULES
   Sampid.KEEP
   HARMLESS_CHANGE
   HARMFULL_CHANGE
   TEST_OPEN_FIELD
   IF TEST_OPEN_FIELD.IsVisited = YES THEN
      FLIsVisited := 'Yes, this field was visited'
      TEST_AUXFIELD := 'Auxfield has value'
   ENDIF
   TEST_IS_VISITED
ENDMODEL
```

Since the type of the field had been changed, we were unable to install the survey over the existing one without overwriting the old session data. An attempt to do so will result in the following error message:



Instrument	Status
HRS24	Active
HS_SandBox	Active
IBUC2023	Active

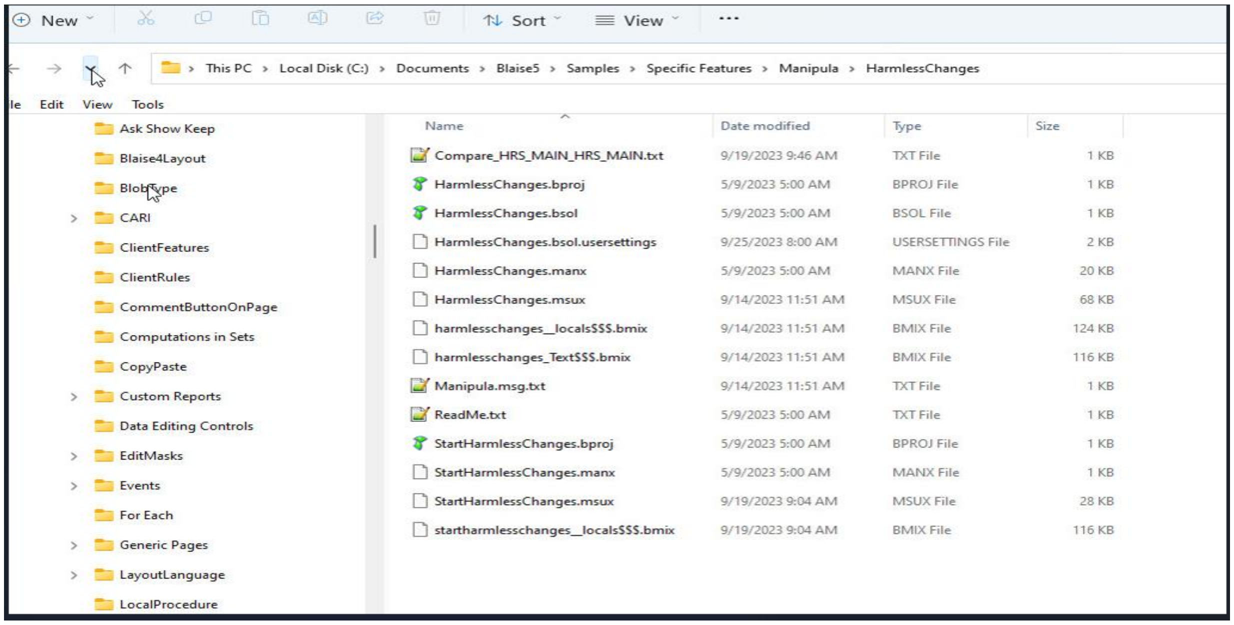
Machine	Status
hrs7dxt	Active

Installation completed with 1 errors

LocalDevelopment - Surveys:

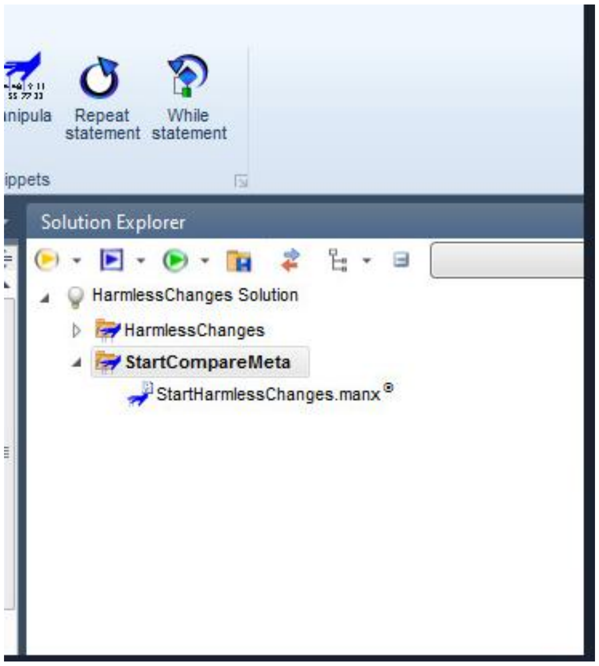
Name	Status	Package
IBUC2023	Install failed :Data interface is incompatible with existing data. Please remove existing survey first or allow overwrite of datafile.	C:\SVN\instdev\Blaise5_Examples\IBUC2023\IBUC2023\IBUC2023.bpkg

Given the complexity of HRS and the number of datamodels released into the field after the data collection has started, we always run a harmless change tool to check for harmful changes. This useful tool provided by the Blaise team is located here:

