Using the Blaise Component Pack from Within the .NET Framework to Develop Data Management Tools

Leonard Hart, Robert Thompson & John Mamer (Mathematica Policy Research, USA)

1. Introduction

Mathematica Policy Research, Inc. (MPR), has been conducting complex surveys for the United States federal government and U.S.-based foundations for over 35 years. Early surveys were conducted either in person or by telephone and generally were captured as hard-copy documents for data entry. Complex routing, rostering, and grid-based data collection were often a part of these surveys, thus the automated checking and control that became available with the advent of computer-assisted interviewing (CAI) was essential to increasing the quality of the data collected. MPR adopted computer-assisted telephone interviewing (CATI) tools in the early 1980s and computer-assisted personal interviewing (CAPI) tools later that decade. More recently, Web surveys have become an important aspect of our data collection activities. MPR currently uses the Blaise platform for most of its CATI, CAPI, and Web surveys.

To add to the difficulty of fielding these often long and complex surveys, many of MPR's surveys are longitudinal in nature and involve mobile, hard-to-locate populations. Automated support for sample management activities is essential in this environment. External to Blaise, MPR's Sample Management System (SMS) provides facilities for locating and tracking. The SMS, developed in Microsoft's .NET platform, usually operates simultaneously with Blaise in a production environment, and data are exchanged between the two systems in an overnight process.

Finally, data and sample management within the Blaise platform itself—beyond what is provided inherently in the Blaise call scheduler—is essential when dealing with our target populations so we may achieve the high response rates required in nearly all of our studies. Among the first tools we developed for our CATI operation was a Supervisory Review utility that allowed supervisors to review and re-status survey instruments in a very granular fashion. The first sections of this paper describe the evolution of the Supervisory Review facility from its initial implementation in Maniplus to its current version, renamed as the MPR Blaise Explorer and implemented in the .NET environment using the Blaise Component Pack (BCP). The next sections discuss the technical and performance issues we encountered in this effort. We end with conclusions based on this experience.

2. Supervisory Review Utility

MPR has called the tool built to handle the sample management tasks within the Blaise environment the "Supervisory Review" utility. In this section we provide an overview of the utility and discuss its implementation in Maniplus.

2.1. Overview of Supervisory Review Facility

The following paragraphs describe three of the primary functions provided in the Supervisory Review utility. While more functionality is implemented in the utility, this description provides a general overview of its basic purpose and functioning.

Reviewing Classes of Cases. One of the primary functions of the Supervisory Review utility is to provide the supervisors with the ability to display classes of cases for review. The menu below illustrates the ways in which supervisors may subset cases. They may extract cases by individual or ranges of primary keys (MPRID), final and current disposition, dates of interviews, and more. For example, the selection below will choose all cases with status code 380 administered between February 15th and 21st that are in release 4 and 5.

Select cases

	BEGIN	END
MPRID		
Final		
CDSP	380	
Code		
To Whom		All / Empty group only
Date	2/15/2006	2/21/2006
Attempts		
Sup Review		
Release	4	5
Stratum		
Gender		
Copy History		no / yes

Selection Exit

Cases are then displayed with a variety of options for looking at the case in more detail. The screen below lists 12 cases that fit the criteria.

Access And Action Screen for KFS

MPRID	N	Final	CDSP	Code	User ID	R	Group	Last date	B T
10276837	2	·	380	380	52929	5	REFUSAL	2006/02/16	
10181061	14	·	380	380	54120	4		2006/02/15	
10191721	2	·	380	380	52929	5	REFUSAL	2006/02/16	
10191802	4	·	380	380	52929	5	REFUSAL	2006/02/16	
10176827	10	·	380	380	50467	4		2006/02/20	
10268861	6	·	380	380	56014	5		2006/02/18	
10257115	2	·	380	380	55049	5	REFUSAL	2006/02/20	
10174612	6	·	380	380	50568	4	REFUSAL	2006/02/20	
10174638	10	·	380	380	52929	4	REFUSAL	2006/02/16	
10314265	4	·	380	380	53002	5		2006/02/20	
10232499	8	·	380	380	53002	5		2006/02/20	
10276390	2	·	380	380	52929	5	REFUSAL	2006/02/16	

1:12

History · Details · Read Only · Do Case · Project · Finalize · Interim · Sup Review

Type MPRID

Sort Search Exit

Once the cases are displayed in this fashion, the supervisor may look at individual cases in a variety of ways, as indicated by the buttons on the right hand side of the screen. For instance, supervisors may look at the history of the case, may follow the logic of the interview in read-only mode, may load the case as a live interview with “Do Case,” or may re-status a case as Final or Interim status. While this capability is quite powerful, the choice of fields in the *Select cases* dialog is relatively static and not under the control of the operational personnel.

