Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

A VBA Desktop Database for Proposal Processing at National Optical Astronomy Observatories

Christa L. Brown

National Optical Astronomy Observatories, Tucson, AZ 85719, Email: cbrown@noao.edu

Abstract. National Optical Astronomy Observatories (NOAO) has developed a relational Microsoft Windows desktop database using Microsoft Access and the Microsoft Office programming language, Visual Basic for Applications (VBA). The database is used to track data relating to observing proposals from original receipt through the review process, scheduling, observing, and final statistical reporting. The database has automated proposal processing and distribution of information. It allows NOAO to collect and archive data so as to query and analyze information about our science programs in new ways.

1. Design Goals

ALPS is a custom database and application designed using Microsoft Access as a development tool. ALPS is designed to meet the Observatories' needs in handling proposals for telescope time in an environment accessible to both the administrative and the technical staff. NOAO chose Access as the development platform for a number of reasons. On the surface, Access offers a user-friendly approach for database neophytes. Its user-friendly features include:

- a well organized Database window,
- wizards for constructing database objects,
- a variety of built-in properties to define each object,
- and a simplified macro scripting language.

However, beneath the surface Access provides a powerful environment for the developer.

- Automation facilitates the exchange of information between Microsoft Office suite applications, providing both the developer and the end user with a wide range of tools best suited for various needs.
- The Visual Basic for Applications (VBA) programming language allows the developer to create a robust application and to automate complex tasks.
- The Access relational data model and Structured Query Language (SQL) foundation allow the developer to make uncomplicated representations of complex data.
- The Jet Database Engine exposes programmable Data Access Objects that allow program code direct access to database data and structures.

2. Application Architecture

The term "application architecture" describes the conceptual and physical arrangement of objects and processes. Within application architecture are these object-related subsets: data architecture, code architecture, and interface architecture. Data and interface architecture are discussed below.

2.1. Data Architecture

Most Access solutions are constructed in two pieces: the front-end application database file with form, report, and program objects, and the back-end data file with tables that are linked to the front-end. The front-end portion of the ALPS application resides on the user's workstation. The back-end is stored apart from the front-end in a Jet Database Engine on a file server. This model creates a file server database application, detaching the shared data (located on a shared drive for multiuser support) from the application objects (placed on a local hard drive for better performance). The separation of data from the interface allows for multiuser data access.

Data enter ALPS via two mechanisms, data entry and imports. The standard mechanism for entering records into tables is for users to key data into forms. Data are also pulled into ALPS by code routines that fetch data from external files submitted electronically, such as observing proposals, Telescope Allocation Committee (TAC) grades, and observing logs. ALPS buffers external data's entry into the application by loading them to a temporary table where they are validated before they are posted to transaction tables.

2.2. Interface Architecture

Almost all of a user's interaction with ALPS is through forms. Because forms are the primary device for allowing user interaction with data, they are the most visible and heavily used application element. Forms are the gateway to data records, provide the mechanism for displaying switchboard menus and selection dialogs, and are used to provide programmable class objects. At least 50 percent of the ALPS application's development budget and time has been spent on forms.

ALPS employs a Single Document Interface (SDI) paradigm. ALPS presents the user with one form at a time and a clear path forward and backward between related records and forms. We have not found a compelling need for users to compare different sets of data at the same time on a single screen. In addition, since many of our users are still limited by 15-inch monitors, screen real-estate is limited.

ALPS does not provide the everyday user with access to the Database window. Instead, users navigate from form to form, occupying a very restrictive and well-defined application universe. Our preferred user-interface model directs users through forms one by one, or at least limits them to some small, manageable group of related forms at the same time. The penalties for not following this model include the following:

- Users could enter dependent data out of sequence. For example, they might try to enter child records before parent records.

- Users could open multiple forms that point to the same record, introducing possible record-locking challenges.
- Users could open too many forms at the same time, impacting application performance or losing track of their workflow.
- Users could keep multiple unsaved records open, usually by minimizing forms on the desktop without saving their edits. This situation would prevent other users from seeing current data.
- Users could lock records, usually by minimizing forms that have edited records in an unsaved state (depending on the form's locking model). This scenario would potentially lock other users out of specific records for an unreasonable amount of time.

By enforcing only permitted behavior and events, users are guided through the application in a friendly and logical manner. The ALPS user-interface model includes the following components:

- A main switchboard menu form enables users to navigate to the various features of the application.
- Toolbars and menu options aid navigation, but do not include any built-in Access options that are dangerous to the specific application.
- Forms that open in a single-instance modal state create a simplified environment in which the user performs one task at a time.
- A controlled data mode provides a filtered recordset with the ability to add, edit, browse, and delete selectively within the recordset, as appropriate to the context.
- Keystroke trapping provides form and record navigation. ALPS doesn't assume that users always navigate with a mouse.
- A limited feature set, with features disabled when they are not needed in a specific context, gives users access only to options that they can use effectively to execute the task at hand.
- A clear path to navigate enables users to move forward and backward between application elements. The application establishes and enforces order in the workflow.

3. Switchboard Menus

At the top of the ALPS navigation pyramid is a switchboard menu. The switchboard menu is a single page form with a button for each main functional area in the application. Each menu button contains attached code that opens a form, prints a report, runs a data process, or opens another switchboard.

The ALPS switchboard interface model uses a column of small buttons, each with a text descriptor next to it. The strength of this design style is the capability of conveying varying lengths of explanatory text to the user for each application option without creating wide command buttons.

Compared to table-based menu forms, however, this option is very inflexible. To add a feature to the menu, a developer must add a button and its related code to the form. On the positive side, this switchboard loads and executes faster than a form tied to a menu options table.

Another positive aspect of button-based menus is the capability to enable and disable features selectively on the fly by toggling the Enabled property of each button. For example, the Import Proposals button on the Proposal Submission switchboard is enabled only if a proposal has been submitted and resides in the directory reserved for these files. Thus, a user can see if there are any proposals to import by simply looking at the button status on the Proposal Submission switchboard. This very simple programming technique conveys immediate status information to the users via the menu form. In addition, the technique produces highly visible results for the small amount of development time it takes to create a smart menu. Achieving this result is much more difficult in a menu based on a list box, in which you must change a flag value in the underlying table and requery the menu. Consequently, you must also keep the menu list in the local database so that table changes do not affect other users, which may adversely affect your application maintenance strategy.

Another weakness of the button-based menu is that only a limited number of options can be displayed on this menu. Using a list box instead of a button-based metaphor would enable the list of menu options to be essentially infinite. In addition, a list-box menu could be driven by a table containing its options. Modifying the application's capabilities would become simply a matter of adding or editing the table's records. A list-box menu could also allow a layer of programmatic security to be implemented by only listing options in the list box that are permitted for the current user.

Options in the menu list are sorted based on a process-flow model. The order of menu options follows the orders that tasks are executed. For example, the proposal submission and Telescope Allocation Committee (TAC) preparation processes have a logical flow; steps must be performed in sequence.

4. Conclusion

The ALPS database and application have been in operation at NOAO for two years, during which time they have grown in complexity and functionality. ALPS is presently being modified to allow support for multiple observing sites and for replication among sites to meet NOAO's changing needs for support of observing proposals.