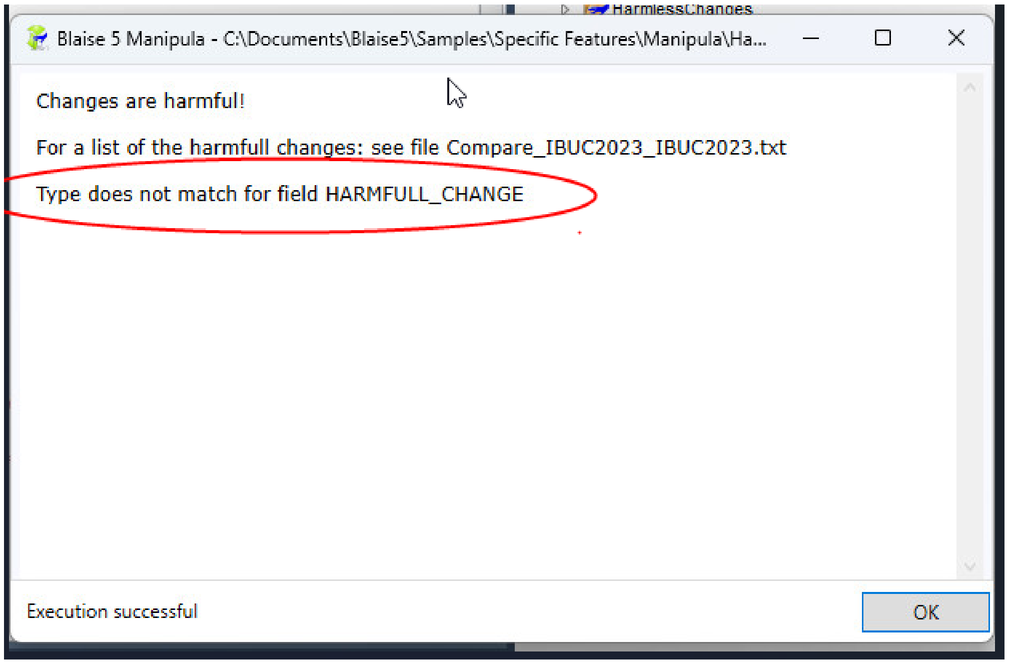


Name	Date modified	Type	Size
Compare_HRS_MAIN_HRS_MAIN.txt	9/19/2023 9:46 AM	TXT File	1 KB
HarmlessChanges.bproj	5/9/2023 5:00 AM	BPROJ File	1 KB
HarmlessChanges.bsol	5/9/2023 5:00 AM	BSOL File	1 KB
HarmlessChanges.bsol.usersettings	9/25/2023 8:00 AM	USERSETTINGS File	2 KB
HarmlessChanges.manx	5/9/2023 5:00 AM	MANX File	20 KB
HarmlessChanges.msux	9/14/2023 11:51 AM	MSUX File	68 KB
harmlesschanges__locals$$$.bmix	9/14/2023 11:51 AM	BMIX File	124 KB
harmlesschanges_Text$$$.bmix	9/14/2023 11:51 AM	BMIX File	116 KB
Manipula.msg.txt	9/14/2023 11:51 AM	TXT File	1 KB
ReadMe.txt	5/9/2023 5:00 AM	TXT File	1 KB
StartHarmlessChanges.bproj	5/9/2023 5:00 AM	BPROJ File	1 KB
StartHarmlessChanges.manx	5/9/2023 5:00 AM	MANX File	1 KB
StartHarmlessChanges.msux	9/19/2023 9:04 AM	MSUX File	28 KB
startharmlesschanges__locals$$$.bmix	9/19/2023 9:04 AM	BMIX File	116 KB

Compile and run HarmlessChange.bsol. You will be asked to provide the locations of the old and the new .bmix files that need to be compared to detect if there are harmful changes.