Communicating with Interviewers on Specific Cases. Interviewers can communicate special situations that they have encountered on a case by adding notes to the case while administering the survey and putting the case in Supervisory Review status. Using the facility we described in the previous section, supervisors can pull up all cases that have a Supervisory Review status, read the notes, and then handle the case as appropriate, given the information they see in the notes and other aspects of the case history. In some instances this may involve changing a case’s status, as described in the next section. In other instances this may involve writing notes back to the interviewer and placing the case back in the interviewing pool.

Changing Case Statuses. Within the Supervisory Review utility, supervisors may change the status of a case depending on the circumstances presented in the case history and the notes. Often this involves final statusing a case, but a number of other options are available. For example, on many surveys we have a survey instrument set up so that a case that has been refused twice will be automatically changed to Supervisory Review status. Within the Supervisory Review utility a supervisor may decide to send the case back to the interview pool so that it is handled by a refusal conversion interviewer. Alternatively, the supervisor may look at interviewer notes and see that the respondent has threatened a lawsuit if he or she is called again—a situation that has occurred more than once—and in this instance the case would be final statused as a Refusal and not called again.

2.2. Implementation of Supervisory Review in Maniplus

Early in this decade, MPR was in the process of changing its survey platform from CASES, a product of the University of California at Berkeley, to Blaise. Having used CASES for almost twenty years, MPR had developed a wide variety of tools to support CASES surveys. After several early Blaise efforts with relatively straightforward surveys, in 2003 we embarked on a multi-mode, worldwide survey of college graduates for the National Science Foundation. Some sort of Supervisory Review utility was essential for this complex multi-mode survey and the lead time was short. Given the tight integration of Maniplus with the Blaise data model and all other aspects of the Blaise platform, we deemed implementation of the Supervisory Review utility in Maniplus as the only feasible way of having the utility available in the required time frame.

A closely related reason for developing the utility in Maniplus was the availability of staff resources with the appropriate skills. In 2003, MPR was also beginning a transition to the .NET environment and the SQL Server database platform for all its major applications. While we had a number of proficient .NET and SQL Server programmers on staff, none of them had experience in the operational requirements of fielding a complex survey. This would have presented a steep learning curve had we embarked immediately on the task of implementing a Supervisory Review utility in .NET. At the same time we had a number of staff who understood surveys well and either knew Maniplus or could learn Maniplus with a relatively short learning curve. Both these reasons combined to make Maniplus the logical choice for getting the features we needed available in the requisite time frame.

2.3. Reasons for Change

As a consequence of the decision-making process outlined in the previous section, we believed from the beginning that a Supervisory Review utility implemented in Maniplus could be viewed as a prototype—albeit a very useful one—for the functionality that eventuality would be incorporated in a .NET application. As expected when Supervisory Review was first written in Maniplus, we have added more .NET trained staff that over time have became more familiar with survey requirements. The MPR Sample Management System (SMS), mentioned in the Introduction, became fully developed over the last several years using the .NET platform and a SQL Server backend. Hence, a Supervisory Review utility implemented in .NET that communicated to the Blaise database via the Application Programmer Interface (API) using the Blaise Component Pack could also be tightly integrated with the SMS.

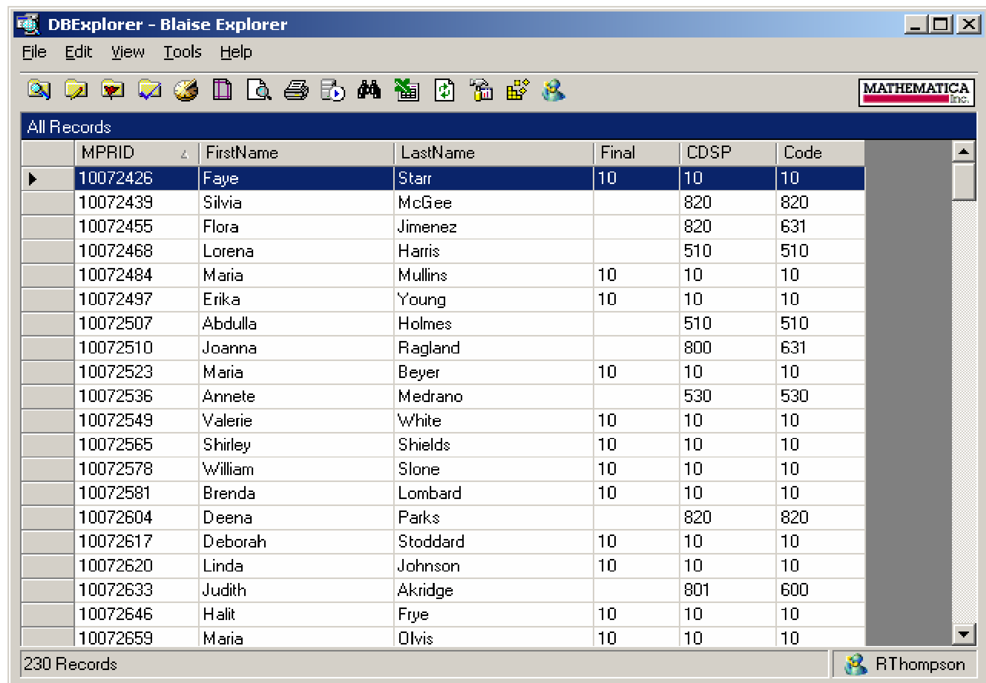
Another goal in the transition to a new system was to provide user authentication for a wide range of facilities. In order to grant access to the facilities deemed appropriate for a given staff member in a given position, we wanted to provide login authentication mechanism that would allow a user to access just those facilities.

