An Introduction to SAP's New and Improved Frontend Printing

Stefan Fuchs

Dr. Stefan Fuchs has worked since 1987 developing new operating system concepts for the Chair for Operating Systems at the Technical University of Karlsruhe, Germany. Since 1995, Stefan has been working for SAP Germany, in the Basis Department, on the development of the R/3 Spool System.

(complete bio appears on page 48)

How many of you, as R/3 administrators, have had to define an output device[1] for each and every user who wants to print to his or her personal, frontend printer? And how many of you, after creating dozens, hundreds, or even thousands of output device definitions, have had to contend with end-user complaints because:

- Users do not have a way to interactively select an output device via a familiar and intuitive Windows dropdown box, as is the case with most applications from which they print?
- Users can only print to a fixed printer name (or, in newer R/3 releases, to their current default printer) because the name is a fixed part of the output device definition?
- Users who work on a different terminal each day and who require local output must locate the output device that is associated with their current terminal?
- Users on terminals with a dynamic IP address (e.g., laptops that connect by a phone line) have difficulty printing to their frontend?

These are common R/3 Release 3.0 frontend printing scenarios.

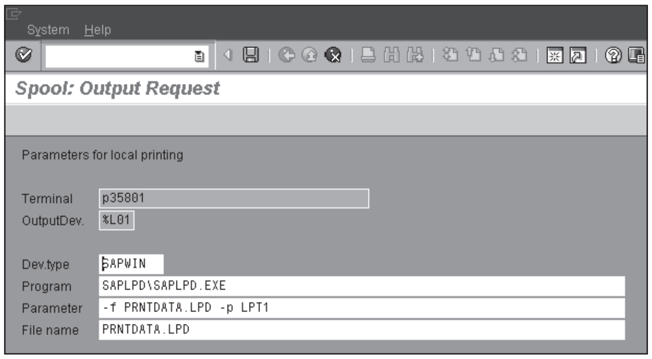
A clean and easy way for users to print directly from the SAPGUI has long been an elusive goal. The reason is that R/3 is embodied in a three-tiered client/server structure, where print data is never completely available at the frontend. Only the visible portion of the data is transferred to the GUI for display. So R/3 is very different from most Windows applications, for example, in the sense that R/3 is actually *not*

[1] An output device is a dedicated object in an R/3 configuration. It denotes the logical destination of an output request, while a printer is an actual, physical printing device.

running on the frontend.[2] Only the GUI is. That being the case, SAP could not simply add a print function to the GUI to print out all the data associated with a spool request. We had to develop a new spool access method that allows the transfer of print data from the application server to the current location of the SAPGUI, and from there have the data sent to the local spooler on the user's particular frontend. That new spool access method — method "F" — is what we at SAP call "frontend printing."

Frontend printing makes life a lot easier for both end users and administrators. The predecessor of frontend printing, the "local print" functionality in transaction SP01, can *only* be used from within transaction SP01.[3] This means that a user first has to choose an arbitrary output device and then create a spool request without the "immediate" print option. Afterward, the user has to call transaction SP01, locate his or her spool request, click the "Print" button, and then click the "Print locally" button. This gives rise to a pop-up, like the one shown in **Figure 1**, with four parameter lines that most users do not really understand.

Figure 1   The "Local Print" Pop-Up



Spool: Output Request

Parameters for local printing

Terminal	p35801
OutputDev.	%L01
Dev.type	SAPWIN
Program	SAPLPD\SAPLPD.EXE
Parameter	-f PRNTDATA.LPD -p LPT1
File name	PRNTDATA.LPD

How many of your end users, for example, understand that the **Device type** parameter should be the same type as the device type of the output device that was selected to generate the spool request? When they are not the same type, printing errors can happen or the output can take on an appearance that was not intended by the user. How many of your users know that the **Program** that is entered on the next line to execute the printout is normally the SAPLPD program, but theoretically could be any command line that does printout? Do most of your end users know that the **Parameters** are parameters to the program that was specified in the preceding line? For SAPLPD, the -f parameter gives the *filename* of the temporary print file and the -p parameter is followed by the name of the Windows *printer* that should receive the output. And for our bonus round, how many of your users would be able to explain that the **File name** in the last entry specifies the name of the temporary print data file that is downloaded to the frontend? This name must be the same as the one supplied with the -f parameter in the preceding line, otherwise SAPLPD would not find the print data. (The file automatically gets deleted by SAPLPD after processing the printout.)

With frontend printing, users don't have to be concerned with any of these details. In fact, they never see the pop-up shown in Figure 1. Instead, they are presented with the standard print pop-up shown in **Figure 2**. In order to do frontend printing, the user just has to enter the name of an R/3 frontend output device ("LOCL" in this example).

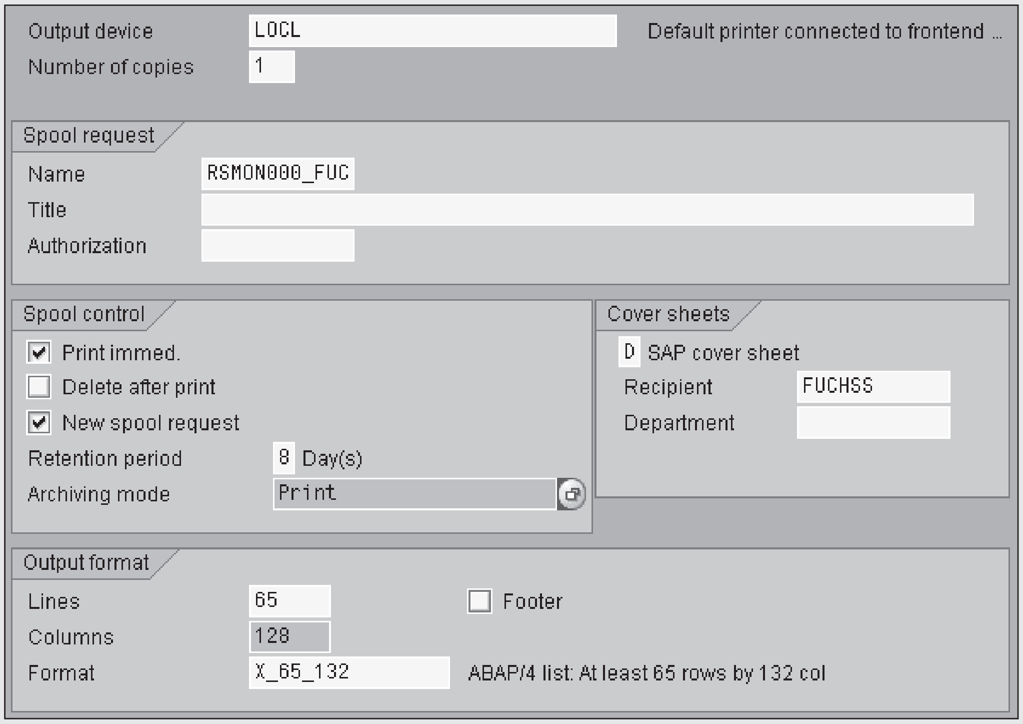
As of Release 4.6A, this print pop-up has been extended to allow the selection of the frontend printer, as shown in **Figure 3**. After entering a frontend output device name ("LOCL"), a dropdown box appears that lists all the printers that are available on the frontend. The user can interactively select a printer from this list, just as he or she would in any Windows application.

Since this solution revolves around a *single*, generic output device definition that can support *all* frontend users, administrators don't have to do a lot of work to facilitate this functionality. There is no need to establish a definition for every frontend printer in your R/3 environment. One generic definition will do the trick. R/3 takes care of the rest. New terminals with attached printers can even be added to

[2] Frontend printing does not only apply to Windows frontends. It is supported on other platforms as well.

[3] Local print functionality is only available in Release 3.x, not in 4.x.

Figure 2 The R/3 Print Pop-Up with "LOCL" Entered As the Output Device



Output device	LOCL	Default printer connected to frontend ...
Number of copies	1	

Spool request

Name	RSMON000_FUC
Title	
Authorization	

Spool control

- [✓] Print immed.
- [ ] Delete after print
- [✓] New spool request

Retention period	8 Day(s)
Archiving mode	Print

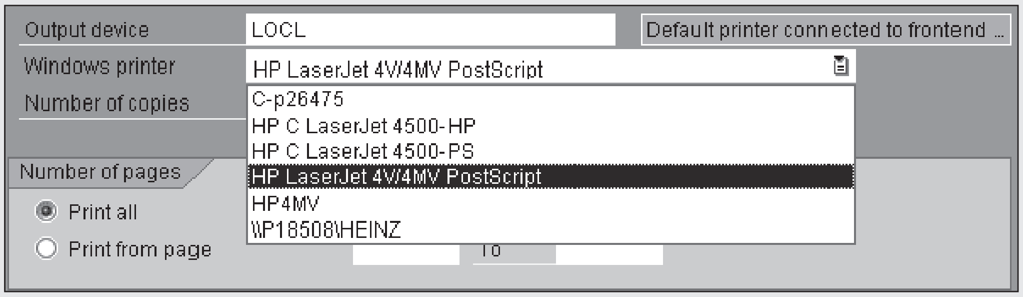
Cover sheets

D	SAP cover sheet
Recipient	FUCHSS
Department	

Output format

Lines	65	[ ] Footer
Columns	128	
Format	X_65_132	ABAP/4 list: At least 65 rows by 132 col

Figure 3 R/3 Release 4.6A Dropdown Box of Available Windows Printers



Output device	LOCL	Default printer connected to frontend ...
Windows printer	HP LaserJet 4V/4MV PostScript	
Number of copies		

```
C-p26475
HP C LaserJet 4500-HP
HP C LaserJet 4500-PS
HP LaserJet 4V/4MV PostScript
HP4MV
\\P18508\HEINZ
```

Number of pages

- (●) Print all
- ( ) Print from page To

the R/3 System without any additional configuration work necessary in the R/3 spooler.

Frontend printing is not limited to R/3 Release 4.6A. By applying the proper kernel patches and Hot Packages (which I will describe in detail later in this article), you can incorporate frontend printing — with some limitations — into your printer landscape in any R/3 System starting with 3.0D.

