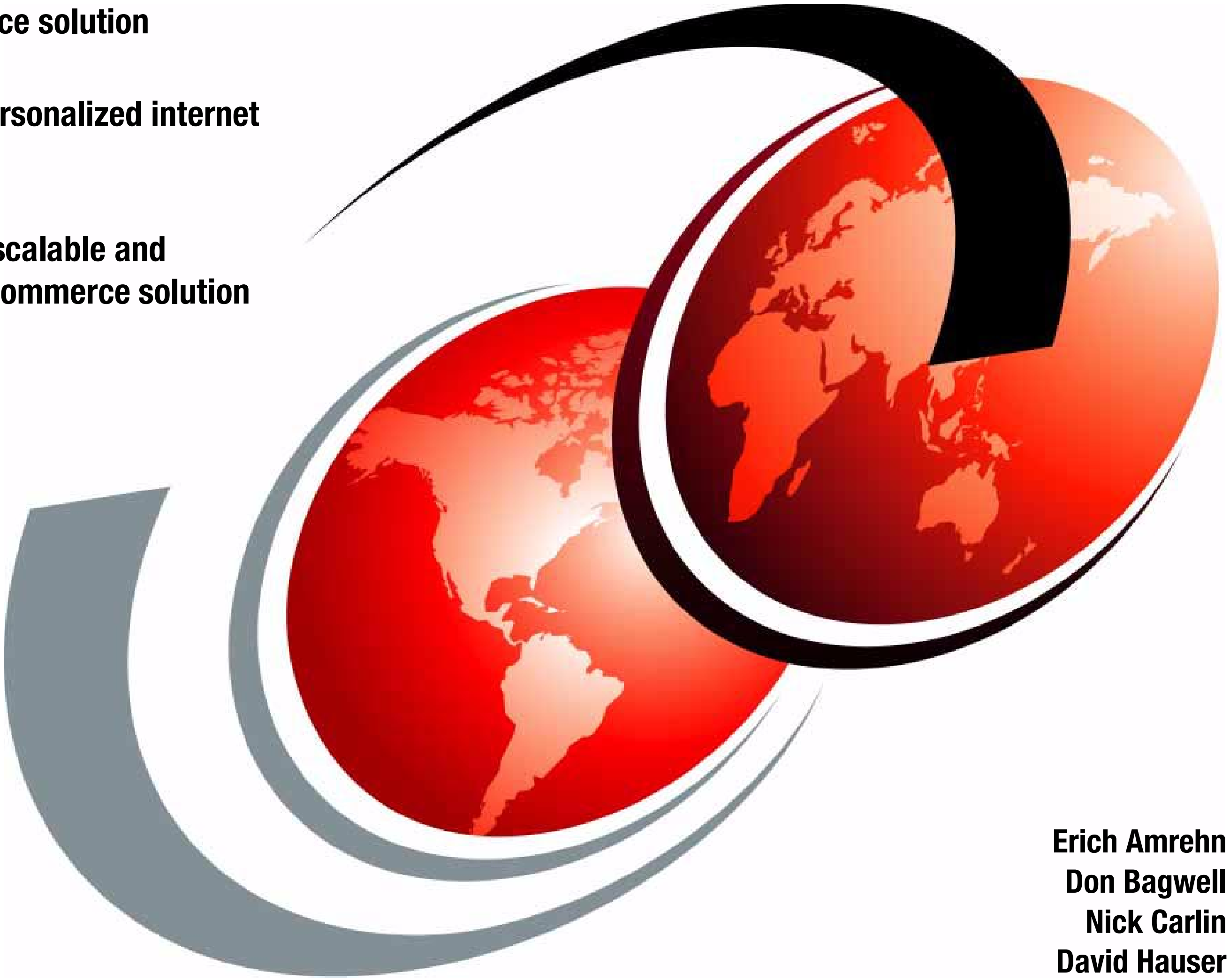
IBM WebSphere Commerce Suite V4.1 for OS/390: First Steps

Leverage the strength of OS/390 for an e-commerce solution

Provide personalized internet auctions

The most scalable and secure e-commerce solution available



Erich Amrehn
Don Bagwell
Nick Carlin
David Hauser

Redbooks

ibm.com/redbooks

International Technical Support Organization

SG24-5683-00

IBM WebSphere Commerce Suite V4.1 for OS/390: First Steps

March 2001

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special notices” on page 207.

First Edition (March 2001)

This edition applies to Version 4, Release 1 of IBM WebSphere Commerce Suite Pro Edition for OS/390 Program number 5697-G05.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

WebSphere Commerce Suite V 4.1 is the latest offering in support of e-commerce for OS/390. It is a flexible, scalable, secure application suite that makes it easy to produce functionality for net-based commerce solutions.

This IBM Redbook describes the architecture and features of WebSphere Commerce Suite, as well as presenting a review of the features carried forward from the predecessor product, Net.Commerce V3.1.2 for OS/390.

This redbook helps you plan for, install, configure, and tailor WebSphere Commerce Suite for OS/390. Detailed instructions for migrating from Net.Commerce to WebSphere Commerce Suite are provided. Step-by-step examples of creating and publishing a Web-based "store," creating a customized on-line auction, and designing personalized pages are also presented.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Poughkeepsie Center.

Erich Amrehn is a certified Senior IT Specialist at the International Technical Support Organization, Poughkeepsie Center. Before joining the ITSO, he worked as a technical consultant to the IBM System/390 division for e-commerce on S/390 in Europe, the Middle East, and Africa. He also has 13 years of VM experience in various technical positions in Germany and other areas in Europe and worldwide.

Don Bagwell is a Senior IT Specialist in the Washington Systems Center, located in Gaithersburg, Maryland. He has 16 years of experience with IBM in subject areas as varied as logistics management, software testing, networking sales and support, RS/6000 sales, and OS/390 technical support. For the past three years, he has focused on the IBM e-commerce suite running on OS/390. He focuses on technical education and technical marketing, and has worked with several customers on their Net.Commerce/390 implementation.

Nick Carlin is the EMEA Beta Program Manager for Net.Commerce/390, based in Hursley Park in the United Kingdom. He has four years of experience working with customers implementing OS/390 e-business solutions. He has worked with IBM for 14 years. His areas of expertise include Net.Commerce for OS/390 and Tivoli Systems Management for OS/390. He writes both IBM and industry publications in these subjects and is a regular Guide/Share speaker.

David Hauser is an Advisory Programmer at IBM Santa Teresa Lab. He has 15 years of experience in the IBM DB2 for S/390 Development and Performance organization. Dave is now a member of the Net.Commerce for OS/390 Development team, specializing in DB2 matters.

Thanks to the following people for their invaluable contributions to this project:

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Kevin Curley, Steve Wehr
IBM US Poughkeepsie

Heather Dunning-Johnson
IBM US Michigan

Derek Fabb
IBM UK Hursley

Stew Edelman, Al Natividad, Anthone Ciccone
IBM US San Jose

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 217 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. An overview of WebSphere Commerce Suite OS/390

WebSphere Commerce Suite Version 4.1 for OS/390 is IBM's latest offering in the e-commerce arena on the S/390 platform. That name is a mouthful, so it will frequently be abbreviated to simply WCSuite throughout this redbook.

The WCSuite product is very similar in function to the offerings of the same name on the NT, AIX and OS/400 platforms. In fact, all of the platforms work from the same set of source code. Each platform has its own set of unique features that are specifically taken advantage of by the developers for that platform, and WCSuite for OS/390 is no different.

This chapter explores the WebSphere Commerce Suite for OS/390 product, how it works, and what other OS/390 components are required to build a solution.

1.1 WebSphere Commerce Suite for OS/390: a high-level view

The field of electronic commerce (or "e-commerce" as many are fond of calling it) is a very active one today. The newspapers are full of stories of various "dot-com" companies that are doing (or planning to do) all manner of things over the Internet. Regardless of whether applied to a business-to-consumer site (B2C) or business-to-business (B2B) site, big company or small, all of the solutions share the same essential *technical* components:

- A Webserver that receives the URL requests from the browser and handles them according to the configuration of the Webserver
- A place where data is stored and retrieved; usually this is a relational database
- A means of taking information from the database and dynamically wrapping it in HTML for the user's browser to receive
- Some programming, or code, or logic — whatever you wish to call it — that controls the flow of the site and performs key tasks

If you were to draw a picture of this, it would look something like Figure 1 on page 2.

