Converting Legacy Superbase

Where to Begin?

Explaining how to convert from a product that is as multi-faceted as the legacy Superbase product is not an easy task. The number of different ways that people have used the product means that any detailed set of instructions will be certain to fail the needs of a large percentage of those interested. Instead, this chapter will discuss some guidelines and techniques for conversion.
It is probably easiest to start with what things can be converted fairly easily. That list is as follows:

- Database files (assuming they are not encrypted)
- Display forms (currently DisplayTextBox objects are not supported – rotated editable text boxes)
- Print forms (these are the same forms in legacy Superbase, but are primarily meant to be printed and are handled separately in Superbase NG)
- Dialog definitions (the DialogFrame object is not currently supported)
- Menu programs (as saved from the Superbase Menu Editor)
- Graphic Reports (some hand-tweaking may be required in the resulting XML)

What is noticeably lacking from all of the above is any mention of program code. Legacy Superbase supports three distinct styles of BASIC programming:

- Early QuickBasic with only global variables, and GOTO, GOSUB, and RETURN, with both line numbers and symbolic label names.
- Procedural BASIC with SUB main() local and global variables, user-defined functions, and an event handling mechanism for creating event-driven programs.
- Object BASIC with supplied objects for the GUI components, like forms, and form controls, dialogs and dialog controls.

That list mirrors a clear progression in the development of programming languages over the course of time. The problem is, legacy Superbase allowed the use of these different styles of programming concurrently. That isn't so bad for the final two, since the object BASIC is layered over the top of the procedural BASIC anyway. The problem is the original BASIC, and the excessive use of the GOTO and global variables.

There is nothing inherently wrong with the GOTO command (although some might argue very strongly about that), if it is used carefully (and sparingly) in the hands of a skilled programmer, but unfortunately it changes the direction of program execution permanently, and often cannot easily be followed by someone (or a program) reading the source code. Often the original author of the code will no longer understand how it works within even a few months of having written it. Needless to say, if the original author no longer understands how their program works, the likelihood of any program understanding it, even one that is designed to convert source code, is very low. Having said that, it is not as bad as it sounds. Please read on.

How Superbase NG Differs

When SIMPOL was designed, one of the strongest factors in the design of the language was to make the code easy to learn, easy to use, and easy to maintain later. An unfortunately common scenario that has played out far too often in many places using tools like legacy Superbase, is where an island solution built by an inspired layman programmer achieves a degree of success. Then as its star rises, it requires additional professional assistance to make it to the next level of usability. At that point, the professionals investigate the software and discover that it is written in a way that is non-standard, complicated to understand, and possibly built using a tool that they personally have no experience using. At which point they decide to discard the original and start over again. The problem is the original solution probably took one dedicated person 1-3 years of work to build using a very powerful tool. The new version usually is estimated to require 3-5 people several years to produce, and would cost a fortune to achieve it. At which point the whole project might be scrapped as too expensive.

To prevent the solutions that were built by non-expert programmers from being discarded as unmaintainable or unsupportable once they reached this level, every effort was made to avoid this result. All the factors that were bars to entry for quickly learning the language were discarded. As much as possible, redundancy was removed from the design. The keyword set was reduced to the bare minimum and everything was turned into a type or a function. There is also no way to jump out of any block statement, so it is always clear how the code flows, and there are no global variables.
Not having global variables is one place where SIMPOL strongly differs from many languages. The choice to not allow them went back to the problems that are commonly associated with them:

- Random unexplained changes in one module as a result of calling code in some other module
- Assignment to apparent local variables changing the value of global variables
- The constant search for a new and valid name for a global variable
- The inability to distinguish in the code between a local variable and a global one

So what was the gain for SIMPOL by not having them, and how does it cope with certain situations that appear to require them? By not having global variables, all variable changes are specific to the function in which they are created. If a value is need in a function from outside the function, it must be passed into the function as a parameter. It is always clear where the values are coming from.