Finally, in terms of user interfaces, for all our applications at MPR, we have tried to maintain the look and feel of Microsoft applications to ease user startup and training. Moving from Maniplus to .NET as an implementation platform allowed us to achieve this goal for the Supervisory Review facility.

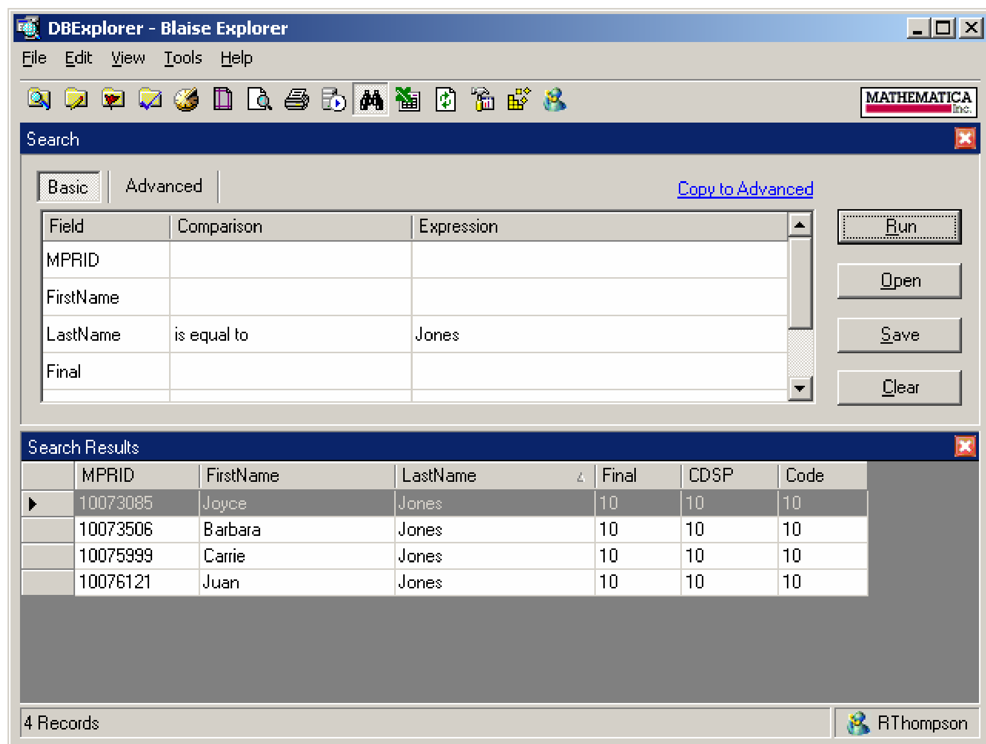
3. Blaise Explorer

"Supervisory Review" was a limiting name for the facilities we hoped to provide in the new version of this utility. As we moved to the .NET platform we planned, and still plan, to extend the functions embedded in the application beyond those that might be deemed supervisory in nature. As noted, the software would provide user authentication that allows authorized users access to the facilities they need. Because of the new functionality envisioned for the software we renamed the application the MPR Blaise Explorer.

On startup of MPR Blaise Explorer, the user is presented with the screen below, which lists all the cases. The new tool has the look and feel of a Microsoft application. In addition, supervisors at the call center can display useful data without getting a programmer involved to change the screen. Under the Supervisor Utility written in Maniplus, if a supervisor wanted to display different data on its main screen, a programmer needed to make the changes.