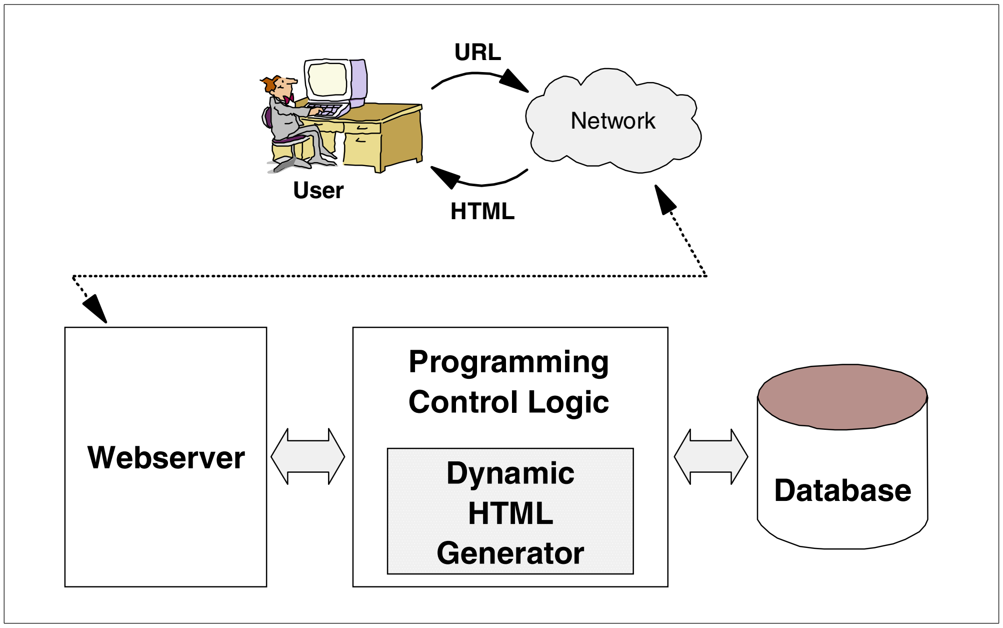


Figure 1. Basic technical components of an e-commerce solution

There is, as you might imagine, a great deal more detail behind this than is illustrated by this simple picture. But the picture is useful because it reduces what are very complicated solutions down to some relatively simple concepts that apply to all e-commerce sites. The complexity comes into play when one explores how each piece of this picture is actually implemented. We'll start that process in this chapter.

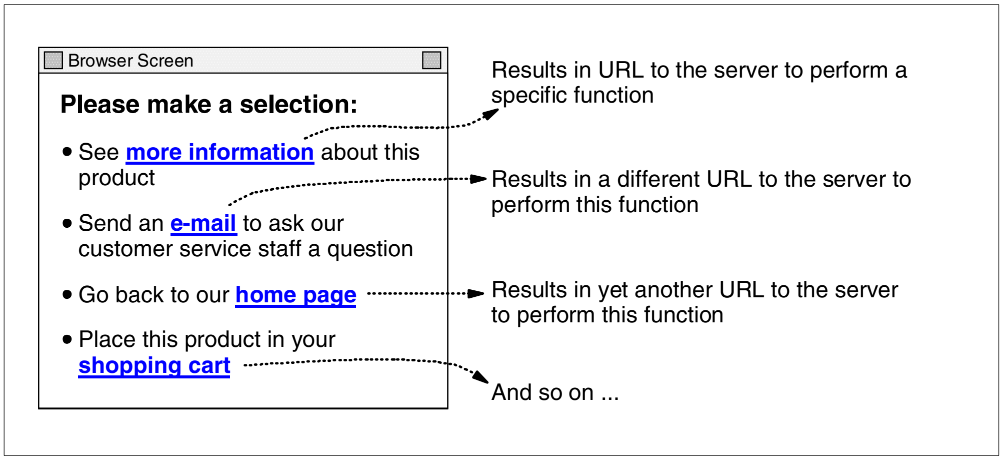
1.1.1 The user's "window" into the site

Any e-commerce solutions would be of little value if people didn't use them, and the way in which people use them is to access the site from some kind of device, typically a computer with a browser.

Note: To show you how rapidly things are changing, even the previous statement is coming close to being only sometimes true. With the popularity of handheld devices like cellular phones and PDA's (personal digital assistants, like the Palm product) growing every day, people are increasingly connecting to these e-commerce sites with something other than a PC and browser. Even the person in this mix is giving way to computer programs talking to computer programs. To keep things simple we'll focus on a person at a browser.

The browser represents the user's "window" into the e-commerce site; it gives the user a means of seeing the information sent by the server, and it gives the user a way of indicating — through mouse clicks on links or buttons — what action the user wishes to take.

Now this is a key point, because it's through the user's mouse click that all of the other activity we're about to discuss is initiated.



Please make a selection:

- See **more information** about this product
- Send an **e-mail** to ask our customer service staff a question
- Go back to our **home page**
- Place this product in your **shopping cart**

Results in URL to the server to perform a specific function

Results in a different URL to the server to perform this function

Results in yet another URL to the server to perform this function

And so on ...

Figure 2. How the user's mouse clicks determine the next action the server will take

These URLs that we mention will be interpreted by the Webserver and handled according to the configuration of the Webserver. That's discussed next.

1.1.2 The Webserver's role: receiving URLs and issuing HTML

The URL comes from the user's browser, and based on the "host name" in the URL finds its way to the server running the e-commerce application. The Webserver will receive this URL and *map* it against the Webserver's configuration information to see how it should handle the request. There are two basic forms of requests to consider at this point in the discussion:

1. Requests for *static* information like HTML files, JPG or GIF image files, or possibly WAV sound files or MPG movie files. The term static is used because the information resides in files on the server and is simply retrieved and sent to the browser. WebSphere Commerce Suite for OS/390 does not get involved in these requests. The Webserver will handle these all by itself.
2. Requests for a *service*, which in this discussion means the URL contains a WebSphere Commerce Suite *command*, and the Webserver's configuration file has information about how to pass that request to the WebSphere Commerce Suite server code running on the OS/390 system.

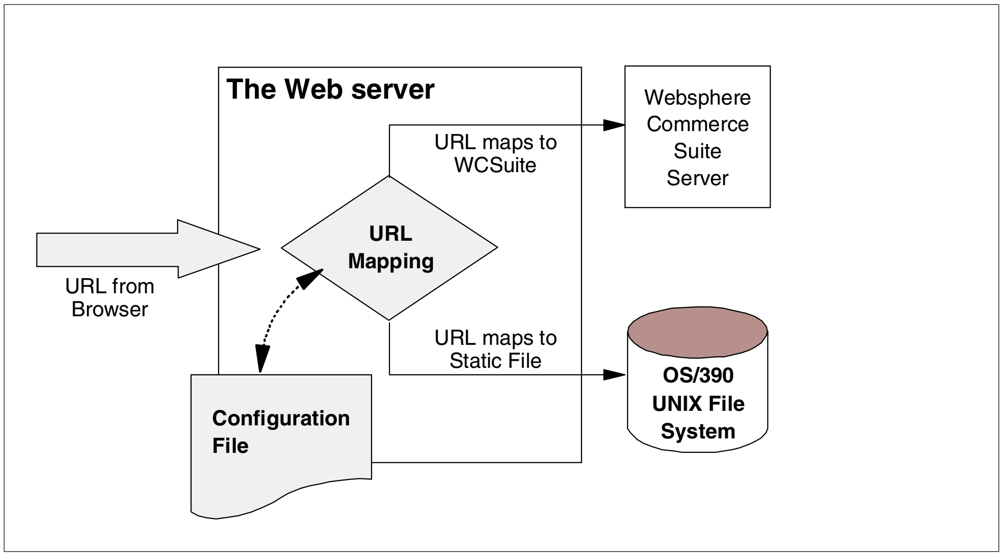


Figure 3. The Webserver's handling of URLs from the user's browser

We're starting to get just a bit ahead of ourselves here, so let's forget the discussion of WCSuite commands and how they're handled until later. For the moment, just assume that the command results in some HTML being generated by the WCSuite server. That HTML is then handed back to the Webserver, which passes it down to the user's browser.

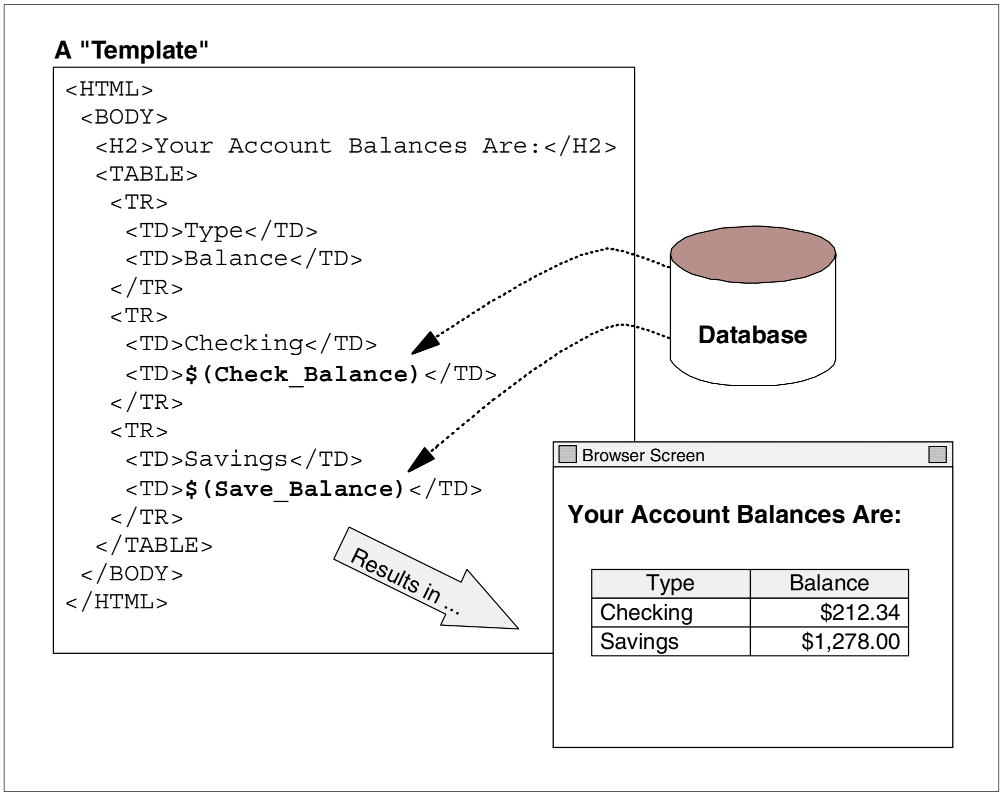
The cycle is complete: user clicks a link or button, which results in a request going to the server, which results in some activity that results in new HTML being delivered to the browser. The user now has a new set of links or buttons from which to choose. Clicking on one of them repeats the cycle.