But what about event handlers? How do we get the data we need into an event handler if we don't call it directly? That is handled by every event having an additional property called reference. This property is declared to be of `type (*)`, which is a special placeholder that allows a variable to hold a reference to any data type. This is the mechanism used to pass quasi-global data to an event handling function. With that, the loop is closed and there is no other need for global variables.
Every SIMPOL program starts in the function `main()` and ends when that function is exited (unless the program is multi-threaded and one or more threads are still executing at that time).
That all sounds like loads of work, if there is a lot of legacy Superbase code to change. The reality is a little different. As it turns out, much of what people code is about working around how their environment works. Legacy Superbase is no different. The easy route is to move the data and the forms, migrate the menus, and then see what works and what is missing. Then add the code as event handlers for the various event types.

One important difference to note is that Superbase database files allow the definition of calculations, constants, and validations as part of the field definition. The initial Superbase NG database engine is a pure storage engine, and it does not cater for these field-level operations. This may seem to be a significant drawback, but in fact, most of the more advanced legacy Superbase developers had stopped using these in their projects quite some time ago, since in any complex project these sorts of things could get in the way and cause as much trouble as they provided help.

To resolve this in SIMPOL, it is necessary to migrate those settings into a function or set of functions. In an earlier chapter, Chapter 6, GUI-Style Database Programs, a special function was built to create the serial number when a new record is created in a table on a form. A similar function would be needed for each time a record is created in any table that needs constants to be generated at that time. A similar function would be needed for calculations, which could be called every time a record is saved (it could also be called during the onlostfocus event of certain controls).

So What's the Good News?

If the legacy Superbase content is primarily data and forms, with a few reports, the conversion should be pretty quick and painless. If there is a large amount of code, then the process can use the phased migration approach.

What phased migration means, is that there is a methodology where the data can be converted to use the new Superbase NG database format, the legacy Superbase application can be converted to use the PPCS method for accessing the data, and then over time, modules of the Superbase program can be converted into Superbase NG and called from the legacy Superbase program. Depending on the design of the legacy Superbase application, some items might be ready to convert sooner than others. Also, since both Superbase and Superbase NG can access data via the PPCS protocol, web server applications written in SIMPOL can be used to provide browser-based access to aspects of the converted Superbase data in real time. This capability to keep the original application in legacy Superbase and to gradually migrate it over the course of time is not available with other tools. It has the advantage that the existing software can be maintained and updated

(wherever possible building new modules only in SIMPOL) and gradually more and more of it will actually be in SIMPOL. The key to this is that both can share the same database concurrently. Changing an existing legacy Superbase program to use PPCS instead of the normal method of accessing data rarely takes more than a single day, no matter how complicated the program is. PPCS was designed to be an easy move for legacy Superbase programmers. It does require the user to be on a fairly recent version of the legacy Superbase product, though. No less than version 3.6i, preferably as of build 496. Many of the supplied conversion tools need to run on the Superbase 2001 version or later.

Converting Superbase Databases to Superbase NG

There is a very useful tool supplied in the Utilities directory called sbf2sbm.smp, which converts legacy Superbase database files into Superbase NG's `*.sbm` format. This reads the data file directly, so it does not require any extra action to make it available, with one exception. It cannot read encrypted Superbase database files. In that case the file needs to be converted to a non-encrypted database file. Just as a note, currently there is no encrypted file format for Superbase NG database files. At the same time, since multi-user access is only via PPCS, the location of the physical data does not need to be accessible to every user as is the case with the older Superbase LAN and Distributed LAN networking.
Running the SBF2sbm tool presents a dialog window like the one below:

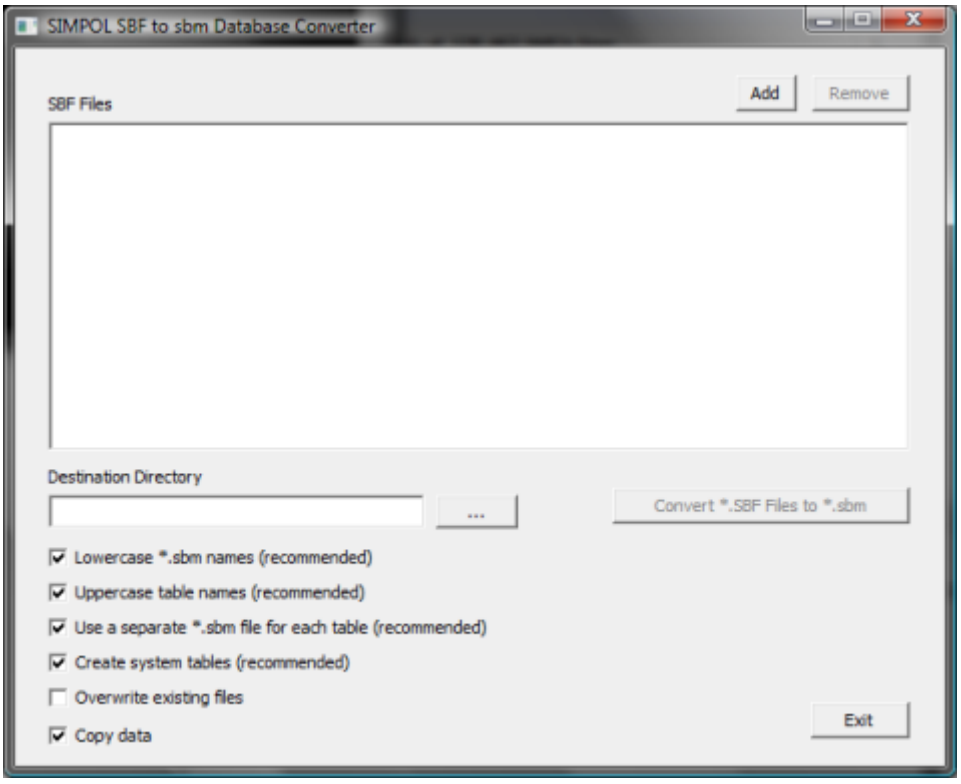


Image of the SBF2sbm dialog.

As an experiment, we are going to import the database tables from the Superbase Air example that ships with all versions of Superbase 3.x (Superbase 95, Superbase 3.01, Superbase 3.02, Superbase 3.2, Superbase 3.5, Superbase 3.6i, Superbase 2001, and Superbase Classic).
To start with, we click on the Add button, which let's us select the `*.sbm` files and add them to the lis select all the files at once.
Th t will be set to the same as th

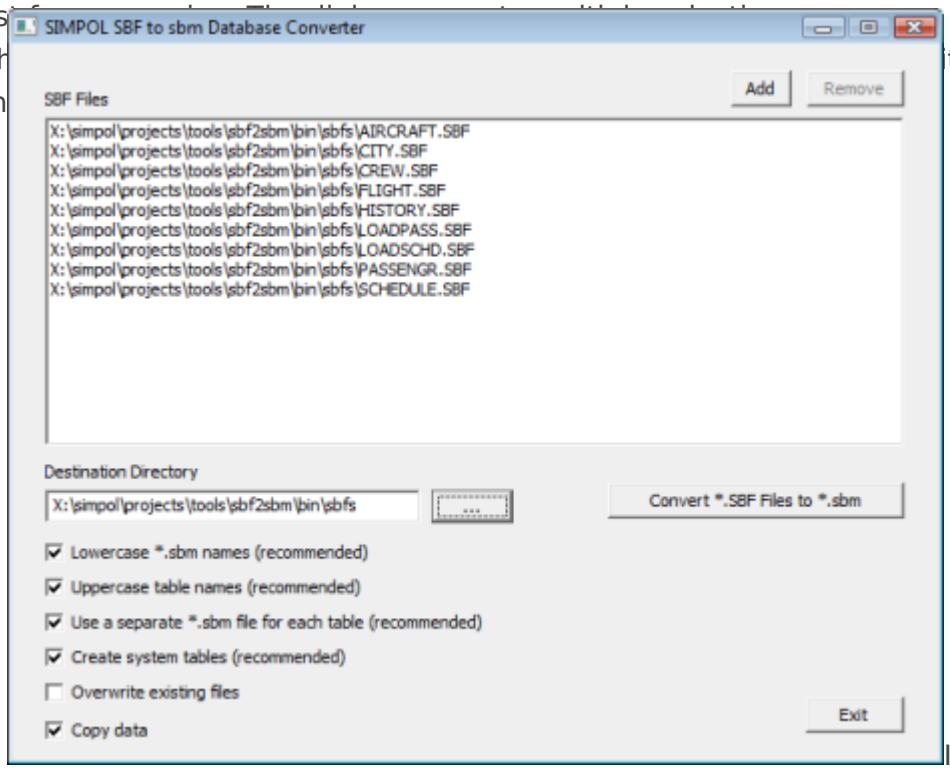


Image of the SBF2sbm dialog

ready to convert.