How does the spool access method "F" work? How does it remedy the end-user problems listed at the very beginning of this article? How does it eliminate the need for multiple output device definitions for all users who want to print to their frontend? What do administrators need to know in order to ensure its successful deployment? What are its benefits and limitations? These are questions that will be answered in this article as we review:

- The fundamental differences between printing data to a frontend printer with spool access method "F" versus printing to a frontend printer via traditional host spooler access methods like "U," "S," "L," and "C"
- The benefits and limitations of the new spool access method "F"
- What administrators need to know about different versions of the four components that support frontend printing — the R/3 kernel, ABAP-based communication methods, the SAPGUI/LPRINT RFC-Server, and SAPLPD
- A typical R/3 Release 4.x frontend printing setup for a Windows 32-bit environment, using SAPLPD
- Defining an output device for a frontend printer in a 3.x system
- Configuring non-standard systems

Frontend Printing with Method "F" — More Flexible Than Methods "U," "S," "L," or "C"

Those of you who support Release 3.0 R/3 environments have basically two choices for configuring an output device to work off a user's frontend. You can use a network access method like "U" or "S," or you can use a local access method, like "L" or "C." Neither is ideal:

- **Network access methods:** Whether you use method "U" (for the Berkeley RFC1179 protocol) or method "S" (for SAP's own protocol), you have to enter the IP address (or the name that gets resolved into an IP address) of the target computer where the LPD is running and waiting for print jobs. You also have to enter the name of the target printer. Based on this IP address and target printer name, there is a one-to-one fixed mapping between an output device definition and a physical printer. This means there has to be one output device definition in R/3 for each and every frontend where printing to a personal printer is to be supported.

  Needless to say, while it is *possible* to employ these network access methods for frontend print requests, it is not recommended. Any time a user switches off his or her PC, you run the risk of network timeouts at your R/3 spool system, which can seriously impede normal printing operations.

- **Local access methods:** With the local access methods "L" and "C," you only need to enter the name of the target printer, which is defined on the local R/3 application server.[4] There is no need to enter the IP address information because this information is stored in the remote printer definition on the local application server.

  In the spool system of that application server, you once again have to enter the name of the target host (or the IP address) and the printer name on that host. This again implies a one-to-one fixed mapping of an output device definition to a physical printer. The only benefit of a local access method over the network access method is that network problems have no influence on the R/3 spool work process; it doesn't have to deal with network timeouts in such a configuration. This means other R/3 print jobs are processed even if there are network problems with some frontend PCs.

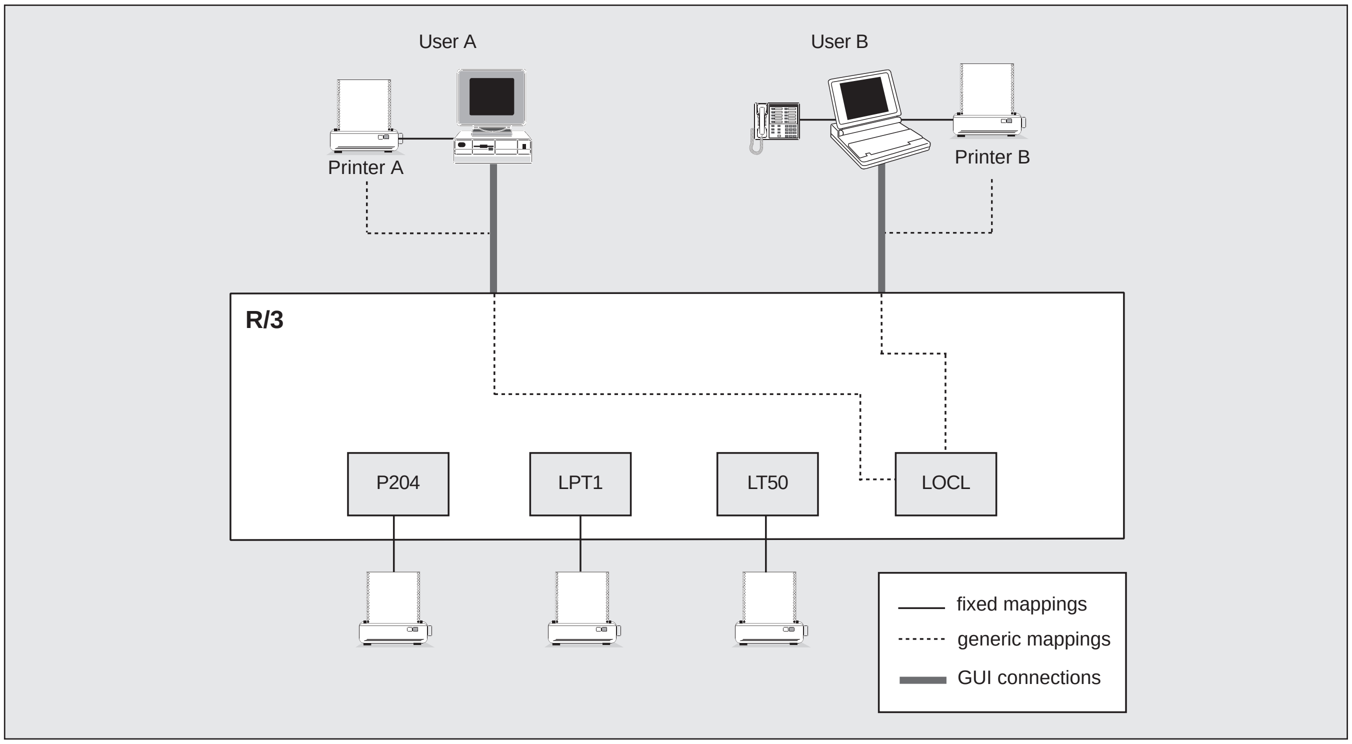
Of the two options, local access methods are the better configuration alternative, since they do not delay other print requests the way that network access methods can. They are still bound, however, by that cumbersome one-to-one fixed mapping between an output device definition and a target printer.

Flexibility Through Generic Mapping

In order to get rid of that fixed mapping, method "F" does away with the need to enter a dedicated printer name and, more important, does away with the need to enter IP address information within the output device definition. Here's how:

[4] Note that access method "C" is only available for NT and AS/400 platforms.

Figure 4 *Fixed Mapping vs. Generic Mapping of Frontend Output Devices*



- **To eliminate the need to enter a dedicated printer name for every frontend output device,** method "F" addresses the default printer on the frontend with a generic name, "__DEFAULT"(that's two underscores before DEFAULT).[5]

  The mapping of this generic name to the actual default printer is done by SAPLPD. SAPLPD, and SAPLPD alone, recognizes __DEFAULT as a special name and maps it to the default printer. Other LPDs will *not* recognize __DEFAULT. They will handle __DEFAULT as a normal printer. That means if you want to take advantage of this generic naming feature, you will have to use SAPLPD as your LPD on the frontend.

  Note that the __DEFAULT name is not specific to method "F." It can be used with any access method, but access method "F" is the place where this generic name is most useful with regard to frontend printing.

- **To eliminate the need to enter IP address information in the output device definition,** method "F" stores terminal information with each print job, making it possible for print data to be delivered to the current frontend location where the user is running his or her GUI. This frees the output device definition in R/3 from any IP address information. *This* is the foundation upon which we built access method "F"; the __DEFAULT name is merely a convenience.

Once you eliminate the need to enter IP address information in the output device definition, you move away from one-to-one fixed mappings and can adopt generic mappings, which are far more flexible, manageable, and scalable.

In **Figure 4** you can see the difference between fixed and generic mappings. The output devices

[5] In older releases, this was "%DEFAULT%," which should no longer be used in any release, as it is incompatible with UNIX environments.

"P204," "LPT1," and "LT50" are defined using one of the traditional access methods (network or local). Note that each output device represents exactly one physical printer, since, by definition, their fixed output device definitions contain the IP address of the target spooler.

"LOCL" is an output device that is defined with access method "F." *There is no IP address associated with this output device.* So it can be mapped to an unlimited number of destinations. In this case, it's just being mapped to two destinations — User A and User B. That number could be 200 or even 2,000.

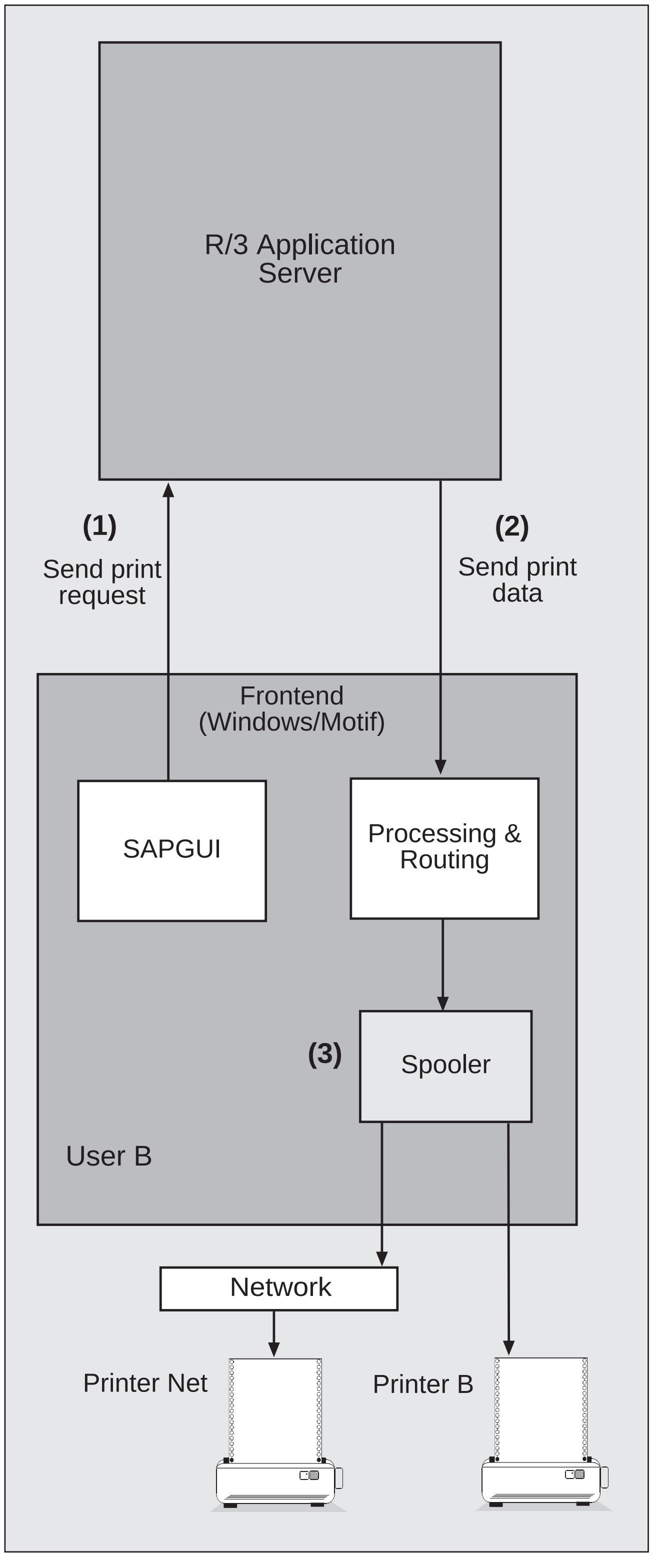
The destination information is determined by the terminal (i.e., the GUI connection) where the print request comes from, as shown in **Figure 5**. In this example, the print request is coming from User B, so the print destination is understood to be User B's default printer, *Printer B*. If User B had made *Printer Net* the default printer, the output would have been sent there.