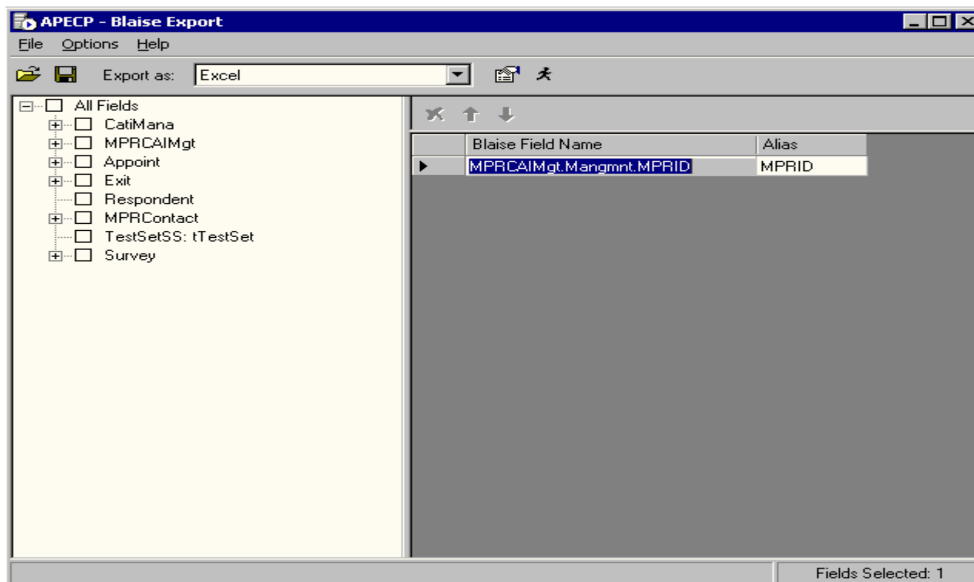


The search facility, which allows the user to subset the cases as desired, is denoted as in most Microsoft applications with the "binoculars" icon on the toolbar, or the user may use the menu bar. Invoking the search facility displays the following screen:

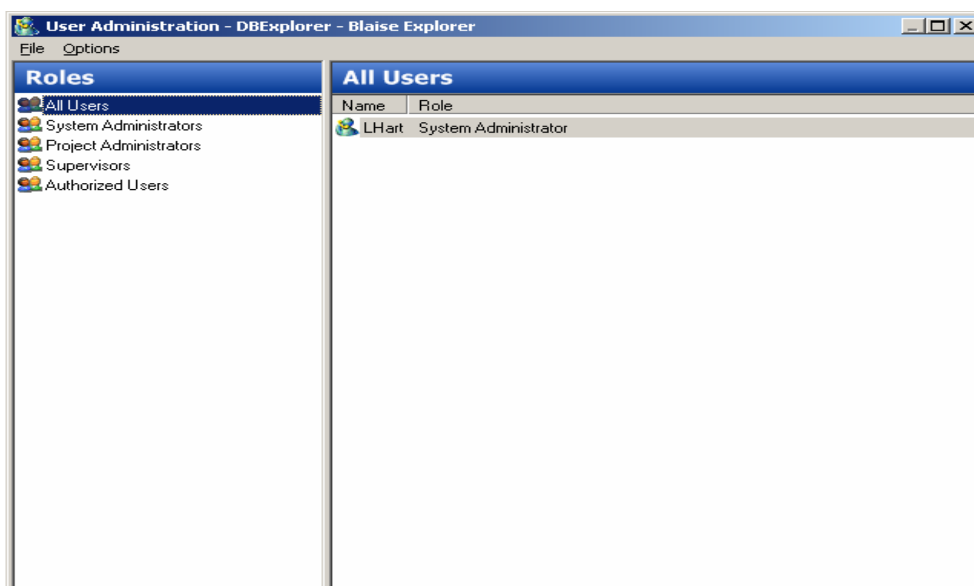


This screen allows the user to do many of the "case subsetting" functions of the Supervisory Review utility implemented in Maniplus but expands on those capabilities greatly. In the toolbar at the top of the screen are icons that allow the user to perform some of the activities on the individual cases such as display case history. As in most standard Windows applications, these activities are also made available though the main menu as well as right-click, context-sensitive popup menus. Reproducing Windows functions in this way allows users to work with the application in the manner that they are accustomed to.

One of the functions embedded in Blaise Explorer, which expands on and distinguishes Blaise Explorer from its Supervisory Review predecessor, is Blaise Export. We should note that due to the application's underlying architecture, Blaise Export can also be invoked as a standalone application in its own right and can be used by those not needing or requiring the full capabilities of Blaise Explorer. This facility displays the familiar Blaise data model "tree" and allows the user to select any parts of this tree to export to another data set for analysis. Thus at any time during a survey, supervisors or other project personnel may export Blaise data in a variety of formats (Excel, SAS XML, tab delimited, or XML) which they may then analyze using the application of their choice. Here is the Blaise Export screen:



Finally, a system administrator is given the facility to administer the system, granting and withholding rights for specific users to gain access to certain aspects of the system through the user administration screen below. This allows us great flexibility in granting specific rights to specific users.