1.1.3 Generating dynamic HTML

As we described in the previous section, it is possible to request from the Webserver a static HTML page. The Webserver will happily serve that up quite rapidly. But designing a Web site based on just static HTML pages — while possible — would be extremely difficult to manage for more than a handful of different pages. Imagine if you were the site administrator for a site with 20,000 HTML pages, and you were told to change something that appears on 15,000 of those pages.

We hope the need to avoid having to manage static pages is obvious to you. The question is this: How is that accomplished?

The answer to this question is best illustrated with a picture, which is provided in Figure 4 on page 5.



```
<HTML>
 <BODY>
  <H2>Your Account Balances Are:</H2>
  <TABLE>
   <TR>
    <TD>Type</TD>
    <TD>Balance</TD>
   </TR>
   <TR>
    <TD>Checking</TD>
    <TD>$(Check_Balance)</TD>
   </TR>
   <TR>
    <TD>Savings</TD>
    <TD>$(Save_Balance)</TD>
   </TR>
  </TABLE>
 </BODY>
</HTML>
```

Your Account Balances Are:

Type	Balance
Checking	$212.34
Savings	$1,278.00

Figure 4. Using an HTML “template” and populating fields with information drawn from a database

The idea of a “template” is that information in an HTML file that will change — or be “dynamic” — is represented by a variable that “holds the place” of the dynamic information within a framework of HTML. In this example, the dymamic information is the customer’s checking account balance and savings account balance. The title that says “Your Account Balances Are,” by contrast, is static: every customer accessing the site will get that, so there’s no worry about it being “hardcoded” into the template.

The dynamic content is drawn from the database and inserted into the template where the variables for that data reside. So this one template could be used for hundreds or thousands or even millions of different customers.

In WebSphere Commerce Suite V4.1 for OS/390, two different ways of producing dynamic HTML are provided:

- **Net.Data**, which is an IBM product that produces dynamic HTML from a number of data sources (SQL queries against DB2 being the most important for WCSuite), is supplied with the WCSuite product
- **Java Server Pages**, or **JSP** for short, is an industry standard way of using Java technology to retrieve and populate an HTML file with dynamic information.

These two methods are conceptually identical, but the underlying technology and method by which they are invoked are quite different. These differences will be illustrated throughout this redbook.

1.1.4 Where the data is stored: the database

Dynamically generating HTML is fine, but would not be of much value if you didn't have a "data store" from which to draw the dynamic content. As we mentioned earlier, that data store can be of almost any type (flat file, another computer program), but the most common data store is a relational database.

For WebSphere Commerce Suite Version 4.1 for OS/390, that relational database is DB2 Version 6.1 or higher.

The DB2 database is where all the information about the products being sold through WebSphere Commerce Suite is housed: descriptions, prices, and such. However, that's not all that's stored in the DB2 database. With WCSuite for OS/390 (and NT, AIX and OS/400 as well), the database holds information about the shoppers, the products for which the shoppers have indicated an interest (the "shopping cart"), those products the shoppers have chosen to purchase, as well as a great deal of information used to control the flow and operation of the WCSuite site.

Net.Data or JSP pages access this database to get the information to dynamically create HTML pages. The WCSuite program itself accesses the database to get information about how to operate and behave. Even the Webserver accesses the database to retrieve information.

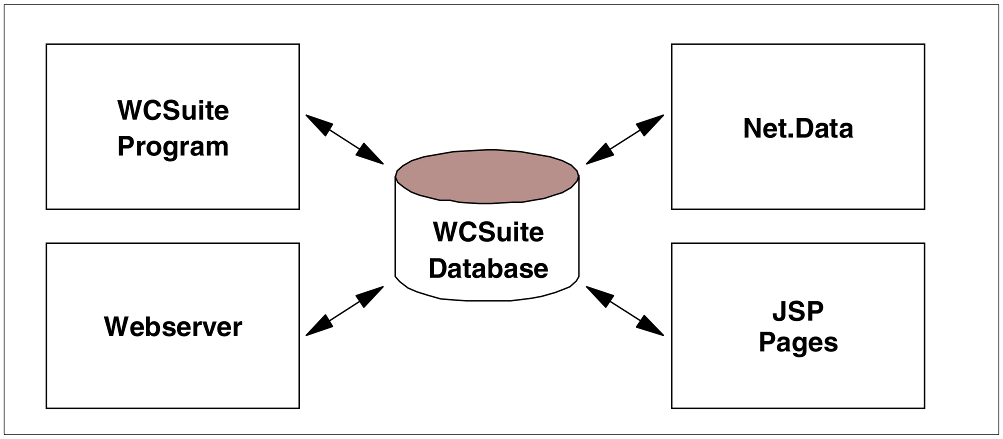


Figure 5. The WebSphere Commerce Suite database is at the heart of the solution

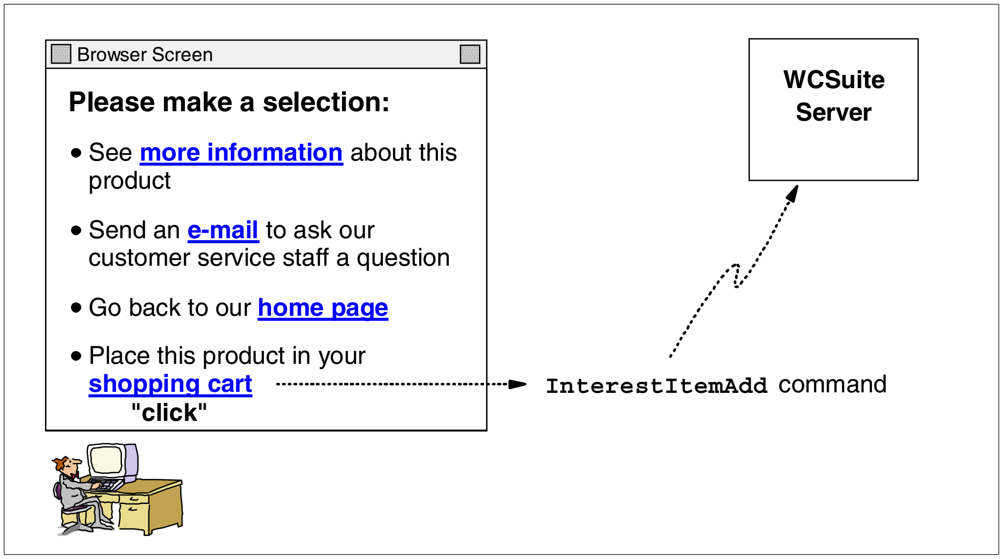
The database is the heart of WebSphere Commerce Suite for OS/390. WCSuite is at its core a database application. Products offered by other companies and against which WCSuite competes are also database applications at their core.

1.1.5 Controlling it all: the logic provided by the WCSuite product itself

What ties this all together is what we referred to back in 1.1, "WebSphere Commerce Suite for OS/390: a high-level view" on page 1 as "programming, or code, or logic." This is the set of internal functions and operations that determine how WCSuite behaves. Without this, a URL handed over to WCSuite from the Webserver would be met with a dull look from WCSuite.

This logic is provided in the form of a set of *commands* that can be issued to the WCSuite product. So, for example, the command `InterestItemAdd`, when sent to the WCSuite server, tells the server to add the product referenced on the command to the shopper's shopping cart. The WCSuite product is coded in such a way that it recognizes what `InterestItemAdd` means, and then carries out the function defined for that command.

WebSphere Commerce Suite Version 4.1 for OS/390 comes with several dozen such commands. You, as the designer of the site, determine what function you wish to implement behind a link or a button on the user's browser screen, and based on that intended function you make sure the appropriate WCSuite command is sent to the server when the user clicks that button.