Even though Users A and B have chosen different names for their default Windows printers ("Printer A" and "Printer B" respectively), both use the same *logical* output device — the one we have named "LOCL." Provided that the administrator defined the logical output device with the generic name "__DEFAULT," all a user has to do in order to use this common, generic output device is make his or her intended Windows printer the default printer, and print to the output device named "LOCL."

On the server side, the administrator has to make the proper output device definition for output device "LOCL." You have to make sure that the required Hot Packages and kernel patches have been installed. Each client must have either a recent SAPGUI or LPRINT installed. SAPLPD should also be available for the most convenient printout facility.

I'll provide much more detail on these administrative activities later on in this article.

Figure 5 ***Processing of Frontend Print Request***



But What About Different Printer Types?

In R/3, every output device has an associated device type (e.g., PostScript, PCL, or Prescribe). So how does R/3 know which of these device types is associated with the current user's default printer? This is fundamental to being able to have only one output device definition.

Here, the R/3 device type SWIN (SAPWIN in older releases) comes in very handy. SWIN is a device type for Windows printers. This device type is interpreted by SAPLPD, which translates it into commands to the Windows Graphics Device Interface (GDI). This way, every Windows printer that has a Windows printer driver can be used independently of the exact device type. Note that this feature once again requires that you use SAPLPD, as other LPDs will not understand device type SWIN.

Do you always have to use SWIN for frontend printing? No. Using SWIN makes the most sense in environments where users have different types of frontend printers. If you are in an environment where every user has a PostScript printer, you can choose device type POSTSCPT[6] in the definition of the frontend output device instead of SWIN. You can also define multiple frontend output devices (one for each device type that you must support) at the same time if you have a heterogeneous environment.
In this case, however, the user must know which frontend output device has a suitable device type for his or her personal printer.

To eliminate the need to enter IP address information in the output device definition, method "F" stores terminal information with each print job, making it possible for print data to be delivered to the current frontend location where the user is running his or her GUI. This is the foundation upon which we built access method "F."

[6] You can choose POSTSCPT or POST2 as of Release 4.5A.

Benefits and Limitations of Frontend Printing

I hope it's clear that both end users and administrators alike stand to benefit from this new print spool access method. R/3 users now have a way to print to the very same printers that support printing from their Windows applications. This means:

- ✔ Users who previously found they could only print to a fixed printer name are now able to print to their current default printer. By changing the default printer in Windows, any printer can be used.

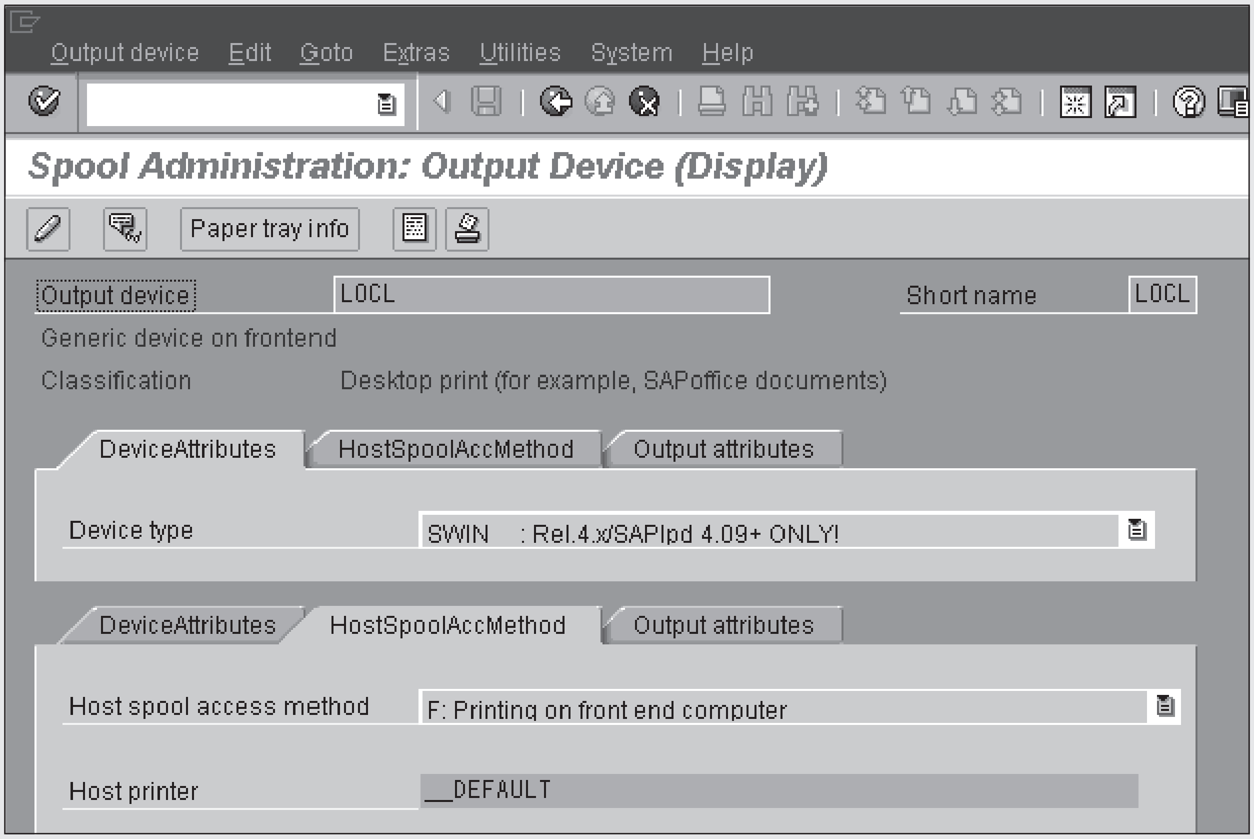
- ✔ As I mentioned earlier, Release 4.6A takes this one step further. A new dropdown box makes it possible for users to interactively point-and-click on any printer that has been defined on the current frontend. The list of available printers is retrieved by an RFC call to the frontend (SAPGUI or LPRINT), which returns all defined printers. This list is displayed in an additional dropdown box when the user enters the output device name of a frontend output device (e.g., "LOCL").

  The Windows printer name that the user finally selects is stored together with the spool request that was created when the user was prompted for an output device name.

  The R/3 host printer name that is associated with the output device definition only serves as a default in this case. If this particular name does not exit on the current frontend, or if it is the special name __DEFAULT, the default printer will be used automatically.

- ✔ Users who work on a different terminal each day and who require local output are now able to use the same output device name, no matter what their current frontend is or where they are located. They no longer have to locate the specific output device name that is associated with their current terminal.

Figure 6 Typical Definition of a Frontend Output Device in SPAD (Release 4.6A)



Spool Administration: Output Device (Display)

Output device LOCL Short name LOCL

Generic device on frontend

Classification Desktop print (for example, SAPoffice documents)

Device type SWIN : Rel.4.x/SAPlpd 4.09+ ONLY!

Host spool access method F: Printing on front end computer

Host printer __DEFAULT

✔ Users on terminals with dynamic IP addresses (e.g., laptops that connect by a phone line) can now print in the same way that LAN-connected users with fixed IP addresses print. With the traditional host spool access methods, this was not possible because the IP address information was stored together with the output device definition. Each time you logged on with a different IP address, you would have to use a different output device! Even if you used computer names instead of IP addresses, you would experience the same problem because R/3 keeps an internal cache with the IP addresses. (This cache is normally not deleted, which means that a new connection from the same host, but with a different IP address, is not recognized.)

Administrators now have an easy way to set up frontend printing for a large number of users. You no longer have to configure countless individual frontend output devices. This provides three *significant* administrative benefits:

✔ The first, and perhaps most important, is that a single generic output device definition is now sufficient for *all* frontend users.[7] This generic output device definition is made with transaction SPAD, specifying the new access method "F," as shown in **Figure 6**. In the interest of space, I have combined two screen shots into one. In the top portion of the tab section, the **DeviceAttributes** tab is active, and in the bottom

[7] Provided that they use the same device type (e.g., SWIN for Windows frontends).

portion, the **HostSpoolAccMethod** tab is active. In the actual system, these separate active tabs would have to be viewed on different screens.

✔ New terminals with attached printers can be added to the R/3 System without any additional configuration work necessary in the R/3 spooler. A big area where this feature is currently being leveraged is SAPnet (the former OSS system of SAP). Every customer can print in this system using the output device name "LOCL," which is defined with access method "F."[8]

✔ Lastly, frontend printing is a convenient way to allow DHCP (Dynamic Host Configuration Protocol — i.e., dynamic IP addressing) clients to print to the frontend location. Access methods specifying terminal names that get resolved to IP addresses (e.g., "S" and "U") do not work in this case, as IP addresses are only resolved once by R/3 and cached afterwards. So R/3 will not recognize any change in IP addresses in the future until R/3 is restarted (or the cache has been reset manually). Whether or not local access methods "L" and "C" work in such an environment is up to the underlying operating system, as it is the operating system that has to resolve the dynamic IP addresses correctly in this case.

This Is Not the Frontend Printing of Yesteryear — Still, It's Got Limitations

Our original design envisioned frontend printing as a solution to print relatively small documents, online, to the current frontend location. But users soon began using method "F" for printing from the update task and batch jobs, and presenting us with a variety of other challenges that we had not forseen. Some we were able to tackle. Some we were not.

[8] If you're wondering about the derivation of the name "LOCL," it was completely arbitrary. Here, at SAP, we agreed to make it the name of the frontend output device for Windows (device type SWIN). This is just a convention. The name can be different in your own System.

