# Overview

During this exercise you will add the code to process the employee details input window to your application. At this point, much of the foundation work that you have done will start to come together. You will use the input window you have designed, along with the special input processing menu columns, and the record definition you created.

# Resources

* [Synergy DBL Language Reference](https://www.synergex.com/docs/index.htm)
* [Synergy Best Practices - Coding Standards](http://jobfunc2.cu.net/Job%20Functions/Programmer/Programmer%20Handbook/Tims%20Best%20Practices%20-%20Standards/Synergy%20Best%20Practices%20-%20Coding%20Standards.docx)
* [Traditional Synergy in Visual Studio - CU Wiki](http://echo.cu.net/cuwiki/Traditional_Synergy_in_Visual_Studio)
* [Traditional Synergy in Visual Studio Common Terminology - CU Wiki](http://echo.cu.net/cuwiki/Traditional_Synergy_in_Visual_Studio_Common_Terminology)
* [Debugging (TSVS) - CU Wiki](http://echo.cu.net/cuwiki/Debugging_(TSVS))

# Exercise

1. Using Visual Studio, open the previously created “**Demo**” project.
2. Using Visual Studio, open “**Demo.dbv**”.
3. Modify the call to DTK\_START to remove the footer.
4. Also remove the E\_SECT call that displays text into the footer.
5. Load, but do not place, your input window in the initialization section of the program.
6. Create a new internal subroutine called pProcessEmployee.
7. Code the subroutine to process the "STTEmpdtl" input window, as follows:
   * Place the input window.
   * Create an input processing event loop, which does the following:
     + Calls the input processor, naming the record area where the data should be returned, along with the ID's of the two standard menu columns.
     + When the input processor returns, decode the event, which caused the input processor to terminate. This can be one of three things.
     + When you detect menu entries, add code to detect both the "CU\_EXIT", and "CU\_PANIC" menu entries and have the code exit from the input processing event loop in both cases.
     + Where you detect break fields, display a message indicating which break field is being processed.
     + When you detect that input processing is complete, display a message containing the data, which has been entered (in the "employee" record), before exiting from the input processing event loop.
   * After exiting the event loop, remove the input window, before returning from the subroutine.
8. Edit the code in your existing menu processing event loop and when the "MAINTAIN" menu entry is detected, call the new "pProcessEmployee" subroutine instead of displaying a message.
9. Compile, link, and run the application.

# Discussion

The program should now be processing the input window and returning the entered data to you in the “employee” record. At this point, there are several outstanding issues with the way in which the window is being processed (for example it is possible to enter employee details without first entering an employee ID) but we will address these in the next exercise.

The basic structure of the event loop that you have created is applicable to ANY input window that you process. Indeed it is possible, with a little thought and careful planning, to simplify input processing even further by encapsulating the event loop in a re-usable subroutine or include file.