4. Implementation in .NET

As previously noted, MPR adopted the use of the Microsoft .NET Framework as well as the Visual Studio .NET development environment shortly after their initial releases in February 2002. To date and for the most part, that experience has been a positive one. Although the .NET and Visual Studio .NET learning curves are steep, once programmers master them significant productivity gains can be realized.

The current implementation leverages a number of legacy and new technologies that include the Blaise 4.7 API Components, Microsoft Visual Basic 6.0, Microsoft SQL Server 2000, Microsoft .NET 1.1, Microsoft Visual Studio .NET 2003, and Microsoft Internet Information Server. While a solution relying solely on newer technologies would be preferable, the simple reality is that most enterprises cannot abandon their existing investment in legacy technologies, and the .NET Framework provides facilities for leveraging legacy applications.

4.1. Building Reusable .NET Components

While the discussion so far has provided a logical view of the Blaise Explorer and Blaise Export implementations, the physical implementations are markedly different in order to achieve the primary goal of developing a reusable set of .NET components that could be used to meet current and future MPR survey requirements. At the same time, these components were developed with extensibility in mind, to ensure that future requirements could be met with a minimal amount of code rewrite through a plug-in architecture. The core set of components reside in two .NET assemblies: MPR.Blaise.Shared and MPR.Shared. For those unfamiliar with .NET terminology, assemblies are simply the equivalent of executables or dynamic link libraries, and the .NET Framework makes little distinction between the two. MPR.Blaise.Shared is comprised of classes that are specific to Blaise and uses the Blaise 4.7 API. It contains all the Explorer, Export, and Administrator functionality outlined above. MPR.Shared is comprised of generic classes that contain no specific Blaise dependencies and are reusable in a wider variety of MPR applications.

Two additional assemblies, MPR.Blaise.Explorer and MPR.Blaise.Export, serve as standalone .NET applications that use the same MPR.Blaise.Shared assembly. Both executables have small footprints that optimize application start time, an important consideration with the deployment model that is discussed later. Excluding all of the boilerplate code automatically generated by Visual Studio .NET, both of these amount to a few lines of actual substantive code:

```
If MPR.Blaise.Shared.Authentication.Authenticate() Then
   Application.Run(New MPR.Blaise.Shared.Explorer)
End If
```

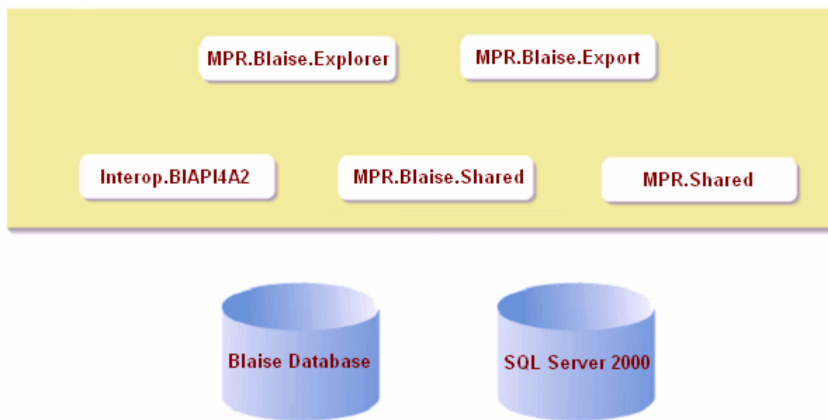
MPR.Blaise.Explorer

```
If MPR.Blaise.Shared.Authentication.Authenticate() Then
   Application.Run(New MPR.Blaise.Shared.Export)
End If
```

MPR.Blaise.Export

As shown here, both applications simply invoke MPR.Blaise.Shared where all of the Blaise specific functionality is located. The same approach can be used for exposing Administrator functionality as a separate application.