Let me start by listing three limitations of frontend printing that we did not address in Release 4.6 and that you should be prepared to live with:

- The first is that frontend printing cannot be used to get an *immediate* printout from a batch job. During batch processing there is no connection to a GUI, therefore you cannot print to the GUI location. In Release 4.6A, the output request is generated, but it is not processed. Processing of this request will start as soon as the user begins a new frontend print request in a dialog (or in the update task). The user can also re-trigger print job processing manually via transaction SP01. In older releases, the use of a frontend output device in batch processing caused runtime errors.

- The transfer of the print data to the frontend requires a new session, which means that frontend printing only works if the user is not using all available sessions. If a user has more than five open sessions for the same logon, frontend printing will not work. As is the case with batch processing, such a request will not get processed; it will either restart automatically the next time a frontend printout succeeds, or it will be manually restarted from transaction SP01. (In the first release, each concurrent printout of a user required a separate session. This requirement has been reduced to one session for all printouts, and printouts for one user are serialized.)

- Lastly, there is no status feedback inside R/3 for frontend print requests. Once the processing of the print request is done in R/3 and the data is transferred successfully to the frontend, the print job is set to a "done" state. Any further errors that may occur on the frontend side are not reported back to R/3.

While the limitations associated with batch processing, new sessions, and status feedback still persist (the latter two may be solved in the future), the limitations listed below *have* been solved in the 4.5A and 4.6A releases:

- Printing from the update task had the same problem as printing from batch — there was no connection to the GUI from the update task. This problem was addressed with R/3 Release 4.5A. Now the update task inherits the GUI connection information, making it possible to re-establish the GUI connection for a user *who is still logged on* at the time when the output request is processed by the spool system.

- Formatting of the output data for frontend printing up to 4.5B takes place in a dialog process. In a system with a standard configuration, dialog tasks have a timeout limit of five minutes. If the formatting process for a large document takes more time, the job will run into a timeout, and frontend printing will fail. As of R/3 Release 4.6A, formatting will be done by a spool work process, which offers infinite processing time. This means that even large documents can now be printed using this method.

- Since formatting of the print data takes place in a dialog process, the dialog workload will increase significantly if frontend printing is used by many users at the same time (or in the first release of frontend printing, up to Release 4.0B, even if one user prints multiple times). Moreover, the transfer of the print data to the GUI temporarily requires a second dialog process. In the first implementation (3.1G), three dialog processes were used for a very short period of time — i.e., when frontend printing is initiated. As of Release 4.6A, processing will be done by a spool work process so that the dialog load in the system is not increased.

- The first implementation of frontend printing did not support multiple copies. Instead, the frontend output device had to be defined with the flag "Send each copy as own print request" in transaction SPAD. This limitation caused a tremendous load in the old release, as *n* sessions and *n* dialog processes were required. Because of this, we disabled this flag in SPAD and implemented a copy count for frontend print requests (Release 4.5B).

Let me summarize this benefits/limitations discussion by saying that frontend printing is most useful in a Windows environment where it gives the user the ability to print to his or her default printer, and as of R/3 Release 4.6A, to *interactively* select a frontend printer.

Four R/3 Components Support Frontend Printing

Frontend printing is built upon the following four system parts:

- The R/3 kernel
- ABAP-based communication methods
- SAPGUI/LPRINT RFC-Server
- (SAP)LPD[9]

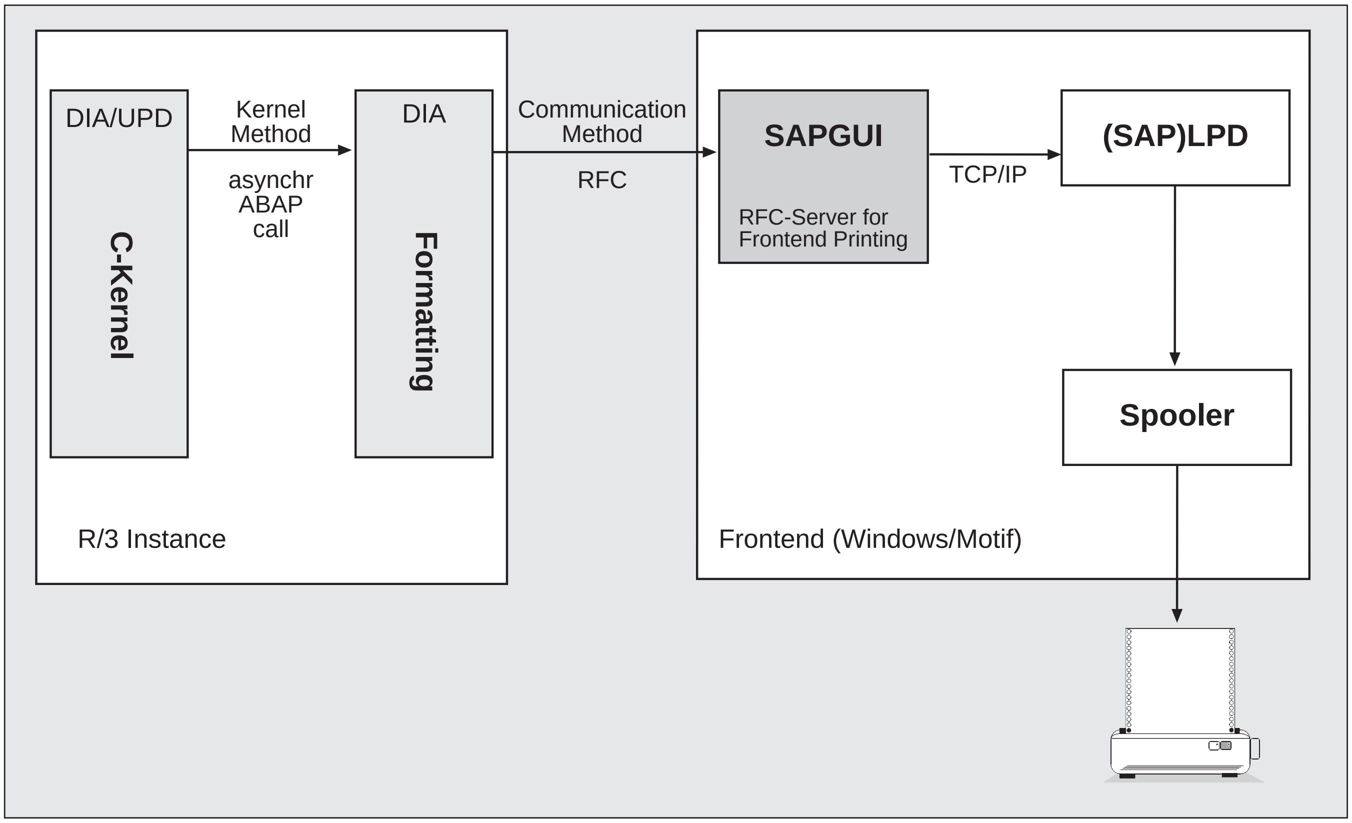
Figure 7 illustrates how a typical Release 4.0B or Release 4.5A frontend print request would utilize these four components. The steps are outlined as follows:

1. The print request is generated by the R/3 kernel, where it runs in a dialog process, or, as of Release 4.5A, as part of the update process.

2. The print request is passed by an asynchronous ABAP call to a second dialog process. This dialog process takes care of the formatting of the print request and the transfer of the print data to the frontend (communication method). The way the print request is passed from the kernel to the dialog process is called the "kernel method." The new process is used here in order to de-couple the generation of the print request, which is done by the application, from the actual outputting of the print data. Otherwise, the application would be hampered in its progress.

[9] The use of the parentheses here is to indicate that either SAPLPD or just an LPD can be used.

Figure 7 ***Components Involved in Frontend Printing***



3. Now the print data is transferred to the frontend using RFC calls to SAPGUI or LPRINT, which act as the RFC-Server for frontend printing.

4. The SAPGUI/LPRINT RFC-Server transfers the data to the locally running (SAP)LPD using the "localhost" IP address (127.0.0.1). If SAPLPD is not running, it will be started automatically at this point. Other LPDs are not started automatically.

5. (SAP)LPD prints the data to the Windows Spooler.

As administrators, you need to understand what new features/functions we have introduced, over the course of various releases, into all of the following: the kernel; the ABAP code that is responsible for formatting and communication; the frontend software (consisting of the SAPGUI/LPRINT RFC-Server); and SAPLPD. Armed with this knowledge, you can apply the right patches and/or perform the appropriate upgrades.

As administrators, you need to understand what new features/functions we have introduced, over the course of various releases, into all of the following: the kernel; the ABAP code that is responsible for formatting and communication; the frontend software (consisting of the SAPGUI/LPRINT RFC-Server); and SAPLPD. Armed with this knowledge, you can apply the right patches and/or perform the appropriate upgrades.

Figure 8   Features and Required Components*

	Kernel	ABAP `	Frontend		
			SAPGUI	LPRINT	SAPLPD
Kernel Methods					
Direct Execution	3.1G 3.1G	3.0D/58 3.0F/46			
Buffered Execution	3.1I 3.1G	3.0D/58 3.0F/46			
Frontend Print Server	3.1I/123 4.0B 4.0B	3.0D/58 3.0F/46 3.1H/29 3.1I/7			
Print Messages	4.6A	4.6A			
Communication Methods					
LOCAL_PRINT		3.0D/58 3.0F/46 3.1G		20.1	
SAPGUI		3.0D/58 3.0F/46 3.1H/29 3.1I/7 4.0B	4.0A Win32** 4.0B Mac		
DOWNLOAD (LPRINT command line interface)		3.0D/58 3.0F/46 3.1H/29 3.1I/7 4.0B/5 4.5A		20.25	
Other Features					
Printing from update task	4.5A				
Delay print request if used in batch (or update task prior to 4.5A)	3.1I/205 4.0B/289 4.5A/58 4.5B				
Binary data transfer		3.0D/58 3.0F/46 3.1H/29 3.1I/7 4.0B/5 4.5A/1 4.5B	4.0B Win32 4.0B Mac	20.27	
Process failed (delayed) print requests		3.0D/64 3.0F/52 3.1H/35 3.1I/12 4.0B/9 4.5A/3 4.5B			
Command execution			4.6A Win32 4.6A Mac	21.03	
Recognize __DEFAULT					2.44
Start SAPLPD iconified					4.03
Device type SWIN		4.0A			4.09
WTS & printer names with spaces					4.24

* Highlighted items are used to support the example configurations discussed in the text.

** The 4.0A GUI works only for non-binary transfer. The Hot Packages mentioned try **only** binary transfer. So the combination will not work. It is therefore recommended that you use at least the 4.0B SAPGUI version.

