

Maximizing the Flexibility of Your LPS System by Providing Data for the General Case

The Loftware Print Server (LPS), along with the entire line of Loftware products, has an open architecture that can be set up in several different ways. By following some simple rules, you can obtain some impressive flexibility. This white paper contains concepts and procedures that you can follow to achieve the goals outlined below. Please note that this discussion is for the LPS system only and assumes a base knowledge of Loftware components.

- Existing Labels can be modified without changing code. This is accomplished by writing your code to extract the variable field names from the label file before each print. If fields get removed or added, your code will take care of it "on-the-fly".
- New Labels can be added without changing code. By constraining the list of available field names in design mode, designers can only use variable fields that your Front End program knows how to handle.
- New Printers can be added from different manufactures. When printer manufactures and models change, simply re-save your label with the new model number. The Front End application need not know anything about this.
- Variable images, special check digits, formulas, serial numbers and many special features present in today's modern thermal transfer printers are supported and can be added to label formats without changing code.
- Tables and Fields can be added to databases and used in new label formats without having to write code. By keeping the constrained field list up to date with database table and field names, users can change their database and their labels without the knowledge of the Front End program.

The Concept of "Data Push"

Although the LPS has the ability to access your data through 32-bit ODBC drivers, experience tells us that this is often the incorrect approach. Chances are that your application already has access to the data. Because of this, it is easier and faster for your application to "Push" the data to the Back End (the LPS). There usually is no need to further complicate your system by having to install ODBC drivers on the LPS server and configure the labels to grab data from across the LAN or WAN. Worse yet, if your data is on a host system, why complicate matters by having to go through bridges and connectivity software. The concept of "Data Push" is meant to simplify your system in setup, performance and maintainability. There are cases where the LPS must "Pull" the data from your database. Perhaps your application is running on an RF hand held device that does not have access to the data. Only "Pull" data when you have no other choice.

Note: If you intend on pushing the data to the LPS (as shown in the following example) make sure that the data source's for the variable fields on your label are set to "keyboard" (default). Many users make

the mistake of trying to attach their variable fields to a database even when they intend on pushing the data. Don't fall into this trap!

Constraining Label Design Mode

The LPS system makes provisions for your Host or PC application to know what variable data fields are present in your label formats. This allows your application to decide what data to pass based on which format is being processed. This avoids having to "hard code" your system for specific labels. Users can make changes and add new labels to the system without having to involve programming staff.

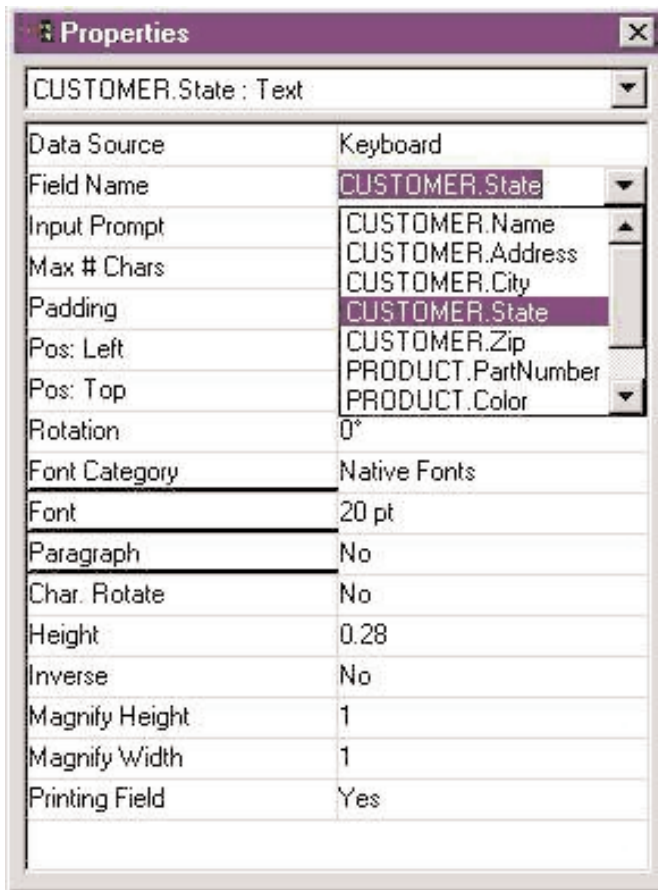
Example: ABC corporation has a relational database on their corporate AS/400 named BigDatabase. It has two tables with several fields as shown in the following diagram.