Leave the settings at their default values for the most successful conversion. Below each of the settings is explained.

- Lowercase `*.sbm` names (recommended) — This makes sure that if the tables are being converted into separate container files, one per Superbase file, that the file names will be forced to lowercase, otherwise they will be in uppercase (like the original files from Superbase).
- Uppercase table names (recommended) — This ensures that the tables are created with uppercase names. This is important if working on a hybrid solution, since SIMPOL is case-sensitive when opening the tables.
- Use a separate `*.sbm` file for each table (recommended) — Although SIMPOL database containers can support multiple tables, there are good, performance related reasons for keeping each table in a separate container. It also makes it easier when doing updates of specific tables, or if reorganizing only one table.
- Create system tables (recommended) — Unlike Superbase, SIMPOL database fields only have a data type, and whether they are indexed or not (plus if they are, an index algorithm and precision). Things like the display format are not part of the core field definition, but the system tables store additional information such as the display format,

help string, share name (which can be different to the field name) and other useful bits. Using the system tables means that the standard PPCS server program can automatically share the table and have it look just like the original from Superbase (minus calculations, etc.).
- Overwrite existing files — If selected, it overwrites an existing file without asking. If it is not selected, it will not ask, and will not overwrite.
- Copy data — This determines if the table is created empty, or if all the data is also transferred.

> Note
> One thing that it is important to understand, is that this tool cannot resolve the calculations for a virtual field. If the table definition has virtual calculated fields, and if those fields are unique indexes, then the import of that table will fail. This was the case with the `SCHEDULE.SBF`, since it turned out to have multiple virtual calculated fields, one of which had a unique index on it. In order to successfully import the table, the fields need to be changed to normal fields (not virtual), and the content needs to be updated by doing a Superbase UPDATE that specifically sets each of these fields to be equal to itself.

Converting the Forms

Now that the data is in Superbase NG format, we can convert the forms. The form conversion tool is a SIMPOL program. This program reads the `*.sbv` files directly. It is also the only way to recover embedded images that are in the form itself. To run this tool, from the Start menu, select Superbase NG → Superbase Classic Conversion Tools. This will launch the tool that hosts the various Superbase conversion tools. Select the one for converting the forms. It will look very similar to the one we used for converting databases. Select the form(s) you wish to convert and provide a target directory. Click on the Convert Forms button and the forms will be converted into the target directory.

In this example, we will be converting the CHECKIN.SBV. The original is shown below.