Figure 8 provides a summary of the features that have been added from the first implementation of frontend printing, along with the patches you need in order to leverage these features (wherever possible) in older releases.

In order to support frontend printing, you need to select a kernel method and a communication method. What makes this whole affair so complicated is the fact that you can introduce most features by applying some kernel patches, Hot Packages, and updates of frontend components, but these three areas sometimes depend on each other.

Support for the other features is generally optional, and is only required in certain environments (e.g., binary data transfer is required if not all components use ASCII code page).

A few examples will help illustrate what needs to be done:

- **Example 1:** You want to use the Frontend Print Server as your kernel method and the SAPGUI as your communication method in a 3.1I system. This requires that you use at least 3.1I kernel patch level 123, 3.1I Hot Package 7, and a 4.0B Win32 SAPGUI. In Figure 8, I have highlighted the choices required in this example.
- **Example 2:** You additionally want to be able to print from the update task. This feature is only available as of kernel Release 4.5A. So you need to upgrade your R/3 in this case. In Figure 8, I have highlighted the "Printing from update task" selection.

The R/3 Kernel

The R/3 kernel offers four different methods for processing frontend print requests inside R/3. In the previous example, we opted to use the Frontend Print Server as our kernel method. The method that will be used by the kernel can be configured by the profile parameter **rspo/local_print/method**. **Figure 9** lists

Figure 9 Four Kernel Methods for Processing Frontend Print Requests

Method	Value	Kernel Release
Direct Execution	0	3.1G
Buffered Execution	1	3.1I
Frontend Print Server	2	4.0B, 3.1I PL 123
Print Messages	3	4.6A

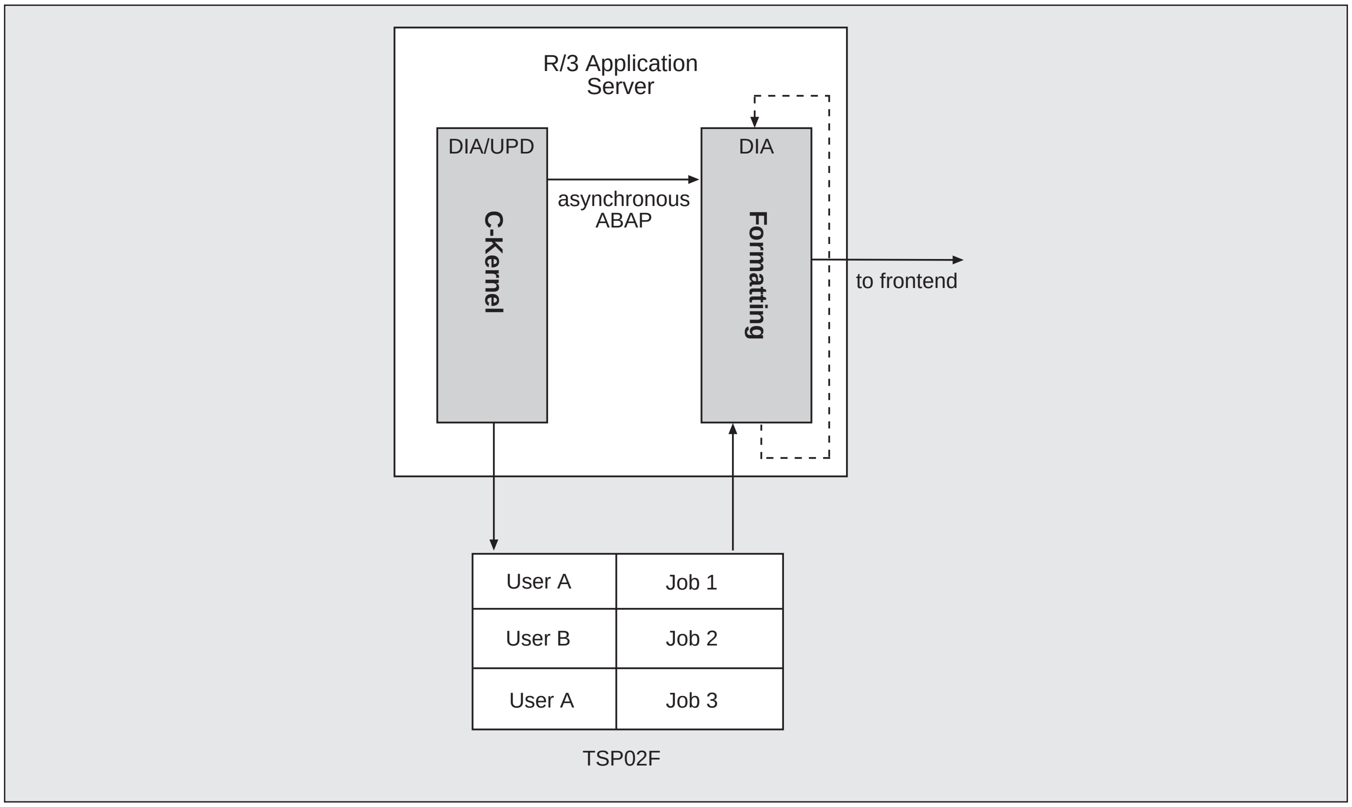
the possible values for that parameter. The default is always to use the newest configuration method that is available with the current kernel.

How do you know which kernel method is best for your environment? Let's review them…

Direct Execution was introduced with Release 3.1G. After the print job has been created by the kernel, an asynchronous ABAP call is performed immediately to process the job in a second dialog process. This ABAP call decouples the formatting process of the print job from the generation of the print job, which speeds up the original transaction. This call also facilitates the RFC call that transfers the data to the GUI. Unfortunately, this method's asynchronous RFC performs an implicit COMMIT on the application, which makes it impossible to perform ROLLBACK after creating the print request. This problem was solved with the "Buffered Execution" method. Note that Direct Execution cannot be used when it should be possible to print from the update task.

Buffered Execution was established to collect all frontend print requests (like normal print requests) up until the time that the application COMMITs. So we introduced a special (limited) queue, where all frontend print requests get stored until COMMIT. The length of this queue can be configured by the profile parameter **rspo/local_print/jobs** (default 50). At COMMIT, an asynchronous ABAP call is performed for each print request. The parameter **rspo/local_print/wait** can be used to limit the

Figure 10 Frontend Printing with TSP02F and the "Frontend Print Server" Method



number of concurrent asynchronous ABAP calls. The parameter specifies the maximum amount of time, in seconds, that the system should wait for the current printout to terminate before the next one is started by an asynchronous call. If the current printout does not terminate in the given time, a new asynchronous call is started for the next job.

Both of these methods (Direct and Buffered Execution) have the disadvantage that each print job gets processed by an individual dialog process. (Parameter **rspo/local_print/wait** can limit this.) This means that a single user could potentially clog up the whole system if he or she prints several frontend jobs at once. This is solved with the Frontend Print Server method.

The **Frontend Print Server** method was designed to limit the number of dialog processes a single user could occupy. To do this, we created a new table, TSP02F, which contains all frontend print requests (of all users) whose jobs require processing. The kernel now just enters a print request into table TSP02F and calls the asynchronous ABAP program after COMMIT. That ABAP program processes all jobs for the current user that can be found in table TSP02F. (In **Figure 10**, for example, jobs 1 and 3 would be processed for User A.)

Whenever a second print request is started for a user, a check is made to see if there is already another dialog that is busy processing frontend requests (using ABAP enqueues). If there is, the asynchronous ABAP program terminates immediately and the running frontend print server will pick up this request in the next scan of the database before terminating itself. This gives us a single running frontend print server that processes all of a user's jobs. If no jobs are left, it simply terminates.

The good news is that this kernel method reduces the number of required dialog processes for each user to just one.[10] (Note that when the transfer of the print data to the frontend is initiated, a second dialog is required for a very short period of time.)

Another benefit is that frontend print requests do not get lost, even if the job cannot be processed immediately. Jobs can get lost if the user runs out of sessions or starts frontend print requests in batch jobs. With the kernel method, however, the TSP02F entry is written in the database and only the asynchronous ABAP call for the processing of that job fails, not the job itself. The print job will be processed as soon as another frontend job for that user has been processed successfully (e.g., the user closes one session and prints again). The old request gets processed because the frontend print server scans the entire table, TSP02F, for jobs that need processing. In this way, all older jobs that failed before will now also be printed for that user. If the user just wants to get all failed frontend jobs out of the system without printing a new job, he or she can do so by calling the transaction SP01 menu item "Restart Frontend Jobs" (available since 4.5A). In older releases, the RSPO_PROCESS_FRONTEND_JOBS function module can be used without any parameters specified.

The **Print Messages** method is currently the reigning solution for addressing very high dialog loads. It processes frontend print requests in the spool work process just like ordinary print jobs. This method requires that the GUI connection information be transferred from the dialog process to the spool work process over the dispatcher or message server. Note that new development work was done to support the Print Messages method in Release 4.6A, and that it cannot be transported into older releases.

While the other kernel methods require no spool work process for frontend printing, the Print Messages method requires at least one spool work process in the system. Although it is not necessary for each application server to have its own spool work process, this configuration reduces communication overhead between application servers in the R/3 System, so it is recommended that a spool process be configured for each application server where frontend printing will be done. If you have an application server that does not have a spool work process, you can predetermine which spool server should handle requests that are generated on this server. The profile variable **rspo/local_printer/server** specifies the server (real or logical) that gets those print messages. If you do not configure anything for this variable, the spool work process with the lowest load will process the frontend print request.

[10] If numerous users employ this print method simultaneously, the dialog load can still be prohibitive.

It is also possible to restrict frontend printing to a certain number of spool work processes on each application server. Otherwise, if all spool work processes are busy with frontend printing, urgent productive print requests may not get processed in time.[11]

The default for the maximum occupancy of spool work processes is 1. The profile parameter **rdisp/wp_no_spo_FRO_max** can be used to increase this limit. For example, if you have five spool work processes on one application server and set **rdisp/wp_no_spo_FRO_max = 2**, at least three spool work processes will be available for non-access method "F" printing. In this example, increasing this profile parameter to "2" does *not* guarantee that two spool work processes will always be available for processing frontend print requests. Higher-priority jobs take precedence. It's possible that no spool work process may be available for frontend printing, for example, if all five processes are being used for productive printing at the same time.

We used to consider frontend printing to be a very low-priority task, but user requirements have changed. This is why, as of Release 4.6B, there is another parameter, **rdisp/wp_no_spo_FRO_min**, which defines how many spool work processes are *reserved* for frontend printing and *cannot* be occupied

[11] For a comprehensive discussion about classifying/configuring your output landscape for optimal support of production, high-volume, desktop (frontend), and test printing, please refer to the article "Achieving a More Manageable and Reliable R/3 Spool Server Landscape Using Release 4 Output Classifications, Logical Servers, and Alternate Servers," which also appears in this issue of the *SAP Professional Journal*.