In our example, the report is generated, warning of the field type mismatch in the datamodels.



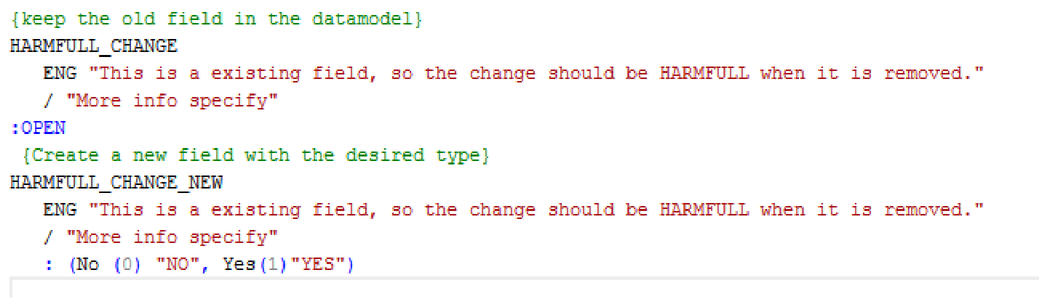
Blaise 5 Manipula - C:\Documents\Blaise5\Samples\Specific Features\Manipula\Ha...
OK

```
Changes are harmful!

For a list of the harmfull changes: see file Compare_IBUC2023_IBUC2023.txt

Type does not match for field HARMFULL_CHANGE
```

Execution successful

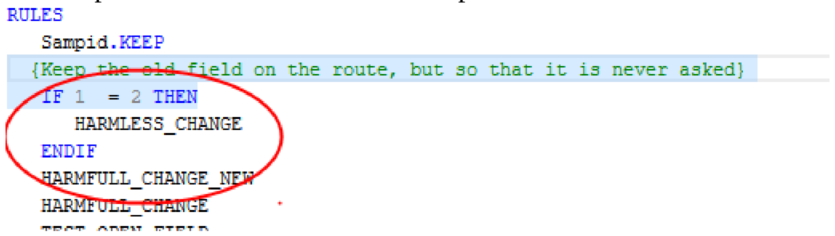
To remedy the problem, HRS uses the following techniques:

1. We create a new field that has the desired type without commenting out the old one.



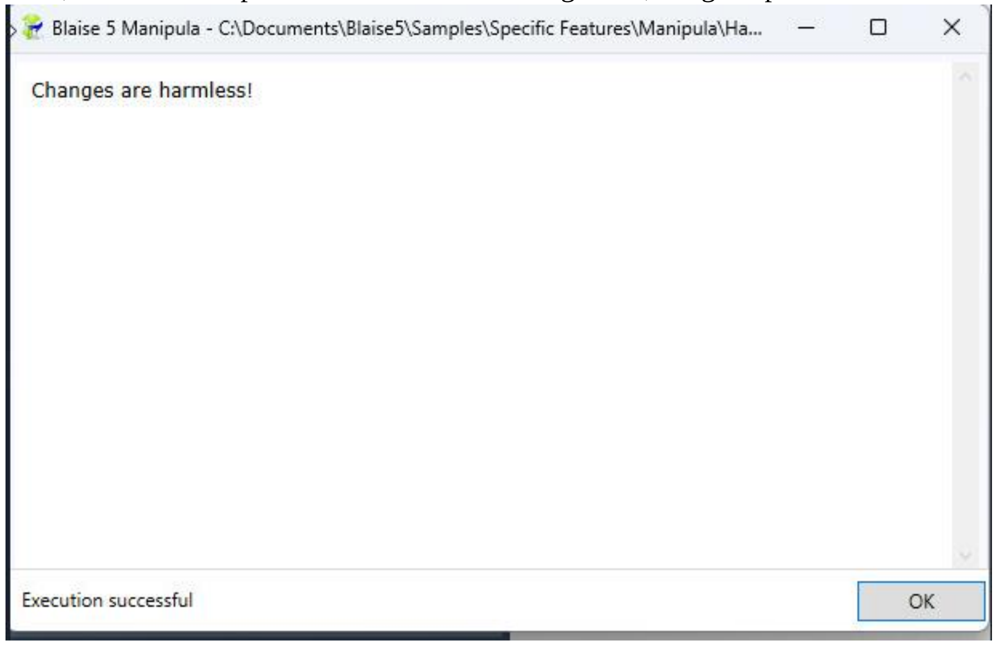
```
{keep the old field in the datamodel}
HARMFULL_CHANGE
   ENG "This is a existing field, so the change should be HARMFULL when it is removed."
   / "More info specify"
:OPEN
 {Create a new field with the desired type}
HARMFULL_CHANGE_NEW
   ENG "This is a existing field, so the change should be HARMFULL when it is removed."
   / "More info specify"
   : (No (0) "NO", Yes(1)"YES")
```

2. We keep the old field on the route but put an IF-THEN statement that is never TRUE around it.



```
RULES
   Sampid.KEEP
  {Keep the old field on the route, but so that it is never asked}
   IF 1  = 2 THEN
      HARMLESS_CHANGE
   ENDIF
   HARMFULL_CHANGE_NEW
   HARMFULL_CHANGE
```

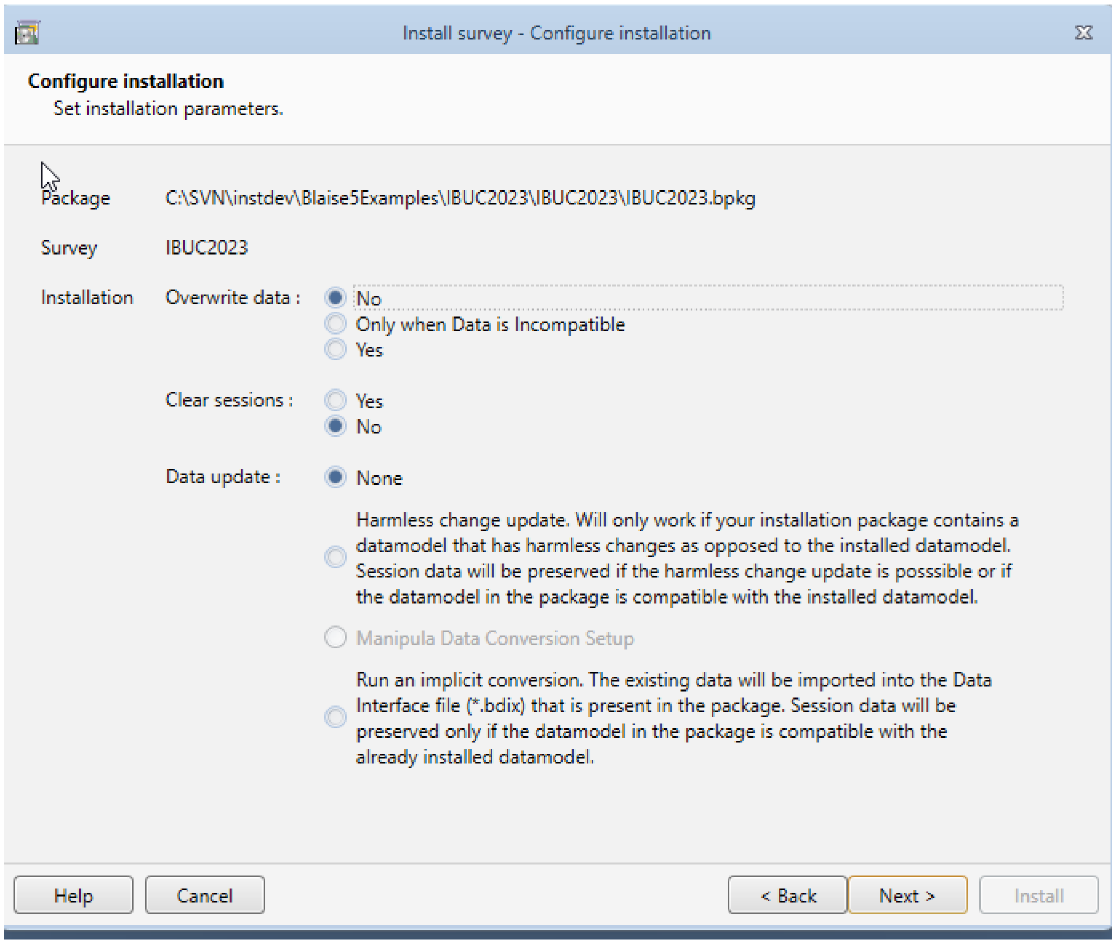
Now, when we compile and run HarmlessChange.bsol, we get a positive result!