While a pure .NET implementation would have been preferable, the reality is that the Blaise 4.7 API is COM (Component Object Model) based. However, it can still be used within the .NET environment using a feature known as .NET COM interoperability. Any practical move toward next generation architectures using the .NET Framework undoubtedly needs to interoperate with existing COM applications, and .NET COM interoperability is the mechanism for leveraging an enterprise's existing COM applications. Using a simple command line utility available with the .NET Framework, a COM interoperability wrapper, Interop.BlAPI4A2, is automatically generated for the Blaise 4.7 API and provides MPR.Blaise.Shared with the ability to both access and update a Blaise dataset. Looking forward, this is an important first step towards eliminating disconnected, fragile overnight processes that are currently used to exchange data between Blaise and the MPR Sample Management System in favor of a more direct real-time data exchange. The initial development efforts have resulted in an architecture that can be extended in straightforward fashion. Currently that architecture can be visualized as shown below and effectively bridges the Blaise environment with the .NET environment:



4.2. Implementing Ad Hoc Blaise Search Capabilities

One of the more frequently recurring requests MPR users have is for specific data that are available within Blaise on an as needed basis. More often than not this requires substantial involvement of a Blaise programmer, whose time may or may not be available to immediately satisfy those requests; and, as we are all aware, users tend to want their data yesterday. One of the key components within the MPR tools that are being developed is the automatic maintenance of an index into a Blaise dataset. That index is maintained within a SQL Server database which provides users with the ability to search for specific Blaise records based on a preconfigured set of Blaise fields.

The figure below depicts a typical index that is maintained within SQL Server:

CaseIndex

	Column Name	Data Type	Length	Allow Nulls
(key)	[MPRCAIMgt.Mangmnt.MPRID]	varchar	8	
	[MPRContact.SampleInfo.FirstName]	varchar	20	✓
	[MPRContact.SampleInfo.MiddleName]	varchar	1	✓
	[MPRContact.SampleInfo.LastName]	varchar	20	✓
	[MPRCAIMgt.Mangmnt.Final]	int	4	✓
	[MPRCAIMgt.Mangmnt.CDSP]	int	4	✓
	[MPRCAIMgt.Mangmnt.Code]	int	4	✓

Note that the SQL Server column names correspond directly to fully qualified Blaise field names. Although these fully qualified column names are unwieldy for typical users, more friendly aliases are displayed to end users throughout the user interface. Using the fully qualified Blaise field names as column names makes the index easier to maintain since there is a direct one-to-one mapping between Blaise and SQL Server. The initial challenge faced was how to maintain this SQL Server index without the need for a disconnected, manual process.

In older versions of Blaise there was a cumbersome process of executing external applications or DLLs through DEP options called alien routers or alien procedures. These were and still can be very useful options, but they have several drawbacks. With the introduction of Version 4.7 of Blaise a new feature called "actions" was introduced ("events" to .NET programmers) that added more flexibility and control to calls outside of the DEP. These actions have given us the control we needed to invoke DLLs at the appropriate time, such as updating a survey management database upon exiting a case in CATI.

MPRBLAPI4A4 is an ActiveX DLL written to the Blaise 4.7 API specification in Visual Basic 6.0. It contains two methods:

```
Update(Database As BlAPI4A2.Database, DepState As BlAPI4A2.DepState)

UpdateEx(ByVal FileName As BlAPI4A2.Field, ByVal Key As BlAPI4A2.Field)
```

Update is invoked by the Blaise Data Entry program in response to an action (such as exiting a case) while UpdateEx is invoked from Maniplus programs. Note that UpdateEx simply invokes Update by creating a BLAPI4A2.Database object and positioning the database to the record specified by the primary key parameter.

The Update method establishes an OLEDB connection to the SQL Server database and updates the index using the configured fields obtained from the Blaise Database that is positioned at the appropriate record. This keeps the SQL Server database in sync with the Blaise record at all times.

The MPR.Blaise.Shared .NET assembly includes classes that are used to configure the Blaise fields that are to be stored in the SQL Server index. Blaise Explorer uses these classes to expose the configuration user interface to authorized users, and the initial startup screen displayed by Blaise Explorer is nothing more than the contents of the SQL Server index table. The search facility allows users to locate records based on any of the configured fields and allows for complex binary searches. In addition, it provides an advanced text-based search that relies on SQL-like comparisons. Search specifications can be saved and later opened to accommodate frequently used searches, or programmers can create even more complex searches, save them and supply them to users. Since the Blaise primary key is part of the index, Blaise records can then be easily located, retrieved, or updated using the Blaise API. Reconfiguring the index to include additional fields

is a non-disruptive, dynamic process. Once reconfigured and the index is rebuilt, new fields become immediately available when the user refreshes the current display. Although the current implementation only provides for a single index per Blaise database, it could easily be extended to encompass multiple indexes per database, thereby providing users with different logical views of the database.