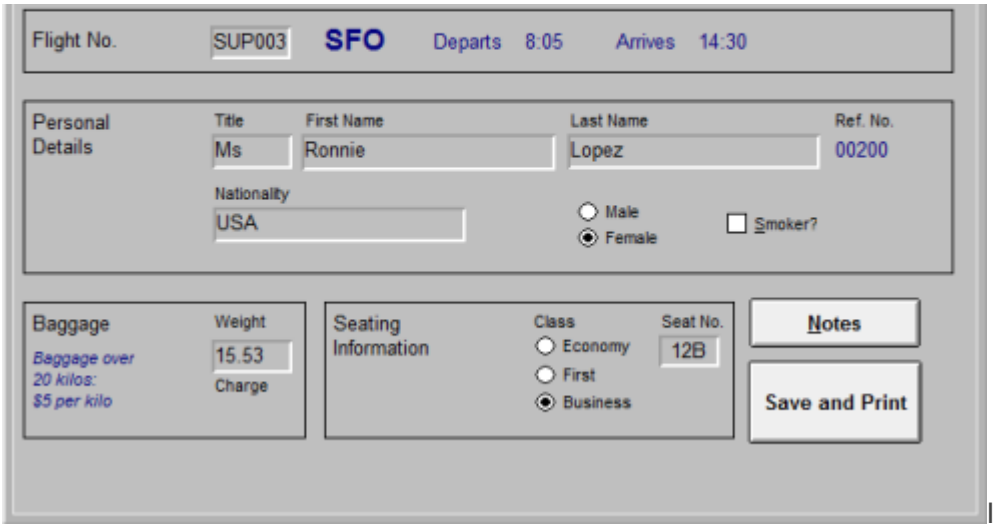
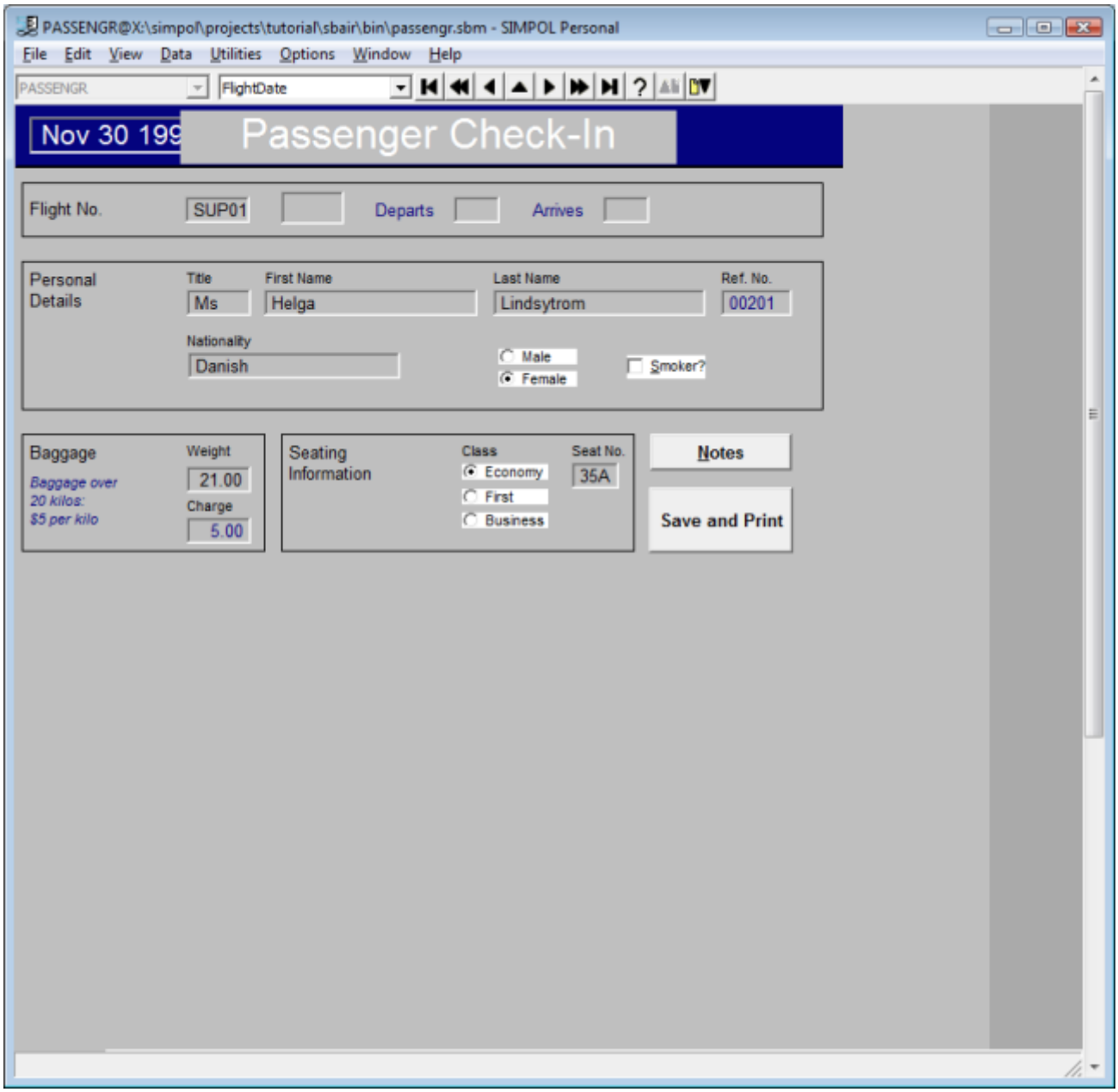


Image of the CHECKIN form

prior to conversion.

