Adventures of a Blaise 5 API jockey

Rod Furey, Statistics Netherlands

1. Abstract

The Blaise 5 Application Programming Interface (API) can be used to access the various underlying objects of the Blaise 5 system in various programming languages. This allows us to manipulate and present these objects to our own advantage. This paper discusses various cases where Blaise 5 API calls have been used in production systems and testing tools.

2. Manga

Long, long ago, back in the mists of time, Manipula for Blaise 5 didn't exist. Alas there was a requirement from a client who wanted to be able to read incoming data and write it to a Blaise 5 file (.bdbx).

The obvious solution for this was to write a simple program which would let someone give the names of the input file and output file as parameters and then do the work for them, but there's no fun in that. Given that Manipula was at that time a long way off, a more generic solution was needed and so Manga was born.

2.1 Manga v1

Manga has had various incarnations. The original variant for the client handled basic export functions and some displays using a syntax that is familiar to anyone who has used Manipula:

Listing 1. Manga Source Code

```
Manga BlaiseToAscii 'Blaise to Ascii'
USES
    datamodel flight 'Flight'
ENDUSES
INPUTFILES
    iBlaiseFlightIn : flight : BLAISE : "C:\CBSTEMP\USER\Flight.bdbx"
ENDINPUTFILES
OUTPUTFILES
  iAsciiFlightOut  : flight : ASCII
ENDOUTPUTFILES
UPDATEFILES
ENDUPDATEFILES
AUXFIELDS
    result : resultok
ENDAUXFIELDS
ELEMENTS
ENDELEMENTS
BODY
// Output an indication that we've started...
conswriteline ("Copying existing Blaise file contents to new Ascii
file...")

// We don't have the appropriate files for the Ascii file at the moment so
// we need to generate them (this sets up the fn.txt and fn.bdix files).
result := iAsciiflightOut.Create("C:\CBSTEMP\USER\FlightOut.txt")
result := BlaiseToAscii(iBlaiseFlightIn,iAsciiflightOut)

consolewriteline ("Finished!")
ENDBODY
```

There were (and still are) a number of restrictions on this, mostly due to the parser. However, it did the job. In this case an API call is used to create the Blaise Interface (.bdix) file. This is followed by the appropriate API calls to the DataLink interface to copy the data from one file to another.

Obviously if Blaise to ASCII is achievable, so is ASCII to Blaise. Equally, given a .bdix that points to an XML file, that is also available for copy actions. Even using SQL Server as a backing store for the Blaise 5 data works.

2.2 Manga v2

Armed with some basic copy actions, a decision was taken to add some extra operations. Having written the first version of Manga as an interpreter, the decision was made to write a compiler for version 2. This version contained a first pass at an expression parser, an arithmetic expression evaluator and a string concatenator.

Implemented as a two-pass compiler, the first pass decomposed the Manga source and generated a list of macro calls and parameters to handle such things as string concatenation, branching, displaying items, arithmetic operations etc. This list of macros was then read and the appropriate Common Intermediate Language (CIL) statements written away to a file which was later fed (by hand) into the Intermediate Language assembler (ilasm.exe). The resulting executable handled these extra functions with aplomb.

2.3 Manga v3

The next step was to add some basic manipulation of the Blaise fields. Given that debugging this was going to be easier if an interpreter was used, the decision was made to adapt the work from Manga v2 (parser, evaluator etc.) so that it could be used in the interpreter from Manga v1.

To facilitate the reading of Blaise fields, a series of API calls was used to open the compiled datamodel file (.bmix) and retrieve the names and types of the various fields. This resulted in some interesting discussions with the Blaise group about the correct call to use and why some of them didn't work as advertised. Coding problems on both sides were analyzed and bugs fixed with the result that read, write/update and delete functions could be carried out via the API calls which retrieved the field, read or assigned its value and read, wrote/updated or deleted a record from the file.

2.4 Manga v4

The final step in the evolution of Manga has been to add boolean algebra and loops to the interpreter.

2.5 Results

Manga has helped a couple of projects a number of times with regards to extracting data from Blaise 5 data sources and writing that data to disk for analysis or debugging purposes. It has also caught a number of bugs in the Blaise 5 codebase which have subsequently been corrected. Manga is still used today as an extended test case for new releases of Blaise 5 in order to trap any regressions that may have occurred in the various API calls that it uses.