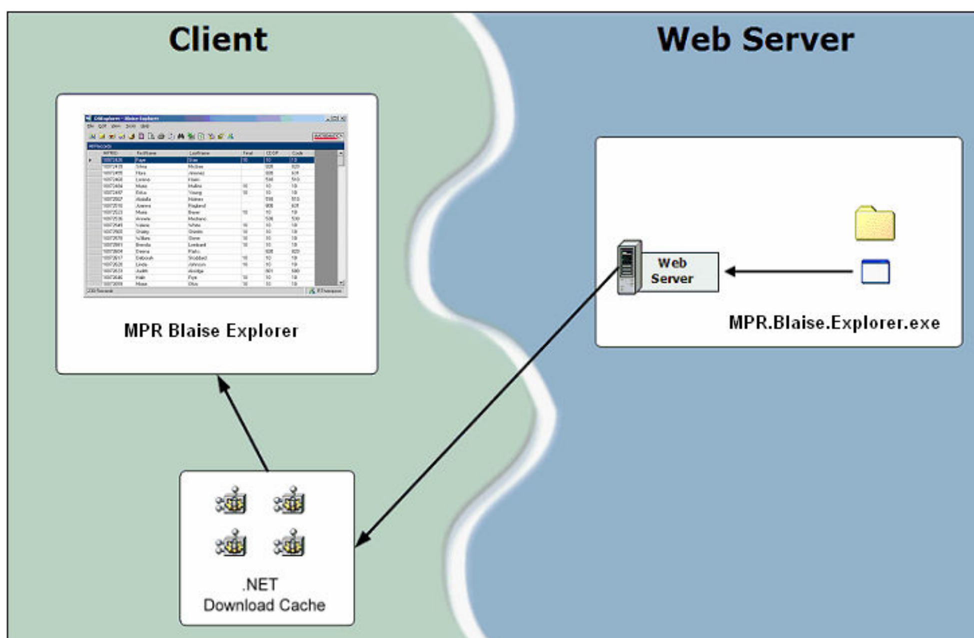
4.3. Deployment

Although not specific to the Blaise .NET integration effort, deploying and maintaining code on the Windows platform has never been an easy task largely due to logistical issues, and browser-based Web applications continue to control the mindshare. When IT managers see that simply updating a Web site automatically gives users the new version, it's hard to get them to go back to the painstaking process of making sure that all desktops in a corporation have been updated to the latest version of a traditional desktop application. However, one undeniable fact remains when it comes to browser-based Web applications: HTML is often inadequate for building fully-featured applications because its user interfaces are, by nature, primitive to use and difficult to implement and maintain because incompatibility between browser versions forces the use of lowest-common-denominator features and therefore the elimination of many user conveniences.

The .NET Framework's no-touch deployment model offers the best of both worlds, enabling richer desktop applications that leverage .NET Windows Forms to be as easily deployed as browser-based Web applications. Just like a Web application, a Windows Forms application can now be deployed on a Web server and launched by merely surfing to a URL (hyperlink) or simply providing the user with a desktop shortcut that points to the application on a Web server.

Deploying new versions of an application is as simple as copying new files to a virtual directory on a deployment server, and the new version is then immediately available to all users the next time they start the application. Deployment of new versions of an application is less disruptive to users, and fixes can be deployed in a timely fashion that generally requires no scheduled system down-time.

The figure below depicts No Touch Deployment. Note that while MPR uses Microsoft's Internet Information Server, applications deployed using the No Touch Deployment model can, in theory at least, reside on any vendor's Web server.



For logistical purposes, the MPR Blaise .NET application suite is deployed from a single Microsoft Internet Information Server that also houses the Microsoft SQL Server 2000 databases, but the applications will scale to different physical configurations as user demands grow. At the same time, this configuration has allowed some initial proof of concept development to proceed. Most recently, we were asked whether it was possible for the Blaise DEP to communicate directly with an XML Web Service. Since a usable XML Web Service was already being developed for another MPR project, within a matter of hours a sample Blaise Survey Instrument was implemented. We quickly showed that the Blaise 4.7 API could be used to invoke an ActiveX COM object that in turn used automatically generated .NET proxy classes to retrieve information from the Web service returning it to Blaise.

5. Performance Issues

While the transition towards Blaise and .NET integration hasn't always been a smooth one, it has been a promising one that has MPR rethinking its Blaise and MPR Sample Management System integration approach. Of those problems that we encountered, performance issues were most prevalent.

COM Interoperability Performance. Although .NET COM Interoperability allows organizations to leverage their existing COM applications while making the transition to the .NET Framework and Common Language Runtime, it shouldn't be considered a long term solution. The intent is to provide a migration path until those COM applications can be rewritten to the .NET specification. There is a significant amount of overhead involved in context switching between the unmanaged COM environment and the managed .NET environment. Within the MPR Blaise .NET application suite, this overhead is most noticeable when reading large numbers of Blaise records for export using MPR Blaise Export. While not so obvious when only reading a single record at a time, the cumulative overhead associated with reading thousands of records is very obvious to end users. For the most part they are willing to cope with this inefficiency when balanced against the fact they can now get the data they want when they want them rather than wait for programmer turnaround.