Please make a selection:

- See **more information** about this product
- Send an **e-mail** to ask our customer service staff a question
- Go back to our **home page**
- Place this product in your **shopping cart**

Figure 6. Commands tell WebSphere Commerce Suite what to do

One of the things a command may do (and indeed, many commands do this) is invoke Net.Data to generate a dynamic HTML page. So perhaps you can start to see a pattern forming in this discussion:

- The user has a screen on their browser.
- They click on a link or a button, which results in a URL being sent to the server.
- The Webserver takes a look at the URL and determines if it is a WCSuite command, and if so then it passes it to WCSuite.
- WCSuite receives the command and then, based on its internal programming, executes the behavior of that command.
- In many cases the command results in the invocation of Net.Data, which generates a dynamic HTML page based on information drawn from the database.
- The HTML is sent to the user's brower, which results in a new screen being rendered, with a new set of buttons and links.
- The user clicks on a button or link ... and the cycle starts again.

And that ties the picture together. That is, in essence, what WebSphere Commerce Suite is all about. As stated earlier, there is a great deal more detail to this, and only through practical experience can you get to be an expert at all the aspects of implementation. But this high-level review sets the foundation upon which you may rely as you read the rest of this book.

1.1.6 A note on how WCSuite solutions are implemented

WebSphere Commerce Suite is designed to be an *extensible framework*, not a ready-made application. The reason for this is quite simple, and rooted firmly in the reality of the world: customers want things customized to meet their particular requirements.

The typical implementation of WebSphere Commerce Suite involves several phases:

- **Customer requirements gathering** (trying to figure out what the customer has in mind for their solution so the development effort can be as efficient as possible)
- **Site planning** (mapping out a screen flow, determining what functions are invoked at each point of the flow, and figuring out what functions of WCSuite need to be customized to provide what the customer is looking for)
- **Development, customization and testing** (constructing the HTML pages, the Net.Data macros and the custom functions, testing each separately and as a system)
- **Implementation, rollout, monitoring, and maintenance** (putting the solution into production, making sure it's all working properly, monitoring the traffic and activity of the site, and modifying/fixing any pieces of the solution that require it)

All of this suggests that a WebSphere Commerce Suite project is more than a simple thing. The same holds true for solutions offered by competitors. The very large and famous e-commerce sites you read about often didn't happen overnight. They took many months and millions of dollars to construct. But the rewards are potentially enormous, and therefore the effort can be justified.

Keep it all in perspective: your worst enemy is unmet expectations.

1.2 WebSphere Commerce Suite for OS/390: a more detailed view

The high-level overview presented in the previous section was intended to give you a basic view of how the product works. In this section we dig a bit deeper, and we position you for the information provided in the rest of the book.

1.2.1 Function that was provided in Net.Commerce V3.1.2

Net.Commerce for OS/390 Version 3.1.2 was the name of the product prior to it being named "WebSphere Commerce Suite." What follows is a list of some of the major functions of Net.Commerce Version 3.1.2. This list will give you a pretty good feel for what the product could do.

Note!

All of the function provided in Net.Commerce Version 3.1.2 for OS/390 has been preserved in WebSphere Commerce Suite V4.1 for OS/390. New function has been added, and that will be discussed in 1.2.2, “Significant new function added with WCSuite V4.1 OS/390” on page 13.

1.2.1.1 Session awareness and persistence of user information

The HTTP protocol used by the Web is what’s known as a “stateless protocol,” which means the server isn’t maintaining a “session” for each user. Each click of the user’s mouse that results in a request being sent to the server is treated as a new connection. That could be a problem in a commerce solution because it’s important to know who is requesting the information.

To address this, Net.Commerce (and WCSuite as well) maintains awareness of each user by setting a memory-resident “cookie” on the user’s browser with the user’s session ID. That cookie *does not* contain any address, phone number or credit card information! (That information is kept in the database.) The cookie is used to identify the user with each incoming request on the “stateless” HTTP protocol.

The user may, if they choose, “register” with the Net.Commerce site. Doing so creates a record in the database. Later, a user may “logon” to Net.Commerce, which results in the setting of a cookie with the user’s permanent userid. This establishes identity and allows Net.Commerce to associate a user with his or her past activities, such as shopping cart entries, orders, and the like.

1.2.1.2 Authentication of registered users

When a registered user returns to the site, they must provide their userid and password before they are permitted to enter the site as that userid. The userid and password provided by the user are compared against the userid/password provided at the time of registration. If they match, the user is allowed in. If not, they are rejected.

Note: Users who don’t register are still allowed to browse the site and make purchases. The difference is when they leave and return later, Net.Commerce sees them as a different user. Information in the database related to their previous visit is tied to a random-looking userid for that visit; for the current visit they have a new random-looking userid.

1.2.1.3 SSL encryption of sensitive transmissions

As part of its control logic, Net.Commerce had a means of being told which requests (or commands) were sensitive in nature and therefore in need of SSL encryption. When Net.Commerce saw that request, it would invoke SSL. For OS/390, the hardware crypto feature could be exploited for running SSL, resulting in substantial performance improvement over software-based SSL.

1.2.1.4 Dynamic HTML generation

For Net.Commerce V3.1.2, this capability was provided by Net.Data, which was bundled with the Net.Commerce product. Net.Data used files known as “macros” (or “templates”, in the language used in 1.1.3, “Generating dynamic HTML” on page 4) that described a framework of HTML into which dynamic data was inserted. SQL queries against the Net.Commerce DB2 database provided the dynamic data.

Net.Data macros were used to display such dynamic content as product pages, category pages, the contents of a user’s shopping cart and the order page. Net.Data was a very powerful tool, and was used extensively by Net.Commerce.

1.2.1.5 Caching of Net.Data pages

It is often the case that the content of dynamic pages don’t change from user to user that access the page. A good example is a product page, which contains information about a product like its price, its description, and an image of the product. You still want the page to be dynamic so you don’t have to manage static HTML. The issue is that invoking Net.Data involves some processing “cost,” and for cases where the content doesn’t change frequently you may wish to bypass the running of Net.Data.

That’s where the caching function came into play. It provided a way of taking the HTML generated from the first invocation of Net.Data and stashing it into DB2. Then, the next time that page was accessed, the HTML was fetched from DB2 directly rather than invoking Net.Data. This could have a significant impact on performance and was a very useful feature.

1.2.1.6 Customizable database schema with referential integrity

Net.Commerce came with a fully defined database schema with a full set of referential integrity definitions. The default database had over 100 tables defined to it. The architecture of the product allowed you to extend the database by adding additional tables if you wished. Furthermore, many of the tables had columns set aside for your use.

1.2.1.7 Scalable product catalog design

The design of the database allowed any number of categories and products. In addition, the relationship between products and categories was defined in a way that permits a product to belong to multiple categories (any number, in fact), and sub-categories to belong to multiple parent categories (any number, as well).

1.2.1.8 Tools to facilitate intelligent catalog displays to customers

This was a combination of the database design for Net.Commerce and a function provided known as the “Product Advisor.” The database design permitted the definition of any number of attributes associated with a product. An attribute took the form of a “name/value pair” such as “color=red” and “size=large.” By describing products in that manner rather than in long text description fields, it permitted intelligent search mechanisms to narrow the product choices for a customer, thus helping them make their purchase decision more quickly and easily.

The Product Advisor was a tool that presented the customer an opportunity to specify the attributes they were interested in and have the number of products that met that criteria selected from the database. For example, if you were interested in only homes with 4 bedrooms or more, you could tell Product Advisor that and it would eliminate from consideration those homes in the catalog with 3 or fewer bedrooms.

1.2.1.9 Customizable command set

In 1.1.5, “Controlling it all: the logic provided by the WCSuite product itself” on page 6 we introduced the concept of the “command.” Net.Commerce came with several dozen commands that could be used to control the flow and function of the site.

These commands were designed to be modular and customizable. The conceptual architecture of a command is as shown in Figure 7 on page 11.

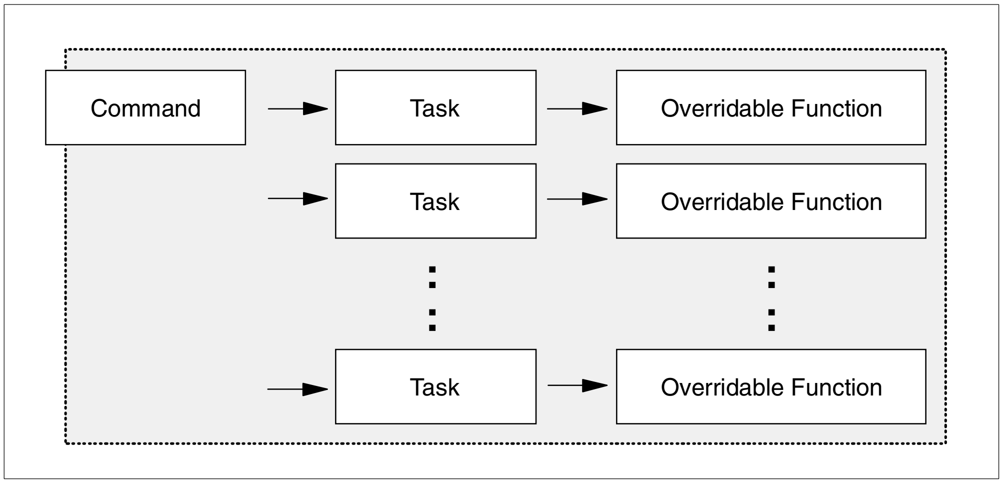


Figure 7. Conceptual architecture of a Net.Commerce command

The “command” is really a container of some number of smaller units of work. Those smaller units of work are known as “tasks,” and for each task there is an associated “overridable function” (OF) that implements, through programming, the intended behavior of the task. So a command’s mission is to call tasks in the the sequence necessary to perform the objective of the command.