3. Login

The Production Statistics questionnaire was chosen for Statistics Netherlands' first production Blaise 5 questionnaire. This questionnaire is protected by a login gateway which uses a userid and password pairing for identification. This gateway basically reflects the login sample which is given in the Blaise 5 help but has to implement some extra requirements from the business. Two of these requirements are:

- do not allow more than one person to be logged in at once

- allow someone else from the respondent's company to "steal" the login after an hour of inactivity

The login questionnaire utilizes an ALIEN call to a C# module in order to satisfy these conditions. When someone logs in, a flag is set in an SQL Server database. This flag is checked on every login attempt to see whether or not login is allowed. A call to the Blaise 5 Session API is made which retrieves any session information for the appropriate primary key of the questionnaire. If any session data exists then a check is made on the <*session*>.Creation property and the <*session*>.LastModification property to see if they were more than one hour ago. If this is the case the new login attempt is flagged as being allowable (it must pass other requirements as well before being carried out).

Stripping out all the exception and error handling, the code for this check effectively simplifies to:

Listing 2. Login Source Code

```
using SDataAPI = StatNeth.Blaise.API.SessionData;
...
SDataAPI.IInstrumentSessionInfo _isi =
    SDataAPI.SessionDataManager.GetInstrumentSessionInfo(
        Guid.Parse(<instrument-id-of-associated-instrument>),
        <server-park-name>
    );

SDataAPI.ISessionInfo _session = null;

// Get the datamodel (is needed to build the primary key):
MetaAPI.IDatamodel dm =
    MetaAPI.MetaManager.GetDatamodel(<path-to-the-appropriate-datamodel>);

// Get an IKey interface for the primary key:
DRecAPI.IKey _primaryKey =
    DRecAPI.DataRecordManager.GetKey(dm,
                                     MetaAPI.Constants.KeyNames.Primary);

// The key consists of just 1 field, userid:
_primaryKey.Fields[0].DataValue.Assign(<userid>);

_session = _isi.Read(_primaryKey);

if (_session == null)
{
    _rc = true;
}
else
{
    if ((_session.Creation.AddHours(1d) <= DateTime.Now)
    && (_session.LastModification.AddHours(1d) <= DateTime.Now))
    {
        _rc = true;
    }
    else
    {
        _rc = false;
    }
}
...
return _rc;
```

4. BISDar

One of the items that has come up on the "Wouldn't it be nice if...?" list is a route checker. There are times when fields disappear off the display or are set to unexpected values when running a questionnaire. Despite extensive pre-release testing, a respondent may fill in a combination of answers that manages to produce an edge case. A structured test method would be nice to have.

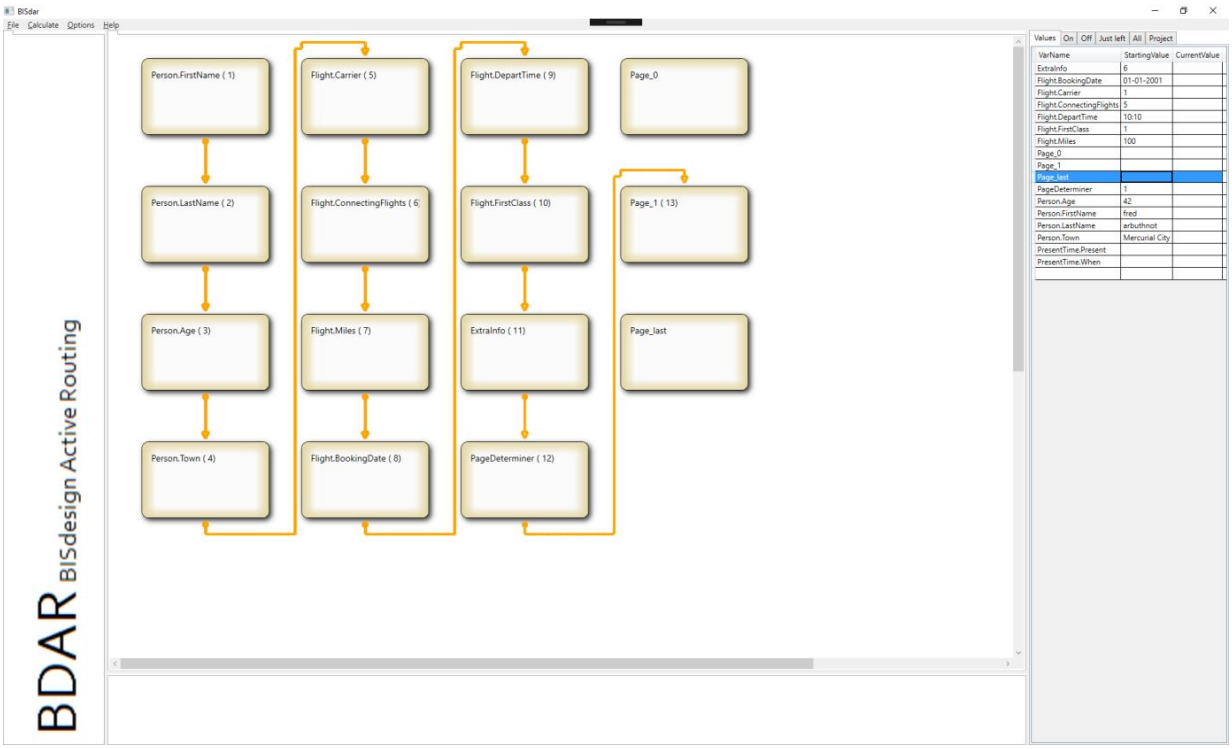
To add an extra layer of testing, BISdar was written. BISdar uses the Blaise 5 API to not only extract the fields and their definitions but also which page the field would be expected to appear on (work in progress; an API call to ensure that this is the same order as the preview has been requested but in the meantime, a small program has had to be written). This is then presented to the questionnaire designer who then has the ability to set various values for the fields and then execute the rules (again, via calls to the Blaise 5 API).