CUSTOMER	PRODUCT
Name	Part Number
Address	Color
City	Description
State	Weight
Zip	

Using Notepad or your favorite text editor, create the following file in the llmwin32 directory. This file reflects the possible field names in the database that may be used on a label. In this example, we have chosen to use the "TABLE.Field" convention for later use in an SQL statement. You may use any convention that suits your needs. The label designer will sense that a file with a .LST extension exists and will go into a mode where you have to pick a field name from the list.

LLMFIELD.LST

```
CUSTOMER. Name
CUSTOMER. Address
CUSTOMER. City
CUSTOMER. State
CUSTOMER. Zip
PRODUCT. Part Number
PRODUCT. Color
PRODUCT. Description
PRODUCT. Weight
```



```
CUSTOMER.Name,15,1,503
PRODUCT.Color,11,2,503
PRODUCT.PartNumber,11,3,507
```

LABEL1.TAB

Note: The .TAB file contains field name, field length, tab order, and a field type code. 503 is for variable text fields and 507 is for variable barcode fields. Chances are, the only data your application will be interested in will be the field name.

If your program runs on a host computer, it will need access to the .TAB file in order to know what fields are on the label. If your application runs on a PC, it could use the Loftware ActiveX control to obtain the same information from the .LWL file. Having this information available allows your program to retrieve and supply the minimum data necessary to print the label. If you choose not to use the .TAB file, you can still achieve field name independence in your application by supplying data for all potential fields on any label. The label being printed will grab the data it needs, and ignore the rest. If you have a large number of fields, sending them all can prove to be inefficient. The choice is yours!

Example (continued): The ABC corporation has written a Visual Basic Client program that determines through various criteria what labels to print. This program utilizes the Loftware ActiveX control to obtain a list of the variable fields on the label. It then builds a SQL query on-the-fly in order to retrieve data from the AS/400. The following SQL statement could have been generated by iterating through the "field name array" that the ActiveX control creates after invoking the "SetLabelName" method. If you are not familiar with the ActiveX control, or you are programming in a different language on a non PC platform, you may achieve the same result by programmatically retrieving the field names from the .TAB file. Note that the SQL statement below is a popular way to retrieve data from a PC application. Although the concept applies to all platforms, the actual method that you use to retrieve data may be different.

```
"SELECT CUSTOMER.Name, PRODUCT.Color,
PRODUCT.PartNumber FROM BigDatabase
WHERE CUSTOMER.Name = 'Anderson, Dana';"
```

This is a very nice way to retrieve only the data needed by the label. It also allows your program to be field name independent. New labels can be designed without having to change the code that supplies data to the LPS! This powerful concept is what we mean when we say that you no longer have to "hard code" variable field information for specific labels. Please note that the key to making this process work is to use actual table and field names in your .LST file that apply to your database. If a field or a table gets added or deleted to your database, you must revise your .LST file to reflect the change.

Example (continued): Assume the SQL statement above retrieved "Blue" for PRODUCT.Color and 4300339 for

The diagram below shows how the properties box in label design mode behaves with the presence of this file. In this mode, a variable field name can not be typed in. It must be chosen from the list. Please note that this list is used to constrain the field name of a variable field on your label during design time. It does not actually connect the field to a database.

Getting Field Names From the Label At Print Time

When the label design is saved, a file is saved with the same name with a .TAB extension. For example, suppose you designed a label with the designer constrained as shown above. The name of the label is LABEL1.LWL and contains fields for CUSTOMER.Name, PRODUCT.PartNumber and PRODUCT.Color. The label and corresponding tab file would look like the following:



LABEL1.LWL

PRODUCT.PartNumber. A pass file for the WatchDog-NT interface can now be generated. The file will be named LABEL1.PAS and tells the LPS to:

1. Load the format named "LABEL1"
2. Set the field data in the label to the data that was retrieved from the database
3. Print one label on the Printer whose name has been set to "Shipping Printer 2"

LABEL1.PAS

```
*FORMAT,LABEL1
CUSTOMER.Name,Anderson,Dana
PRODUCT.Color,Blue
PRODUCT.PartNumber,4300339
*QUANTITY,1
*PRINTERNAME,Shipping Printer 2
*PRINTLABEL
```

Note: Refer to the WatchDog-NT chapter (or web discussion) for detailed information on how this file is used and the different file syntax's supported. The file shown above is called ".pas" syntax and is not necessarily the most efficient one.