Tasks are nothing more than symbolic names that are associated with OFs. Because of this, the command (which is a compiled C++ executable) doesn’t have to have the name of the OF imbedded in its coding. This allows you to swap out the OFs with your own custom OFs without having to recompile the command.

This flexible structure lets you:

- Code your own overridable functions (OFs) if you don’t like the default behavior of an OF and wish to implement your own
- Code your own commands to do significant new functions

It is through customizing these commands that you make connections to the other legacy systems.

1.2.1.10 Ability to extend processing to backend fulfillment systems

Many customers (particularly OS/390 customers) have fulfillment systems already in place. The design objective of Net.Commerce was to permit you to extend your e-commerce solution and tie into those systems. For example, you may have a CICS system that maintains inventory and shipping records. Rather than duplicating that information in the Net.Commerce database, the design of the product allowed you to exit the command architecture and access the backend system (see 1.2.1.9, “Customizable command set” on page 11).

1.2.1.11 Ability to extend processing to other tax calculation systems

Tax calculations on Internet sales is an area rich with complexity and change. The tax calculation algorithm that came with Net.Commere was intentionally simple and generic. Just like for backend fulfillment systems, you could exit the command structure at the point (the “task”) where taxes were calculated and access a tax system designed for the specific circumstances of the site.

1.2.1.12 Ability to extend processing to payment authorization systems

Like backend fulfillment and tax calculations, payment authorization was something you could accomplish on an existing system or service by exiting the command structure of Net.Commerce. This was done by providing an overridable function at the appropriate point that made a call to the payment system.

1.2.1.13 Support for the SET payment protocol

Net.Commerce was designed to accept SET payments by making use of the Payment Manager product from IBM. SET (Secure Electronic Transactions) is a payment authorization protocol defined by a consortium of companies such as Visa and Mastercard and is intended to offer very secure payment over the Internet.

1.2.1.14 Utility to load bulk data into the database

The “Mass Import” utility of Net.Commerce provided a way to bulk-load data from a delimited file into the DB2 database. Small amounts of data could be input into the database using the Administrative Utility (commonly known as “NCADMIN”), but that method grew tiresome for more than a few dozen products. The Mass Import utility facilitated much larger loading of the database.

Of course, the DB2 Load utility could also be employed. But that would require you to understand the schema of the database far more deeply than Mass Import required. Mass Import masked much of the complexity of the database from you, making bulk loading easier for those not interested in knowing the database structure completely.

1.2.1.15 Utility to assist in cleaning and maintaining the database

Periodic cleaning of the database is a necessity, and the NCCLEAN utility provided that function. Any other DB2 method of deleting records from a table could be employed, but the NCCLEAN utility provided a way of doing this without requiring extensive SQL knowledge or knowledge of the structure of the Net.Commerce database.

1.2.1.16 Utility to monitor the peformance of the system

The “Peformance Monitor” function kept track of all the commands received by the Net.Commerce server and recorded statistics on the time it took to execute the commands. With that information recorded, you could then query the information to see if some commands were taking longer to execute than expected. This provided you a good way to focus on performance bottlenecks.

1.2.1.17 Tracing and debugging tools

Finally, as is the case with any system, sometimes things just don’t work properly. To assist in the debugging of those situations, a tracing facility was provided with

Net.Commerce to allow you to investigate the causes of problems. This tracing facility had several levels of detail available to it, from just reporting errors to reporting a full tracing of all the activities of the site.

1.2.2 Significant new function added with WCSuite V4.1 OS/390

In the previous section we described the major functions provided with the predecessor product to WebSphere Commerce Suite V4.1 for OS/390. And as we stated before, *all of that functionality has been brought forward into the WCSuite product* as well, along with some new functions.

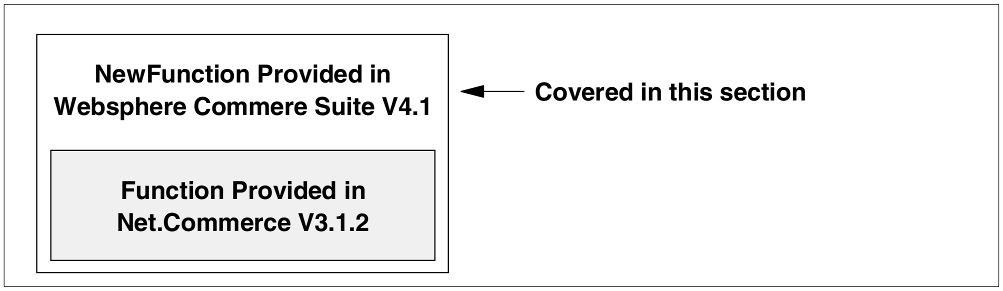


Figure 8. New functions in WCSuite V4.1 relative to functions provided in Net.Commerce V3.1.2

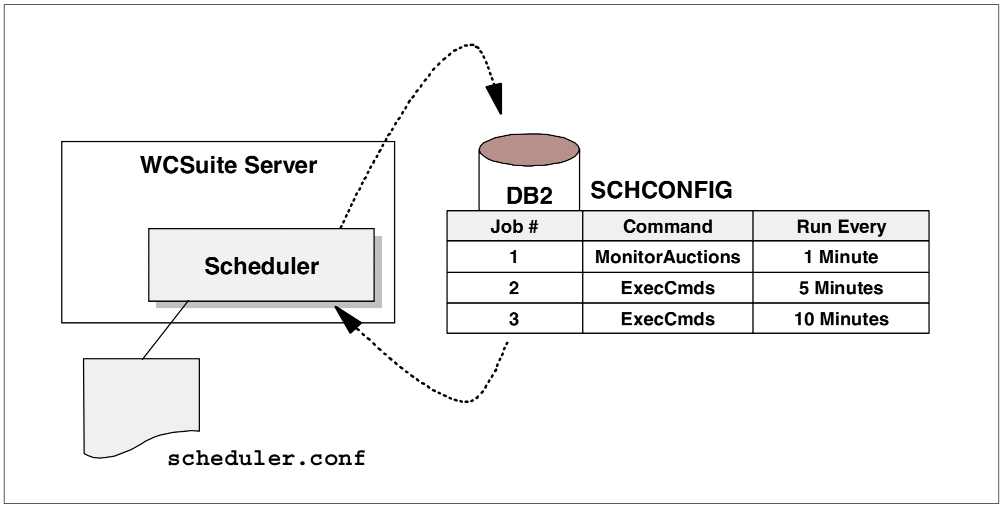
In this section we describe the new functions offered in WCSuite V4.1 for OS/390.

Note: This will not be a complete description of each function. That would be outside the scope of this redbook. See 1.4, “Our assumption of your understanding of WCSuite functions” on page 26 for a picture of what we’re assuming you’ve learned from other sources.

1.2.2.1 Job scheduler

This function of WCSuite V4.1 provides a way to automate functions and execute them on a periodic basis. The most obvious application of something like this is providing a way for shoppers to create recurring, or scheduled orders.

To WCSuite, a “job” is a command, and these jobs are recorded in a database table (the SCHCONFIG) with information about how frequently they are to be run. The Scheduler, which is a separate process that runs in the WCSuite address space, is tapped awake periodically and looks through the table for jobs (commands) that are to be run.



Job #	Command	Run Every
1	MonitorAuctions	1 Minute
2	ExecCmds	5 Minutes
3	ExecCmds	10 Minutes

Figure 9. The WebSphere Commerce Suite job scheduler

When the Scheduler finds a job to be run, it runs the job (command) under the authority of the WCSuite user named in the database. The functions performed by the command are whatever that command is designed to do. Any WCSuite command can be named as a job, even ones that run Net.Data macros (the HTML results from the running of the macro are simply discarded).

This function is critical to the operation of the auction support new with WCSuite V4.1 for OS/390. It is also critical to the operation of the scheduled orders support.

1.2.2.2 Auctions

The auction support provided new with WCSuite V4.1 is based on the premise of a merchant auctioning off products defined in their PRODUCT database table. That's a different approach than the one where users offer up their stuff for auction, such as is the case with e-Bay.