Figure 11 ***Summary of Profile Parameters for Frontend Printing***

Parameter	Default	Description
rspo/local_print/method (3.1I)	Defaults to newest available method	Four kernel methods (see Figure 9) are available for processing frontend print requests. The "Print Messages" method is the newest.
rspo/local_print/server (4.6A)	Use spool server with lowest load	Specify spool server for the "Print Messages" kernel method if application server has nospool work processes.
rspo/local_print/jobs (3.1I)	50	Number of jobs that can be stored until COMMIT for the "Buffered Execution" and "Frontend Print Server" methods.
rspo/local_print/wait (3.1I)	100	Time (in seconds) that should be waited for completion of print request before next printout is started (only applicable for the "Direct Execution" and "Buffered Execution" methods). Can be used to limit the number of concurrent printouts.
rdisp/wp_no_spo_FRO_max (4.6A)	1	Number of spool work processes that can handle frontend print requests when using the " Print Messages" method.
rdisp/wp_no_spo_FRO_min (4.6B)	0	Number of spool work processes that cannot be occupied by services other than frontend printing when using "Print Messages."
abap/no_sapgui_rfc	0	Disables RFC connections to the frontend if set to "1."

by other services. (The default for this parameter is 0.)

Figure 11 provides a summary of all the different profile parameters.

The print message configuration issues basically boil down to two things:

- You should increase the number of spool work processes by 1 (or by the value of **rdisp/wp_no_spo_FRO_max**, if set differently).
- You should ensure that every application server with dialog or update processes has a spool work process.

Of the four kernel methods — Direct Execution, Buffered Execution, the Frontend Print Server method, and the Print Messages method — which should you use? The optimal kernel configuration, Print Messages, comes with Release 4.6A. It enables you to use the spool work process for frontend printing *without* posing additional dialog load on your system. All you have to do is set up a suitable number of spool work processes such that your normal print jobs do not get delayed by frontend printing. If you cannot upgrade to Release 4.6A, the next best thing is to use the Frontend Print Server method. This method will be used automatically if you are using the kernel patches and Hot Packages mentioned in Figure 8.

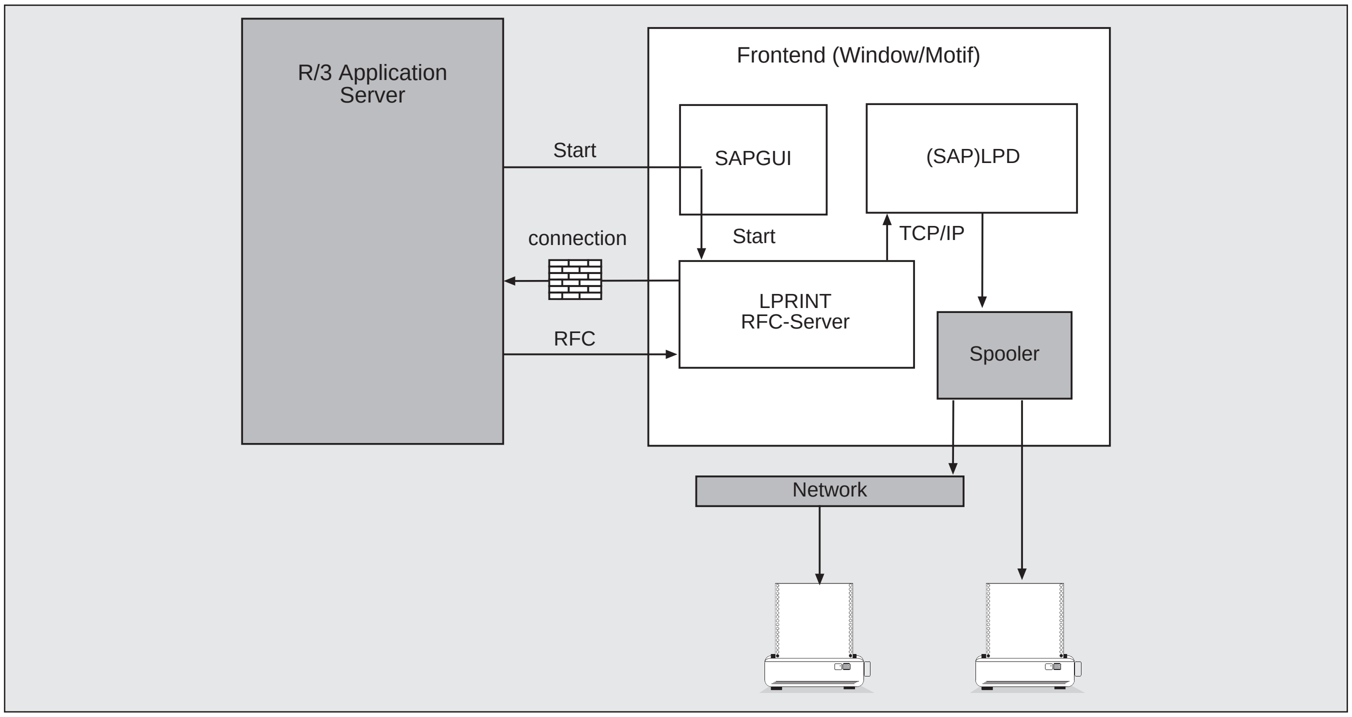
Communicating with the Frontend

In addition to selecting a kernel method to support frontend printing, you also need to select a communication method. The communication from R/3 to the frontend can be handled in three different ways:

1. **LOCAL_PRINT:** LOCAL_PRINT was the first implementation of the R/3-to-GUI communication component. It uses a dedicated RFC-Server named "LPRINT," which has to be installed in the SAPGUI directory. In R/3, the RFC destination LOCAL_PRINT (see transaction SM59 TCP/IP connections) was created so that the RFC-Server LPRINT gets started automatically each time a printout is requested.[12]

[12] This RFC destination is hardcoded in the R/3 kernel. It can only be changed temporarily and will be reset to its original value each time work process #1 gets restarted (e.g., any application server is restarted).

Figure 12 *The "LOCAL_PRINT" Communication Method*



Running on the frontend, the RFC-Server connects to the R/3 System via RFC calls, and with the (SAP)LPD using a local TCP/IP connection, as shown in **Figure 12**. The problem with this method is represented by the brick wall. The connection from the frontend to the R/3 System results in a new connection — the GUI connection is always open as long as the GUI is running, so starting a new RFC server on the frontend establishes a new connection. And this new connection will fail if a SAProuter is involved that is configured to use a "one-time" password, or if the passwords of the SAProuter and the SAPgateway are different. This is a general limitation of the RFC mechanism, where the frontend server is started automatically by the SAPGUI. It is not a special frontend printing problem. This RFC limitation is what led to the design of the SAPGUI method.

2. **SAPGUI:** From SAPGUI 4.0A onward, the functions of the RFC-Server LPRINT were integrated into the SAPGUI. This way, the destination SAPGUI (see transaction SM59 TCP/IP connections) acts directly as the RFC-Server for frontend printing. The big advantage of this integration is that the current GUI connection can be used to transfer the data. There is no need to establish a new connection, and, as a consequence, there are no problems with SAProuters. You also do not have to worry about installing a separate program. This is the most secure method, but it is only available for Windows 32-bit platforms (and, as of Release 4.0B, for the Macintosh). Because of this limited availability we had to create a third communication method (described next) that works on all platforms and on network configurations where LPRINT could not be used.

3. **DOWNLOAD & EXECUTE:** The DOWNLOAD & EXECUTE method works very much the same way that the old local print function from Release 3.0A worked. The print data is downloaded to the frontend. Then a program is executed that processes this print data. The only difference is that we do not execute SAPLPD for each print request, but

instead start LPRINT (a much smaller program), which also has a command line interface to do printouts. This means LPRINT can act as an RFC-Server and has a command line interface at the same time. Then LPRINT sends the spool data to the currently running (SAP)LPD using local TCP/IP. The drawback of this method is that it cannot be used from within the update task or the spool work process (Release 4.6A, Print Messages kernel configuration method). Another disadvantage is that it is not possible to monitor whether the data really arrives at the LPD. As soon as LPRINT is successfully started, the print job goes to status "Done."

If not configured otherwise, frontend printing will try to use the communication methods in the following sequence: SAPGUI, LOCAL_PRINT, DOWNLOAD. By making an entry with key LPRINT_DEST in table TSPOPTIONS, the starting point for this sequence can be changed. If, for example, you enter LOCAL_PRINT as LPRINT_DEST, the SAPGUI destination will be skipped, and only LOCAL_PRINT and DOWN-LOAD will be tried. As of Release 4.6A, this setting can also be changed with the transaction SPAD path **Settings → Spool System → Frontend Printing → RFC destination for frontend printing**.

To wrap things up, the SAPGUI communication method is the most secure and reliable communica-tion method, provided you have installed the neces-sary Hot Packages and GUI release. It will be used automatically if you have not previously configured another method. You should only consider using one of the other methods in the f ollowing cases:

- **You have reasons that disallow you to use at least a 4.0B SAPGUI.** In this case, you can update the standalone program LPRINT on each frontend and use the LOCAL_PRINT destination. It is a good idea to create the entry in table TSPOPTIONS mentioned above to disable the SAPGUI.
- **You are not using a Windows or Macintosh frontend.** For a solution, please see the section entitled "Configuring Non-Standarad Systems," which appears later in this article,
- **You are using the Windows Terminal Server.** Again, refer to the "Configuring Non-Standard Systems" section.

SAPGUI

As of Release 4.0A, the SAPGUI for Windows 32-bit platforms (4.0B for Macintosh) was extended in such a way that it can serve as an RFC-Server for frontend printing. This means you do not need to install a separate program on your frontend in order to transfer the print data by RFC calls. As of Release 4.6A, a new call was added that allows the querying of printer names that are defined on the frontend — this allows users to interactively select printers, as you saw earlier in Figure 3.

Other GUI platforms currently do not support these RFC functions for frontend printing. You would have to use the separate RFC-Server program LPRINT described in the next section.