Blaise 5 Manipula - C:\Documents\Blaise5\Samples\Specific Features\Manipula\Ha...
OK

```
Changes are harmless!
```

```
Execution successful
```

When we install the new survey over the old one, we use the following options to preserve the session data:



Configure installation
Set installation parameters.

Package C:\SVN\instdev\Blaise5Examples\IBUC2023\IBUC2023\IBUC2023.bpkg

Survey IBUC2023

Installation

Overwrite data : [✓ No] Only when Data is Incompatible Yes

Clear sessions : Yes [✓ No]

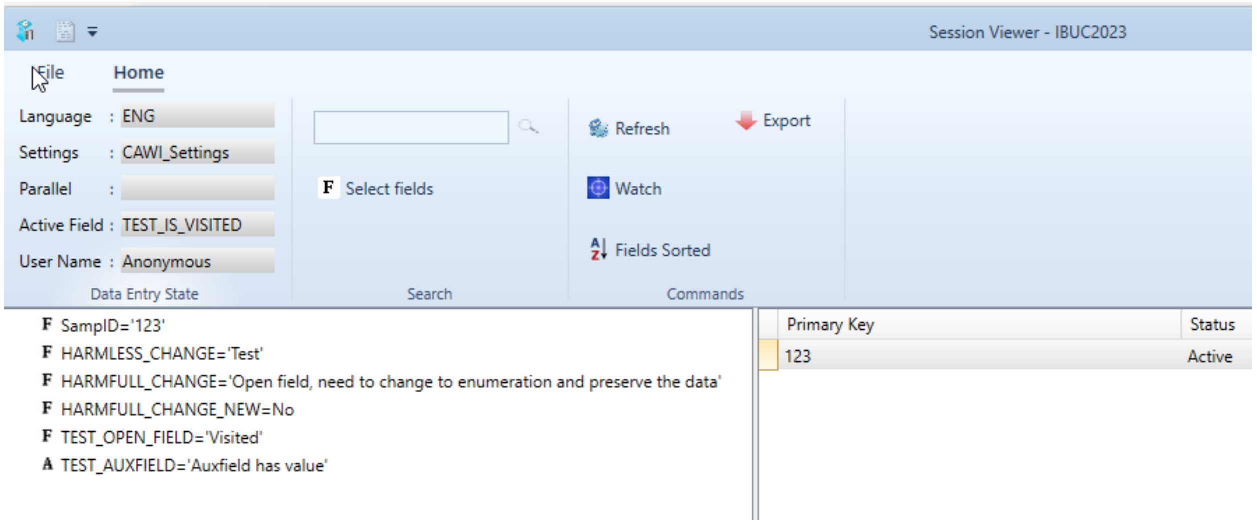
Data update : [✓ None]

Harmless change update. Will only work if your installation package contains a datamodel that has harmless changes as opposed to the installed datamodel. Session data will be preserved if the harmless change update is possible or if the datamodel in the package is compatible with the installed datamodel.