The following image is of the converted form, without any modifications done to it, merely opened as is in Superbase NG Personal.



Converting the Forms

As we can see from the converted form, it looks different in a number of ways. That is in part because the controls on a Superbase form are not native controls. Instead they are all drawn by Superbase. Also, the original form was not sized correctly, it is actually much longer than it needs to be. Also, in Superbase the background of a label could be optionally turned off. This is not an option with real Windows controls. The rest are relatively minor adjustments. One thing that the conversion program does not yet do, is transfer the tab order. That is because tab order works quite differently between the two. Eventually this may also be added to the converter. That was hand adjusted as part of the work in the Form Designer.

Another thing that is different is the Superbase text boxes in the original form that have no border, and are not recessed. This option is not available in standard Windows controls. SIMPOL has its own trick here though: dataform1text controls are data-aware (can be bound to a field), labels in Superbase are not. Also note that the buttons are using a different color scheme to the rest. That is because they were defined as using the standard color scheme. In Superbase the colors are fixed. In SIMPOL, they can be tied to the current theme.

Another problem is the missing image. The reason it is missing is that the image was pasted into the form from the clipboard, and there is no way to extract the image from the form p
B
r
)L Form Designer, the

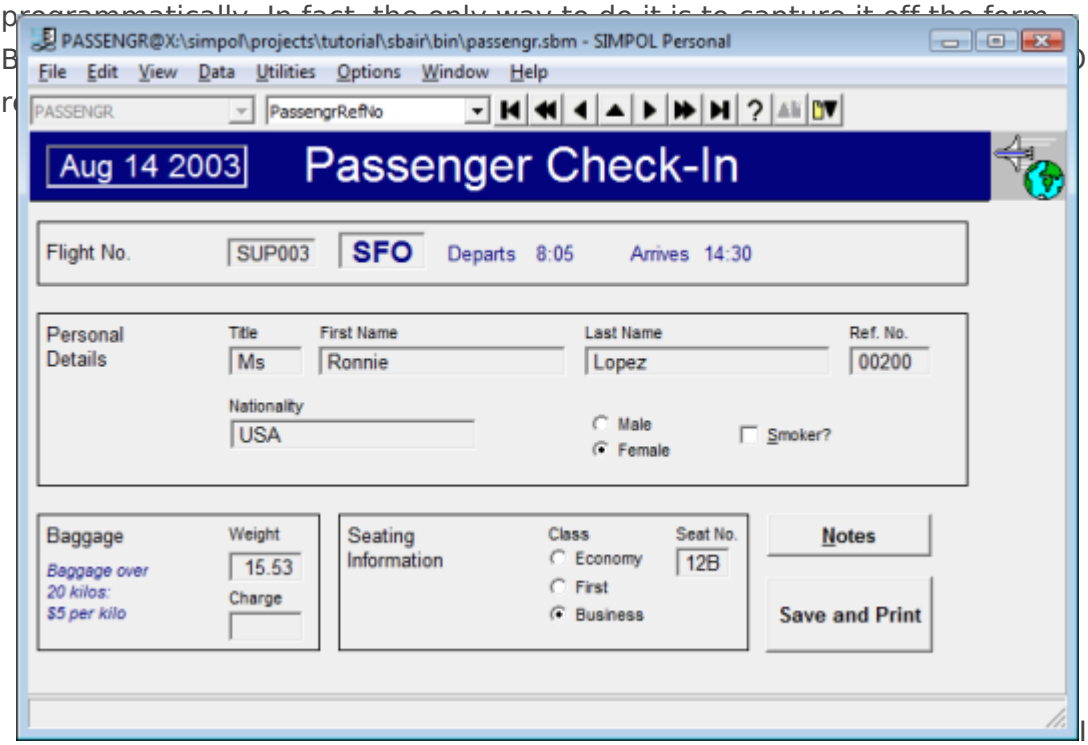


Image of the CHECKIN

form after repair in the Form Designer.