RFC-Server "LPRINT"

The current version of the RFC-Server program LPRINT is version 21.03. As LPRINT is downward compatible, there should be no problem using at least version 21.03 on your frontends. Older versions had the following problems:

- Data was transferred as characters, not as bytes (binary data transfer). This caused errors if the frontend and the application server were running on different code pages — e.g., application server AS/400 (EBCEDIC) and NT frontend in ASCII.
- Temporary file (created by LPRINT) was not deleted in case of error.
- Spaces in the user name or in the frontend printer name caused errors for the DOWNLOAD method (the command line interface of LPRINT).

- SAPLPD was not started automatically.
- Command execution for printing was not available.

Version 21.03 of LPRINT also offers the ability to execute any command line to do printing (very much like access method "L" in R/3). This is especially useful for UNIX environments where a normal lp or lpr command can be used for printing. This means you no longer need an (SAP)LPD to do frontend printing! Using the command execution interface is also the recommended way for frontend printing with the Windows Terminal Server (WTS).

Version 21.01 is mandatory for R/3 Release 4.6A as it offers an additional RFC function to transfer the list of known frontend printers to the R/3 System.

LPD Versions

Frontend printing can use any LPD that is running on the frontend. But remember that some features are not available with ordinary LPDs that are offered by SAPLPD. Normal LPDs cannot:

- **Process R/3 device type SWIN (SAPWIN):** This implies that the only Windows printers that can be used are ones that match the device type of the frontend output device definition. If, for example, the frontend output device is defined using device type *PostScript*, only Windows printers that understand PostScript can be used.
- **Recognize printer name "__DEFAULT":** SAPLPD recognizes this special printer name and maps it automatically to the user's current default printer. This is *very* convenient for the user and the administrator. If a user is using a normal LPD, the host printer name in R/3 and the Windows printer name must match in order to print on his or her frontend. This means that either all users have to use the same Windows name for their printers or multiple frontend output device definitions are required in R/3.

Based on these two features, I strongly recommend the use of SAPLPD as the line printer daemon for Windows platforms.

Some (UNIX) LPDs may refuse to accept connections from the RFC-Server program since this program is not running with root access and is not using secure TCP/IP ports to communicate with the LPD. In this case, the command execution interface (lp and lpr interface) of LPRINT must be used.

Helpful Hints for Configuring Frontend Printing

While it is desirable to implement frontend printing with all the most recent components, I realize it's not always possible. That's why you see so many options listed in Figure 8. As you plan an implementation that is uniquely suited for your R/3 environment, keep the following tips in mind:

✔ **Tips for configuring kernel releases:** Frontend printing requires at least a 3.1G kernel. It is absolutely impossible to use this method in older kernel releases.

The following kernel releases are recommended to get the best results for frontend printing:

3.1I: Patch level 205 or higher
- Uses "Frontend Print Server"
- Prohibits processing of print requests from batch and update task
- Uses language of current user for SAP spool title page

4.0B: Patch level 289 or higher
- Same as 3.1I

4.5A: Patch level 124 or higher
• Same as 4.0B
• Plus: Allows processing of print requests in update task

4.6A: Patch level 0
• Same as 4.5A
• Plus: Uses "Print Messages" to process requests in spool work process

✔ **Tips for configuring Hot Packages:** After the introduction of frontend printing in Release 3.1G, we created transport requests for older releases (i.e., 3.0D to 3.0F). But these transports caused lots of problems when the customer finally upgraded to a Release 3.1G or higher, which is why we discontinued this method of bringing new functionality into older releases. For the benefit of the customer and to ensure ease of use, we decided to use Hot Packages to update frontend printing functionality as of Release 3.0D. The use of the following Hot Packages is recommended to get the newest improvements for frontend printing:

3.0D:	SAPKH30D64
3.0F:	SAPKH30F52
3.1H:	SAPKH31H35
3.1I:	SAPKH31I12
4.0B:	SAPKH40B10
4.5A:	SAPKH45A05

✔ **Tips for configuring SAPGUI/RFC-Server LPRINT:** Frontend printing on the frontend side normally requires an RFC-Server program that is either integrated into SAPGUI or is a separate executable called LPRINT(.EXE). Frontend printing is integrated into SAPGUI only since GUI Release 4.0B for Windows 32-bit and Macintosh platforms. In older releases and on other platforms, the separate RFC-Server LPRINT has to be installed in the SAPGUI directory. The current version of LPRINT is 21.03 (shipped with R/3 4.6A) and can be obtained from our sapservX machines for older releases. You should use at least version 20.40.

✔ **Tips for configuring (SAP)LPD:** It is very useful to combine frontend printing with the use of SAPLPD because it offers access to the default printer and can process device type SWIN. You should use at least SAPLPD version 4.03 (the currently available version is 4.24) — version 4.03 was the first version that had the ability to start SAPLPD iconified. If you run SAPLPD as a service, you do not need to start SAPLPD iconified, and older SAPLPDs will work as well. If you are using a normal LPD, you may have problems with the local TCP/IP connection. That topic is addressed in the section entitled, "Configuring Non-Standard Systems."

A Typical R/3 Release 4.x Frontend Printing Setup for a Windows 32-Bit Environment Using SAPLPD

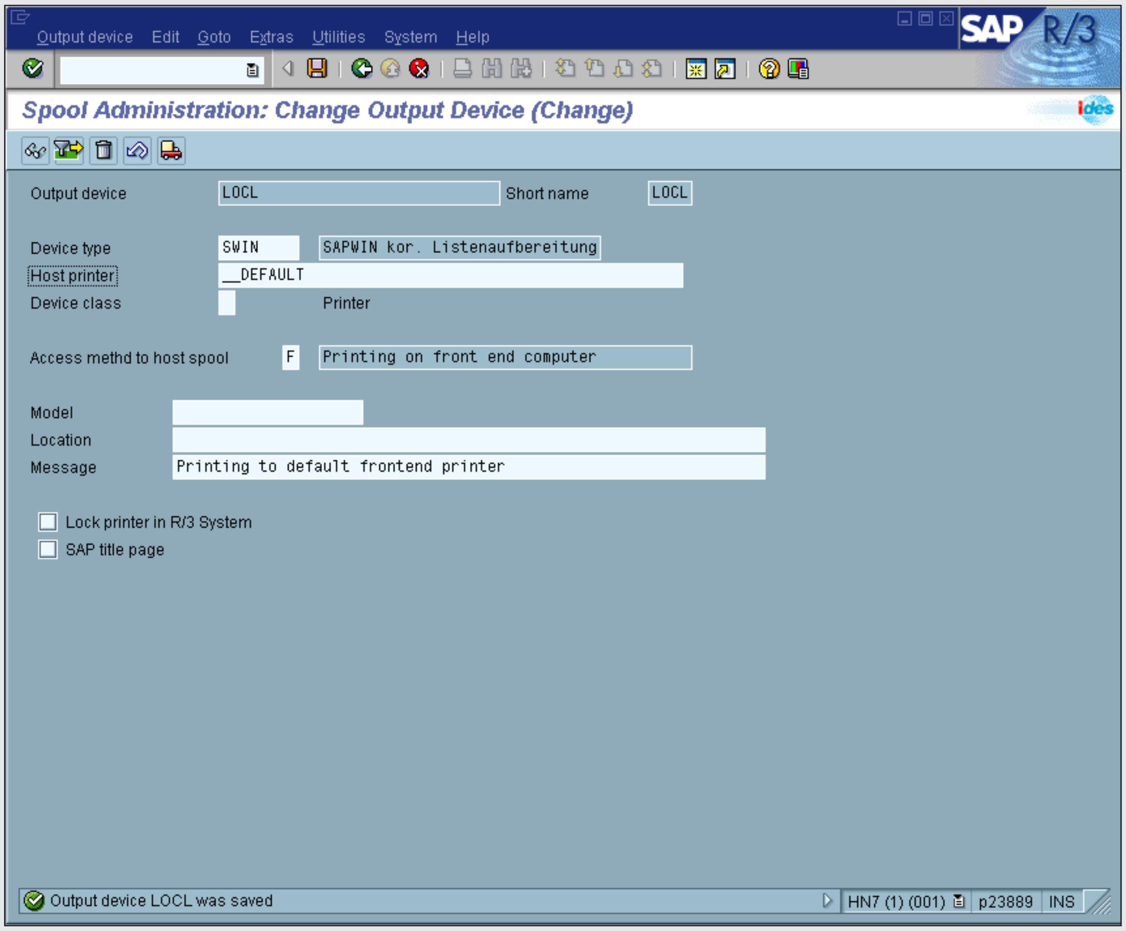
In this section, I show you how to set up a Windows 32-bit environment for frontend printing using SAPLPD and R/3 Release 4.x. There are five steps:

1. Getting the required kernel patch level.

2. Applying the required Hot Packages.

3. Defining a frontend output device using transaction SPAD.

Figure 13 Defining a Frontend Output Device in 4.0B



Spool Administration: Change Output Device (Change)

Output device	LOCL	Short name	LOCL
Device type	SWIN	SAPWIN kor. Listenaufbereitung	
Host printer	__DEFAULT		
Device class		Printer	
Access methd to host spool	F	Printing on front end computer	
Model			
Location			
Message	Printing to default frontend printer		

☐ Lock printer in R/3 System

☐ SAP title page

Output device LOCL was saved

HN7 (1) (001) p23889 INS

4. Setting the required R/3 configuration parameters.

5. Establishing the required frontend configuration.

Step 1: Getting the Required Kernel Patch Level

After deciding which kernel patch level you need, you should follow note 19466 on how to upgrade your system with that patch level.

Step 2: Applying the Required Hot Packages

Follow note 37617 on how to apply Hot Packages to your system. You can get Hot Packages either online from SAP or on a special Hot Package CD.

Step 3: Defining a Frontend Output Device Using Transaction SPAD

Figure 13 shows you a screen capture of the definition screen in a 4.0B R/3 System.[13] Here, you do the following:

1. In the **Access method to host spool** field, choose access method "F."

2. In the **Output device** field, choose a name for your frontend output device. SAP suggests the name "LOCL," but you can select any name.

[13] Transaction SPAD changed a bit from release to release. In newer releases, to simplify things and make sure that there was no room for error (e.g., misconfigurations), we did away with some fields. Were you to view this same screen in a 4.5A release, you would find that some of the fields are no longer present.

3. In the **Device type** field, choose device type SWIN.

4. The **Host printer** field *must not be left empty*. If you do not populate this field with an entry, you will get an ABAP short dump with "Assign length 0" when using this device. This entry should normally be "__DEFAULT" to access the default printer of each user. It is, of course, possible to enter any frontend printer name in that field.

5. The **No longer ask for print requests in host spool** field (which you would find on the second screen) *must not be checked*. If you check this field, all frontend print jobs will go to an error state.