No Touch Deployment Performance. Using No Touch Deployment, applications are launched using Internet Explorer. Because there is no interaction between Internet Explorer and the .NET download cache, applications having large footprints can be slow to load because Internet Explorer doesn't know that a usable version of the application is available in the .NET download cache and will only use a recent version that is available in the Internet Explorer cache. In many cases, Internet Explorer unnecessarily downloads a new version of the application. MPR has minimized the effect of this problem by keeping the startup application executables small, as we previously described with MPR.Blaise.Explorer and MPR.Blaise.Export, and relegating the substantive code to referenced assemblies. The result is that Internet Explorer only downloads a small 20K or less executable that is passed off to the .NET Framework, and subsequent loading of the referenced assemblies is handled more efficiently by the .NET Framework, which can recognize their availability in the download cache.

System Requirements. Traditional standalone Windows application are resource intensive by their very nature. Adding the .NET Framework can make them even more resource intensive because of the need for just in time compilation and the more robust security mechanisms that .NET offers. In many cases the workstations that are being used are simply not up to the task. Although 6-year-old, 650 MHz machines with 128MB of memory were once state of the art, they quickly become

obsolete with rapidly changing technology. Nevertheless they can still be used with the .NET Framework. Memory upgrades do tend to yield better performance. Further extension of their useful lives could also be achieved by moving to SmartClient application architectures that shift resource utilization to a server that is more easily scaled to constantly increasing user demands.

Performance Summary. Despite these performance issues, there are a number of techniques that programmers can use to make the application feel more responsive. While users are willing to wait for a response from a lengthy operation, they quickly become concerned when no feedback is provided during that operation and start questioning whether they actually clicked a button or not. The single most important technique that programmers can employ to make the user experience more comfortable is to provide users with indication that the program is actually working on their request. Sometimes even a simple wait cursor suffices. With the .NET Framework this becomes an even easier task with built-in, easy to use support for worker threads, progress bars, and status bars that can be used to provide more feedback during lengthy operations. At the same time, by using the consistently responsive user interface, end users quickly learn that they can switch to another unrelated task or application and return later rather than waiting or watching for the completion of that lengthy operation.

6. Conclusions

MPR recently implemented its first survey using the new Blaise Explorer and Blaise Export. While Blaise Export is backward compatible with older versions of Blaise, the Blaise Explorer is using the latest features of Blaise 4.7.1.b1000. All desired features are not yet in place for Blaise Explorer but the current feedback from end users at our Survey Operations Center has been positive.

Up to this point two very important items have made the project successful: first, reaching out for end user input and, second, implementing the normal look and feel of a traditional Windows application. From the outset of this project we sought user input into the design. We had a series of meetings with the end users asking questions about what they liked about Supervisory Review and what could be improved. As we started to build the tools, we continually demonstrated their progress to end users and solicited user feedback. This iterative process of review and feedback has given the users more of a feeling of ownership.

The look and feel of a traditional Windows application is already paying off. As we were going through the development process, the end users were given several opportunities to beta-test the tools. With little training most users were able to adapt quickly and tried things that are common in Windows applications. An excellent example of this is the use of Windows-based data grids in Blaise Explorer. Experienced users of Windows applications knew that clicking on the title field would sort the data in that field.

As part of the development process, we have gained important experience with the Blaise Component Pack and the actions within the DEP available to invoke the interfaces we have programmed. The BCP has provided us with many of the features we have needed, but the performance issues described in the previous section presented significant challenges. The uses of the .NET platform for the front end introduced the COM interoperability performance issues noted above. In our initial tests, for instance, the “sequential read” of a Blaise dataset to pull cases based on our common selection criteria took over 30 minutes on a 12,000 record data set. Blaise indexing did not provide the facilities we needed, and an attempt to speed up the application using C#—instead of our standard implementation

language of VB .NET—did little to improve the speed. These measures did not provide the response times needed for production applications.

Our solution to the performance issues was to use the “on exit” action in the Blaise DEP and equivalent exits in several Manipula programs to maintain an indexed SQL server table of searchable data. Searches on this table for individual cases or classes of cases are very fast, and we can then access individual cases in the Blaise dataset via the primary key. Retrieval of single cases via the primary key provides near instantaneous response times. (Note: our use of the “on exit” action uncovered an issue with the Blaise implementation of this facility; in response, the Statistics Netherlands team obligingly provided us with an interim build.)

In conclusion, we believe we have found a technical approach and an approach to the user interface that has made Blaise Explorer, and the accompanying Blaise Export, a successful application at MPR. In the process we have gained important knowledge about the BCP and Blaise DEP actions that should be basis for the implementation of real-time interaction between Blaise and our other .NET applications in the future.