Three types of auctions are supported:

1. **Open cry**: This is what most people think of when they think of auctions. Each person participating in the auction sees the current best bid, and they have the opportunity to outbid the best if they're seriously interested in the product. At some point the auction closes (either because the expiration time of the auction has passed, or some period of time since the last bid has transpired) and the person with the best bid at that time wins.
2. **Sealed bid**: In this form of auction the participants can't see what the current best bid is, so they don't know whether they need to bid again. These auctions usually go just one round.
3. **Dutch**: This one is a bit of a twist: the administrator of the auction starts off and sets a price. Then if the number of people who step forward and indicate they're willing to purchase at that price doesn't satisfy the administrator, they *lower* (yes, that's right) the price to see if more people will step forward. Eventually, after several rounds of price-lowering, enough people have stepped forward and the administrator closes the auction. This form of auction is typically used to clear inventory where the quantity being sold is greater than 1.

There are plenty of intriguing twists to this function. (Such as the case where you're auctioning off two items and you get two winning bids of a different price. Do you offer the lowest price to both winners or offer each their bid price? WCSuite allows either scenario.) We won't go deeper into all of those, because our focus in this redbook is the OS/390 implications of the new product.

The new auction support is implemented with a combination of new commands, new Net.Data macros, and several new database tables. There's nothing that mysterious about the auction support once you understand the notion of commands and macros. The twist to this is the use of the WCSuite Job Scheduler to automatically monitor the auctions, start and stop them as needed, determine winners for closed auctions, and provide automated "proxy" bidding for users.

The WCSuite Administrator function providers panel interfaces to define auctions and to see their progress. A set of sample Net.Data macros are provided to assist you in developing the auction interface your users will see. In Chapter 6, "Auction and personalization functions" on page 161, we provide an example of creating an auction.

1.2.2.3 Support for WebSphere Commerce Studio

The WebSphere Commerce Studio is a suite of products that runs on Windows NT and provides a development environment for many of the new functions of WCSuite V4.1. Figure 10 illustrates the software products that make up the Studio environment.

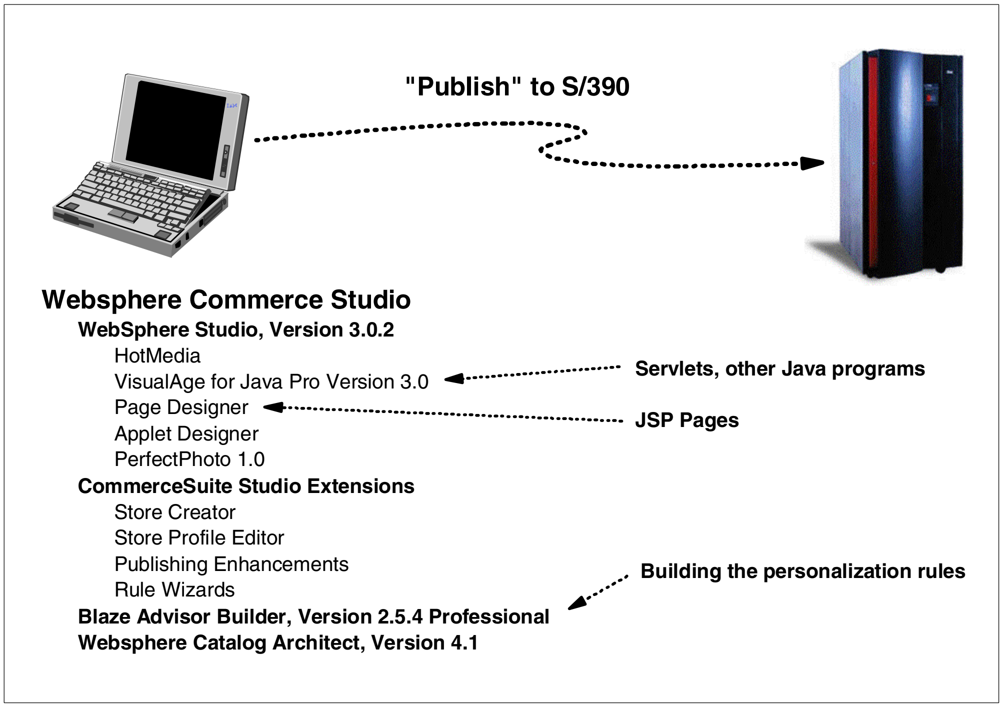

"Publish" to S/390
Websphere Commerce Studio
WebSphere Studio, Version 3.0.2
HotMedia
VisualAge for Java Pro Version 3.0
Page Designer
Applet Designer
PerfectPhoto 1.0
CommerceSuite Studio Extensions
Store Creator
Store Profile Editor
Publishing Enhancements
Rule Wizards
Blaze Advisor Builder, Version 2.5.4 Professional
Websphere Catalog Architect, Version 4.1
Servlets, other Java programs
JSP Pages
Building the personalization rules

Figure 10. The WebSphere Commerce Studio and its relationship with WCSuite for OS/390

The use of WebSphere Commerce Studio indicates a good sign in the general strategic direction of the WCSuite product line. By consolidating the *development* environment to a single platform and focusing resources to develop the best-of-breed tools, it allows the WCSuite developers of the various platforms to focus just on the "catching" of the assets (files, database updates, etc.) created on the development environment. Therefore, we don’t have to recreate the development environment on several different platforms.

The “publishing” of assets to the OS/390 platform is more than just a simple FTP process. Therefore, a special Java servlet is provided with WCSuite V4.1 for OS/390 that has the built-in intelligence to know where to place the assets “caught” from the Studio platform on NT. Figure 11 on page 16 illustrates this concept.

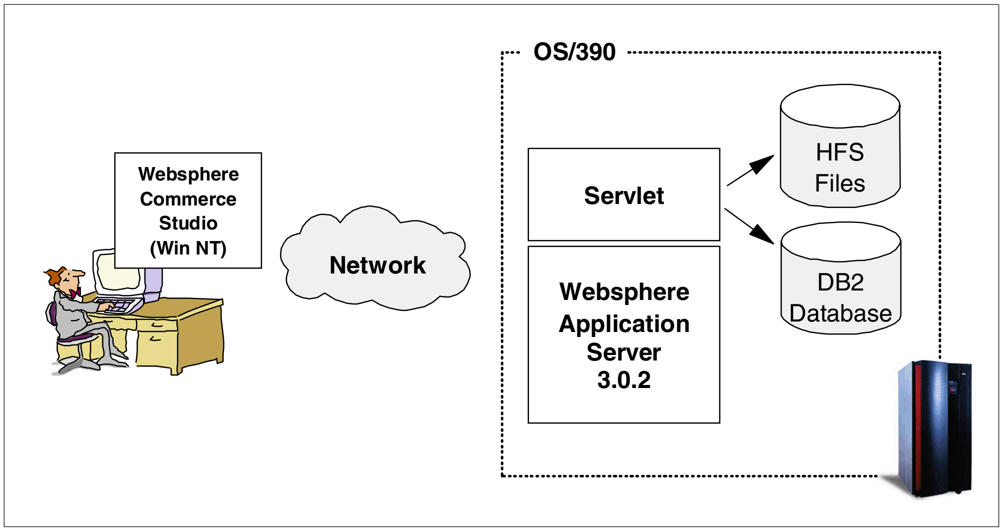

OS/390
Websphere Commerce Studio (Win NT)
Network
Servlet
Websphere Application Server 3.0.2
HFS Files
DB2 Database

Figure 11. “Catching” Studio-produced assets on the OS/390 platform

The WebSphere Commerce Studio environment plays a key role in supporting the new Personalization and JSP Page support provided with WCSuite V4.1 for OS/390.

1.2.2.4 Personalization

Personalization is the dynamic modifcation of pages displayed to a user based on some knowledge you possess about that user. In WebSphere Commerce Suite V4.1 for OS/390, the notion of personalization revolves around the displaying of product recommendations based on the user viewing the page.

The key to this feature is the addition of a “rules engine” that evaluates “rules” you define to determine what products to recommend to a user. IBM licensed this rules engine from Blaze Software. A “rule” might be something like:

```
If customer’s age is > 40
   and marital status = single
   then recommend Product XYZ
```

Any number of these rules can be grouped to form “rule sets,” and when a user accesses one of the pages on our site on which you wish to offer a product recommendation, the WCSuite server passes a request to the rules engine and tells the rules engine which rule set to evalutate.

Graphically, Figure 12 on page 17 illustrates the concept of personalization in the WCSuite environment.