As we can see, it now looks very much like the original. In some ways, it looks better. It now uses the system theme for the vast majority of colors (not all, however, since some text was blue, and it is not supported to have the text color using system colors and the background color using fixed colors). The information in the Flight No. section of the form that is in blue is actually looked up from the FLIGHT table using the FlightNo as the link.

Creating the Application

The steps to turn this into an application of its own are quite simple really. As described in the section called “Summary”, all it really takes is to create a project, which I called `sbair`. I then copied all the program files from the addressbook program into the new project directory, renaming the one called `addressbook.sma` to `sbair.sma`. I also copied all the toolbar images from the `addressbook`\bin directory into the `sbair`\bin directory. Into the same directory I copied the converted database files (at this point we only need `passengr.sbm` and `flight.sbm`), and the converted and reworked form: `checkin.sxf`.
In terms of changes to the program code, all the suggestions made in the section called “Summary” were applied. The resulting program came up immediately and can be seen below. The copying and code changes took less than ten minutes, including compiling and fixing things that were forgotten.

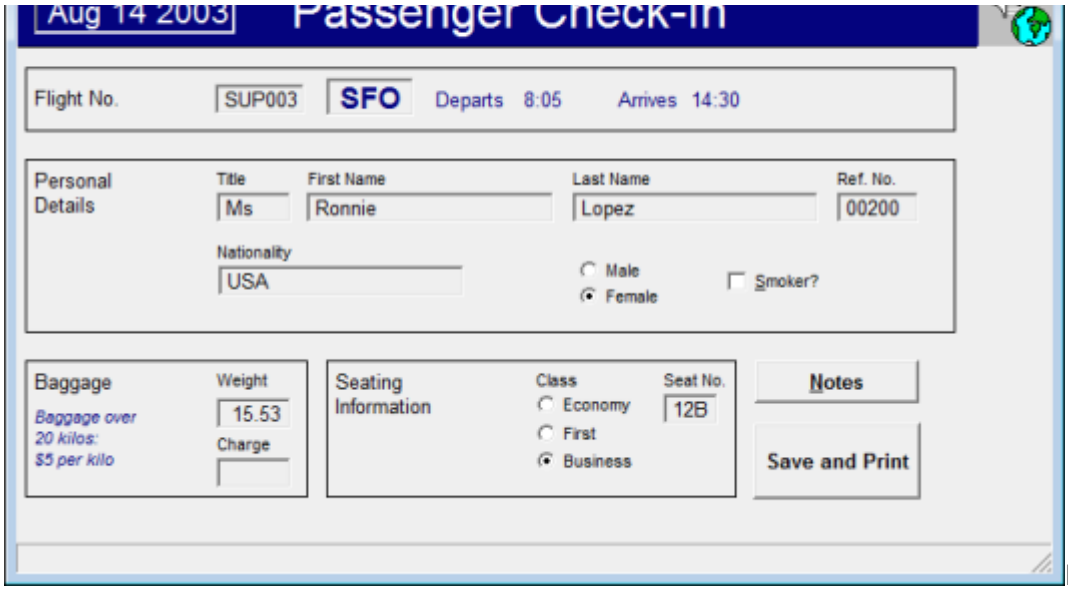


Image of the CHECKIN form in the new SB Air application.

This is, of course, just the beginning. A full conversion would convert each of the forms, add navigation to the menu, add functions to support the buttons on the forms, add functions to handle constants and calculations for the tables, etc. The goal here was only to demonstrate the approach, not do a full conversion.

Summary

In this chapter we discussed the issues facing a conversion from Superbase to SIMPOL. We also looked at various scenarios and discussed why SIMPOL offers the easiest conversion solution for existing Superbase projects, especially the advantage of doing a phased conversion where over the conversion period there is a hybrid Superbase/SIMPOL application that starts out wholey in Superbase and eventually bit by bit becomes completely SIMPOL.
Then using the tools, we converted a portion of a standard sample shipped as part of the Superbase 3.x series, including converting the database tables and one form. That form we then cleaned up in the SIMPOL Form Designer and finally we built a standalone application for it in just a few minutes by stealing most of the code from a standard sample program.

Revision #6
Created 15 November 2021 19:05:24 by Eva Crawford-McKee
Updated 27 December 2021 21:36:32 by Nikolaus Zolnhofer