Recently a new display option has been added. Instead of displaying the fields on a page by page basis, the fields are displayed individually and the current route through the questionnaire is shown by drawing arrows between the display boxes of the fields that are on the route. This is far more useful and the original drawing function has now been deprecated.

After the rules have been executed, the status of each field is read and an indication shown in various list views as to whether it is on the route, has just gone off the route in the last execute, or has been taken off the route before that. This latter information is kept locally in the program.

Sets of values can be saved and loaded back and the project saved to disk.

Figure 1. BDAR

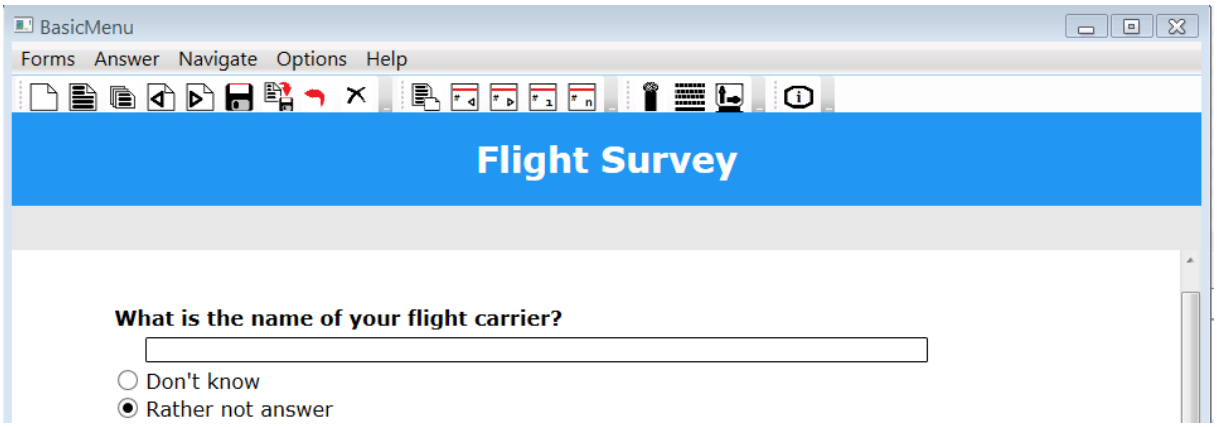


5. BasicMenu

Back in the Blaise 4.8 Data Entry Program (DEP) there is a menu that lets people execute various actions such as displaying all remarks or entering a value of Refusal, amongst others. This menu bar isn't in Blaise 5. The solution? Create a custom WPF DEP and a custom control and use the DataEntry API to recreate the menu bar.

The Data Entry framework utilises the DataEntry, DataEntryWpf, DataEntry.Controls and DataEntry.DataObjects DLLs. The processing code uses the DataEntry, DataLink, DataRecord, Meta, SessionData and ServerManager DLLs to, amongst other things, interrogate the compiled datamodel and session record, execute actions, read records, retrieve the fields on the route, assign values to fields and talk to the server park.

Figure 2. BasicMenu Custom Dep



6. Conclusion

The Blaise 5 Application Programming interface allows access to items from the Blaise 5 milieu from various languages. As discussed, these facilities allow the programmer to create various tools or even other languages to help solve problems in their domain.

It should be noted however, that Blaise 5 is in some respects vastly different in its implementation details from Blaise 4. This affects, for example, the menu bar in subtle and not so subtle ways. One example of this is that the instrument to be queried is expected to be installed in the server park. A bigger difference is that the system-wide implementation of a session means that when navigating between forms, the session information comes back as well which can mean that, for example, the language that is in use in the session can differ from the language that was last set by the user.

However, once these variations in behaviour are taken into account, designing tools such as those discussed becomes a simple matter of programming.

7. And finally...

My serious introduction to Blaise occurred a while ago when I was hired in to work closely with Gerrit de Bolster whose enthusiasm for and knowledge of Blaise is contagious. It has been as pleasure and a privilege to work with him these last 4 years and without him I wouldn't be presenting at the IBUC.