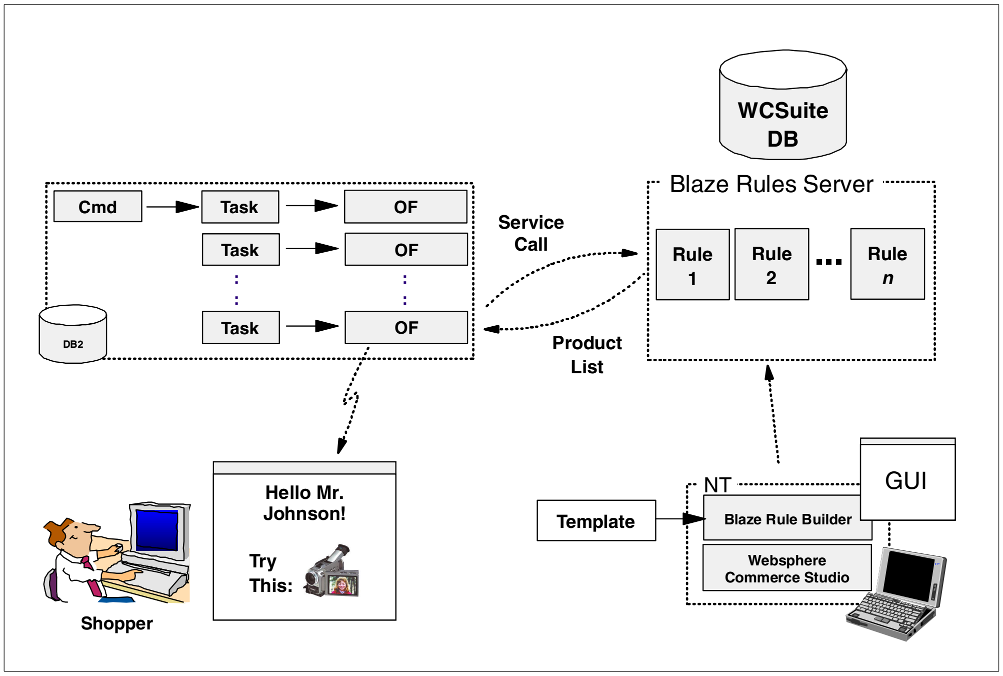


Figure 12. Personalization in a WebSphere Commerce Suite setting

Let's cover the major sections of the picture in Figure 12 separately:

- The rules and rule sets are developed using the WebSphere Commerce Studio suite of products on a Windows NT machine. The Commerce Studio suite comes with the "Blaze Rules Builder" software, which is a graphical environment specifically designed for the creation of Blaze rules. The Rules Builder has been "pre-programmed" with WebSphere Commerce Suite information and sample rules templates to assist you in getting started.
- These rules are then "published" (FTP) to the S/390 environment and placed in an HFS directory accessible to the Blaze Rules Server.
- You (as the site designer) must decide at what point in the shopping flow you wish to have the product recommendations that come back from the Blaze Rules Server displayed. The display must take place on a page generated by Net.Data. Once the point in the flow is identified, then the command used that resulted in the page must be "hooked" with a new overridable function supplied with WCSuite so the call to the Blaze Server is made.
- The call to the Blaze Server passes the userid and the rule set to be used over to the Blaze Server. The Blaze Server evaluates the rules in the rule set using information it gets from the database. This means the Blaze Server is accessing the WCSuite database directly.
- The Blaze Server generates some number of product recommendations (from zero up to the maximum number to be returned according to a configuration parameter) and passes those back to the OF. This string of product recommendations takes this form:

```
ProdRec1=<product reference number>
ProdRec2=<product reference number>
    :
    :
ProdRecn=<product reference number>
```

Where `<product reference number>` is the value found in the `PRRFNBR` column of the `PRODUCT` database table.

- The OF pases that string into the Net.Data macro that will be used to render the page on which the recommendation will appear. *That Net.Data macro must be coded to accept the recommendations and do something with them.* The list of recommended products are coming into the macro as parameter name/value pairs, so traditional Net.Data programming techniques can be used to parse out, query for, and display the products.
- If everything works properly, the user will see on their screen the recommendation made by the Blaze Server.

Personalization is a feature of WCSuite that requires a good deal of thinking ahead of time about what kinds of rules you wish to establish and where you want the resulting recommendations displayed.

1.2.2.5 Java/JSP support

As we stated earlier in this chapter, one of the basic building blocks of any e-commerce site is the ability to generate dynamic HTML based on information drawn from a data store. Up to this point we have pointed to Net.Data as the tool used to do the dynamic HTML rendering. With WebSphere Commerce Suite V4.1 for OS/390, a new method of rendering dynamic HTML is provided: Java Server Pages (JSP).

The concept of a JSP is the same as that for a Net.Data macro: HTML is used to construct the basic framework of the page, and a method of querying for and inserting dynamic data employed. In the case of JSPs, the mechanism is what's known as a Databean, which is a piece of Java code that is programmed to know how to get the information requested.

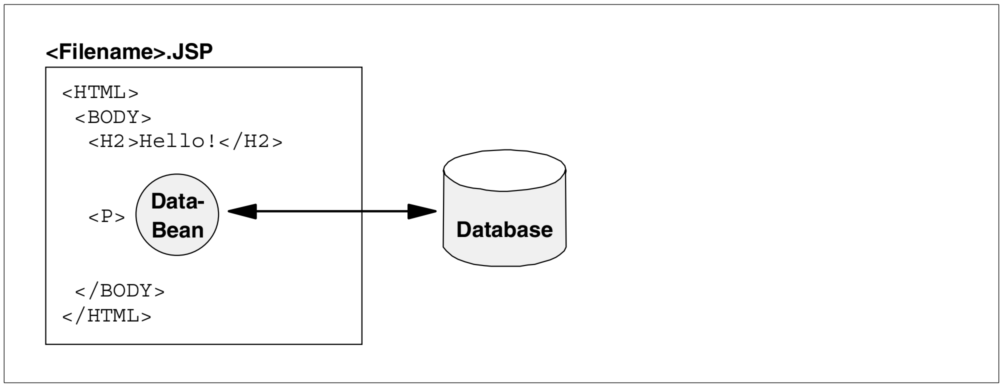


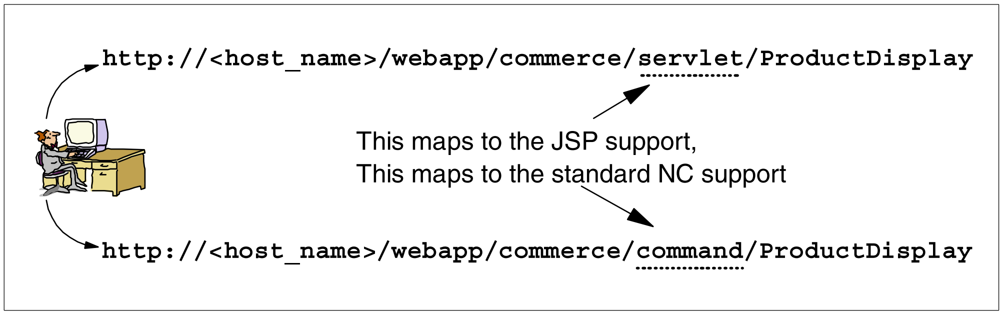
Figure 13. Conceptual overview of JSP page

There are three key things about this new function you should understand:

1. For WCSuite V4.1, at least initially, this new function is applicable to *only two* display environments:
   a. The `CategoryDisplay` function, which is used to display category pages
   b. The `ProductDisplay` function, which is used to display product pages
2. It is an entirely new architecture from the C++ architecture illustrated in 1.2.1.9, “Customizable command set” on page 11. That architecture, known as the NC^2 architecture in some circles, will remain for *at least* another release of

the product past WCSuite V4.1, but the strategic direction is definitely in the Java direction.

3. The command URL used to invoke the JSP page is different from the command URL used to invoke the standard NC^2 architecture. This difference is illustrated in Figure 14.



```
http://<host_name>/webapp/commerce/servlet/ProductDisplay
http://<host_name>/webapp/commerce/command/ProductDisplay
```

This maps to the JSP support,
This maps to the standard NC support

Figure 14. Different URL command structure used for JSP than for standard WCSuite command

The architecture is not fully developed or documented, so for the time being it is *not* open to customization and extension by you. Eventually it will be, and perhaps it will be before the next release of the product, but for now it is not.

This function is based on the use of “Servlets” (Java programs that run on the server rather than the client) and as such it requires that WebSphere Application Server (WAS) be part of your environment. An overview of what components are required for WebSphere Commerce Suite is provided in 1.2.3, “The system components of a WCSuite solution” on page 21. The process of installing, configuring, and testing these components together is documented in Chapter 2, “Prerequisites, installation and configuration” on page 29.

1.2.2.6 Multiple shopping carts

The shopping cart in WebSphere Commerce Suite is more correctly called the “interest item list,” but everyone still calls it the “shopping cart.” In Net.Commece Version 3.1.2 there was only one shopping cart per customer. With WCSuite V4.1, that’s been extended (through extensions in the database) to permit any number of shopping carts per customer. New commands have been added that allow you to take a product and add it to a given shopping cart or all shopping carts.

This feature will be used by site designers who see a need for permitting customers to have multiple carts. Some site designers may decide that one cart is adequate, and that’s certainly okay to do.