The Host printer field must not be left empty. If you do not populate this field with an entry, you will get an ABAP short dump with "Assign length 0" when using this device. This entry should normally be "__DEFAULT" to access the default printer of each user. It is, of course, possible to enter any frontend printer name in that field.

Step 4: Setting the R/3 Configuration Parameters

1. Profile parameter **abap/no_sapgui_rfc** must not be set to 1.
2. Profile parameter **rspo/local_print/method** selects the kernel configuration method that is used for printing (see Figure 9). If you do not set this parameter at all, the newest available method is selected automatically.
3. If you are using an SAProuter, the connection from the frontend to the SAPgateway must be enabled by entering the following line in the routtab:

```
P * <gateway computer> sapgw<SID>
```

   <SID> refers to the SAP system number. Call **saprouter -n** to activate this change.

4. Besides the normal access rights for output devices, a user must have access rights to the output device named "%LOC" in order to do frontend printing, as follows:

   Object: S_SPO_DEV
   Field: SPODEVICE
   Value: %LOC

   This provides an easy way to disable frontend printing for certain users.

Step 5: Establishing the Frontend Configuration

1. SAPLPD (version 4.03 or newer; installing the newest version is always recommended!) must be installed in the SAPLPD subdirectory of the SAPGUI.
2. The program LPRINT.EXE (at least version 20.40) should be installed in the SAPGUI directory.
3. On each frontend, the entry

```
sapgw<Instancenumber>33<Instancenumber>/tcp
```

   is required in

```
\WINNT\system32\drivers\etc\services
```

Defining a Frontend Output Device in a 3.x System

In the preceding section, I gave you a sense of what is needed to configure a R/3 Release 4.x system for frontend printing. Here, I'll walk you through an extended configuration scenario — defining a frontend output device in a 3.x system. Rather than walk you through the entire 3.x configuration routine, my focus here will be on the *differences* between this configuration scenario and the Release 4.x configuration scenario we just reviewed:

- **Difference #1:** Choose access method "F". Note that in R/3 Releases 3.0D to 3.1I, this access method may not be available directly. You may have to run report RSPO0075 first.
- **Difference #2:** Enter an illegal value in the field "Spool server" — a value that corresponds to no active server. If you fail to do this, frontend printing may not work or you will get entries in the system log that say, "Access method 'F' not supported on this platform".
- **Difference #3:** You can leave the "Destination host" field empty, or enter any value you want.
- **Difference #4:** With R/3 Release 3.x, users must have access rights for reading the temporary spool files. (In Release 4.x, this is not required because the access functions have become a system program that can access those files.) You need to add the following to object S_DATASET:

```
Program:   SAPLLPRF
Activity:  06, 33-34
File name: *
```

Note: Older versions (3.x releases not using the recommended Hot Packages) require program "SAPLSPOO" instead of "SAPLLPRF" in the previous entry. This change will cause problems if you already have frontend printing running with the old access rights and now update it with a Hot Package.

For object S_PATH, you need to follow these steps:

1. Call transaction SM30, and select view V_SPT. Define a new access group here — e.g., "SPOF."
2. Edit table SPTH using transaction SM30 and enter the path

```
/usr/sap/<DB-Name>/<SID>/data
```
[14]

for access group "SPOF."

3. In object S_PATH, enter:

```
Activity:  03
Access group: SPOF
```

Configuring Non-Standard Systems

You've just seen how to configure Win32 frontends with SAPLPD. Let's turn our attention now to UNIX systems and the Windows Terminal Server (WTS).

Command Execution Interface

As of LPRINT Release 21.03 and SAPGUI 4.6A, it is possible to specify a print command that when executed will send frontend jobs to the local spool system on the frontend. This method then replaces the local TCP/IP protocol. It is especially useful for UNIX systems, since some LPDs refuse to accept connections from non-secure IP ports.

The environment variable **LP_CMD** defines which command is executed. Possible parameters are:

- &C copies
- &F file name
- &P printer

In an HP UNIX environment, a print command could thus be defined with:

```
setenv LP_CMD /bin/lp -d&P -n&C &F
```

[14] Path part of profile parameter **rspo/to_host/datafile**.

Note that the special printer name **__DEFAULT** is not recognized by the UNIX print commands or the LPD. This means that if you want to use this name in the output device definition, you have to create a frontend printer with that name on the UNIX frontends as well.

The same mechanism can be used when frontend printing should be implemented in a Windows Terminal Server environment. The normal configuration does not work in this case since there can be only one SAPLPD running on the WTS server at any given time. If multiple users were to request a printout, they would all get processed by that single SAPLPD. The SAPLPD, of course, is running under exactly one user account. This means all printouts that are directed to the default printer would go to the default printer of that particular user — probably not the course of action that was intended.

In order to solve this problem, you have to install SAPLPD 4.24 (required for Windows printer names with spaces) and set the environment variable, LP_CMD, to the following value for all clients on the WTS server:

```
LP_CMD=<saplpd-path>/saplpd -f&F -c&C -P&P
```

What happens now is that the command line interface of SAPLPD is executed for each frontend print request. This means SAPLPD is not using TCP/IP; the print data is transferred by a file (very much like the old 3.0A local print function). This file is written by SAPGUI or LPRINT if they determine that the variable LP_CMD is set.

TCP/IP Port

As of LPRINT Release 21.01 and SAPGUI 4.6A, the environment variable **LPRINT_PORT** can be used to switch the port number used for the internal TCP/IP connection to values that differ from the standard number 515. If SAPLPD is not running, it will start automatically with the new port. This is useful if you ever want to run a second (standard) LPD on that frontend.

As of LPRINT Release 21.03 and SAPGUI 4.6A, it is possible to specify a print command that when executed will send frontend jobs to the local spool system on the frontend. This method then replaces the local TCP/IP protocol. This is especially useful for UNIX systems since some LPDs refuse to accept connections from non-secure IP ports.

Optional Configuration Parameters for Table TSPOPTIONS

In the table TSPOPTIONS, several parameters can be altered to configure frontend printing:[15]

- **SAPLPD:** Defines the start command on Windows, which is used if no running LPD is detected. The default is:

  ```
  SAPLPD\SAPLPD.EXE -I
  ```

  which means that the SAPLPD is started in iconified mode. This requires SAPLPD version 4.03. If you use older versions of SAPLPD you should remove the **-I**.

- **LPRINT_DEST:** Defines which destination is first used for frontend printing. The default is "SAPGUI." If set to "LOCAL_PRINT," the destination "SAPGUI" will not be used, but DOWNLOAD will be tried if "LOCAL_PRINT" fails. If set to "DOWNLOAD," only DOWNLOAD will be used.

[15] In newer R/3 releases, these values can also be maintained from transaction SPAD.

- **LPRINT_NUM:** This value is the number of frontend print jobs done in the system so far. It should not be changed as it is required internally to generate unique numbers.

- **FRONT_TRACE:** Activates tracing for frontend printing. The default is 0. If set to 1, entries in the developer trace will be written. The trace must not be enabled if you do not run at least a 3.1I patch level 142 kernel, otherwise you will get an ABAP runtime exception.

- **LPRINT_EXEC** (**as of Release 4.6A):** Defines the command that is executed for the download method. The default is "LPRINT."

Troubleshooting

✔ As of Release 4.0A, you can determine which destination is actually used for frontend printing. Just go to the "Output Request," double-click on the "Status," and click the "Events" button. One of those events will describe the destination that was used for printing. The destination used will determine which of the log files (described next) is actually used.

✔ Setting the environment variable DPTRACE to 3 on the frontend (which can also be done with **SAPlogon→ SAPgui trace level 3**) results in the following frontend printing related files being written:

- ***lprint.log*** — if LOCAL_PRINT is used. If a second process runs in parallel, a file *lprint2.log* will also be written.

- ***lprintg.log*** — if SAPGUI is used as the destination.

✔ Creating a FRONT_TRACE entry in the TSPOPTIONS table with a value of 1 will activate the logging of R/3 actions concerning frontend printing in the developer trace. Here, you can see, for example, which return codes are received from the frontend.

✔ If you encounter problems with frontend printing, the first thing you should *always* do is check to see if normal printing with access methods "U" and "S" (or "L" and "C") still works (with the known limitations concerning IP addresses, routers, etc.) The system should have at least one output device defined that is not using access method "F" and that can always be used as a reference if printing problems occur. If this output device encounters the same problems as the frontend output device, it is clearly not a frontend printing problem, but rather a general printing problem.

✔ The known problem that exists today is that the Novell TCP/IP 16-bit client is not able to print using local TCP/IP. The 32-bit version of this product, however, was tested successfully. The command execution interface of LPRINT could also be a solution in this case.

The system should have at least one output device defined that is not using access method "F" and that can always be used as a reference if printing problems occur. If this output device encounters the same problems as the frontend output device, it is clearly not a frontend printing problem, but rather a general printing problem.

Conclusion

Frontend printing has become very mature as of Release 4.6A. In older releases, there are some limitations concerning ease of use for the user (e.g., no interactive selection of frontend printers) and load problems imposed on the system (e.g., processing of print requests in dialogs). Nevertheless, I think that many R/3 environments would benefit from this new printing method, even those running older releases of R/3, provided that they upgrade their systems with the necessary kernel patches and Hot Packages.

Just recently, frontend printing was extended so that it now can also be used in Windows Terminal Server environments. We think WTS technology will become very successful given the fact that it allows you to couple older frontends, which do not have enough computing power for today's GUI requirements, with powerful WTSs that do have the power to support those requirements.

Looking toward the future, we're considering improvements in the area of status feedback to allow a more accurate trace of frontend print request status. We are also examining ways to save the extra session (or at least raise the number of concurrent sessions).

Lastly, we want to get rid of the distinction between frontend printers and output devices. In the 4.6A solution, the user first enters the name of the frontend output device, then selects the name of the frontend printer from an additional dropdown box. We hope to handle this in a more transparent way in the next (4.6C) release. Note 114426 in SAPnet informs you about any new developments and how to solve problems that occur with frontend printing.

Dr. Stefan Fuchs studied Computer Science at the Technical University of Karlsruhe, Germany from 1984 to 1990. Since 1987, he has worked for the Chair for Operating Systems at the University, developing new operating system concepts. In 1996, he received his doctorate in Computer Science from the Technical University of Karlsruhe, Germany, for his thesis in the field of Real-Time Systems. Since 1995, Stefan has been working for SAP Germany, in the Basis Department, on the development of the R/3 Spool System.