Manipula Data Conversion Setup

Run an implicit conversion. The existing data will be imported into the Data Interface file (*.bdix) that is present in the package. Session data will be preserved only if the datamodel in the package is compatible with the already installed datamodel.

When viewing the session data for the new survey, we can see that the old session data is preserved.



Language : ENG
Settings : CAWI_Settings
Parallel :
Active Field : TEST_IS_VISITED
User Name : Anonymous

F SampID='123'
F HARMLESS_CHANGE='Test'
F HARMFULL_CHANGE='Open field, need to change to enumeration and preserve the data'
F HARMFULL_CHANGE_NEW=No
F TEST_OPEN_FIELD='Visited'
A TEST_AUXFIELD='Auxfield has value'

Primary Key	Status
123	Active

11. Appendix—Source Code

```
DATAMODEL IBUC2023 "HRS Questionnaire"
  FIELDPROPERTIES
            Remark : OPEN
            IsVisited : TIsVisitedFieldProperty
            Mode : STRING
MODES = SELFADMIN DESCRIPTION ENG "taken by respondent" SPN "tomada por encuestado",
                IWERADMIN DESCRIPTION ENG "administered by interviewer" SPN "administrado por el
entrevistador"
LANGUAGES = ENG "English", SPN "Spanish"
PRIMARY Sampid
TYPE
    TIsVisitedFieldProperty = (No (0), Yes(1))
FIELDS
 SampID
    /"SAMPLE ID": STRING[50], NODK, NORF, NOEMPTY
HARMLESS_CHANGE
            ENG "This is a new field, so the change should be HARMLESS when it is added."
            / "More info specify"
            : STRING
{keep the old field in the datamodel}
HARMFULL_CHANGE
            ENG "This is a existing field, so the change should be HARMFULL when it is removed."
            / "More info specify"
:OPEN
 {Create a new field with the desired type}
HARMFULL_CHANGE_NEW
            ENG "This is a existing field, so the change should be HARMFULL when it is removed."
            / "More info specify"
            : (No (0) "NO", Yes(1)"YES")

TEST_OPEN_FIELD
            ENG "This is the permanent open field in the database. When it is visited,
    the IsVisited field property for this field is set to YES"
            / "TEST open field"
            : OPEN
TEST_IS_VISITED
            ENG "Previous field was visited: ^{FLIsVisited}"
            : ( CONTINUE (1) ENG "Continue" SPN "Continúe")

LOCALS FLIsVisited : STRING
AUXFIELDS TEST_AUXFIELD : STRING
RULES
   Sampid.KEEP
   HARMLESS_CHANGE
  {Keep the old field on the route, but so that it is never asked}
```

```
   IF 1 = 2 THEN
      HARMFULL_CHANGE
  ENDIF
   HARMFULL_CHANGE_NEW
   TEST_OPEN_FIELD
   IF TEST_OPEN_FIELD.IsVisited = YES THEN
      FLIsVisited := 'Yes, this field was visited'
      TEST_AUXFIELD := 'Auxfield has value'
   ENDIF
   TEST_IS_VISITED
ENDMODEL
```