1.2.2.7 Catalog improvements

This feature is actually an extension to the database to facilitate the association of products to one another in relationships such as “cross sell” and “upsell.” The relationships, once defined in the database, may then be queried for and displayed using Net.Data. There’s not a great deal more to this than that.

1.2.2.8 Order flow improvements

This enhancement focuses on the ability to do quick orders, quick buys and reorders. Each of these revolve around two key concepts:

- Marking an existing order as a “template” and then using that “template” for other orders. This permits you to take the information associated with an existing order and copy it to a new order, which like any copy operation can save effort since much of the necessary information can be re-used.
- A new command called `OrderCopy`, which is a very long and complex command used to copy the orders you’ve previously marked as templates.

This feature is really intended for the skilled WCSuite site architect who is very familiar with the command structure and knows the order structure in the database quite well.

1.2.2.9 Mass Import utility improvements

The Mass Import utility that came with Net.Commerce Version 3.1.2 was functional, but was limited in two key ways:

1. It could only import to a set number of tables, and those tables were restricted to only the product catalog related tables.
2. The performance of the utility limited the usefulness of it for input file sizes beyond a point. For very large customers with tens or hundreds of thousands of products, this performance profile rendered Mass Import unusable.

To answer these concerns, the Mass Import utility was enhanced on both the input side (to allow more tables to be loaded), and on the output side (to enhance performance).Figure 15 illustrates what the structure is now.

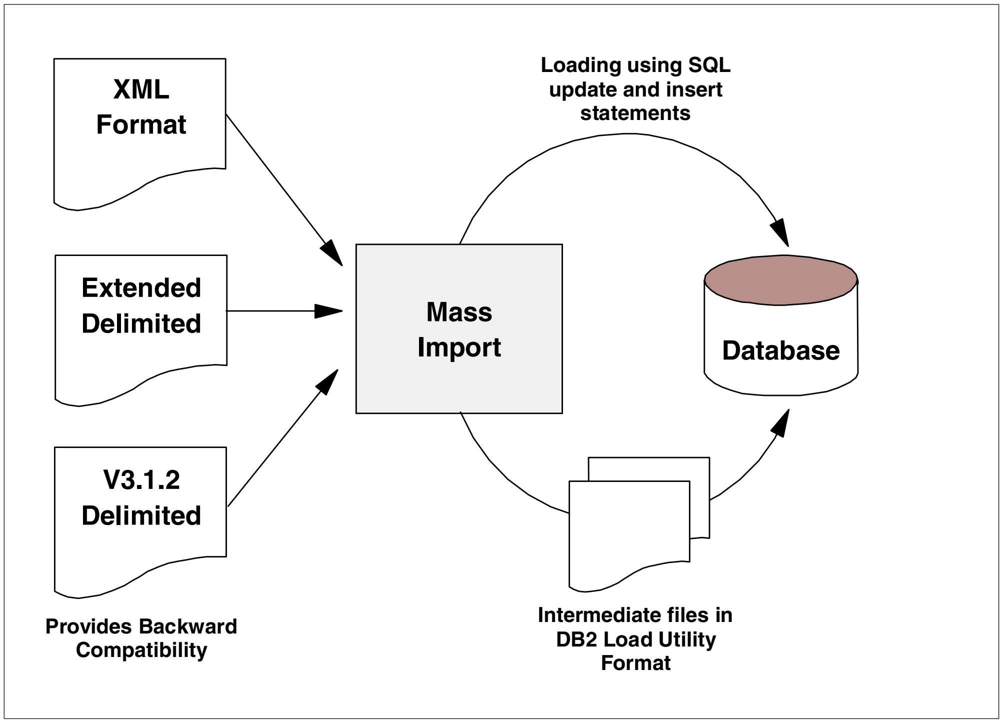

XML Format
Extended Delimited
V3.1.2 Delimited
Provides Backward Compatibility
Mass Import
Loading using SQL update and insert statements
Database
Intermediate files in DB2 Load Utility Format

Figure 15. Mass Import utility input and output methods

On the input side, two new input file formats have been provided. The key benefit of these two formats is that they allow you to define any number of tables (even custom tables of your creation) and import to those tables. The two new formats are:

- **XML**: This format uses XML tags to define the table schema of the database to which you will import data, as well as XML tags to define the data being imported. Each running of Mass Import using the XML option requires that three files be availabile to the utility: a DTD file (Data Tag Definition), which defines the XML tags to be used; a Definition file, which defines the table layouts and uses XML tags defined in the DTD file; and an input file that contains the data, wrapped in XML tags, that will be loaded into the tables.
- **“Extended” Delimited**: This looks very similar to the delimited format from Net.Commerce V3.1.2 except that it’s been extended to allow you to define additional tables into which to load data. Each running of Mass Import using this extended delimited format requires that two files be available to the utility: a Definition file, which defines the table layouts, and an input file that contains the actual data to be loaded.

For the sake of backward compatibility with any input files you might have from Net.Commerce V3.1.2 for OS/390, the utility will accept that format as well.

On the output side you have two different methods you may employ to get the data into the database:

- **Method = SQL**: When this method is used, the Mass Import utility will make a connection to DB2 and then use SQL insert and update commands to load the data into the database. This is the same method used by the Net.Commerce V3.1.2 utility, but it’s been enhanced to perform better.
- **Method = DB2LOAD**: When this method is used, the Mass Import utility will generate intermediate files in DB2 Load Utility format based on the input table definitions and data you provide the utility. The intermediate files include a JCL job stream and a column-offset data input file for each table to be loaded. It’s then up to you to submit the JCL, which invokes DB2 Load Utility to actually put the data into the tables. This method is provided because DB2 Load is a very high performance method of putting data into DB2.

1.2.3 The system components of a WCSuite solution

The picture shown in Figure 16 on page 22 illustrates the pieces to the puzzle for a WebSphere Commerce Suite V4.1 for OS/390 solution.

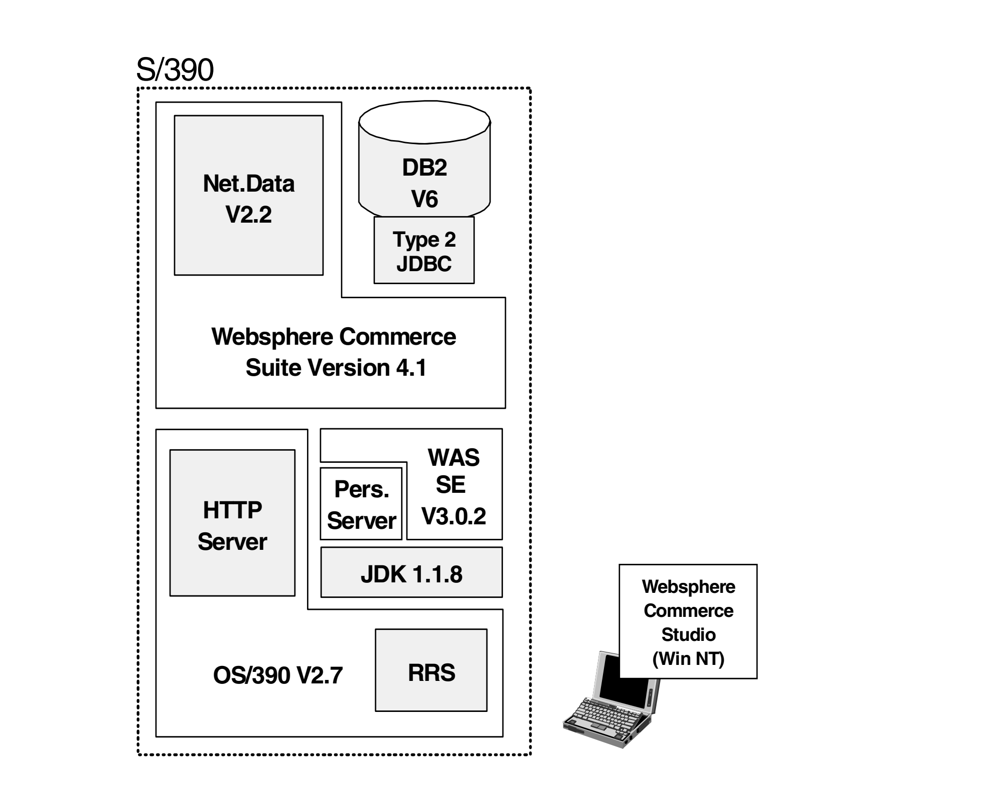


Figure 16. System components of a WebSphere Commerce Suite V4.1 solution

Not all of the components are strictly required: it depends on what functions of WCSuite you're looking to use. Table 1 on page 23 summarizes the relationship of the components to the functions of WebSphere Commerce Suite V4.1 for OS/390 that we covered in 1.2.2, "Significant new function added with WCSuite V4.1 OS/390" on page 13. We don't have the OS/390 operating system and the HTTP server in the table because they're required under all circumstances (well, not *all*: the Mass Import utility doesn't require the HTTP Server).

Table 1. Relationship between components and functions of WebSphere Commerce Suite V4.1

	RRS	DB2	Net Data	JDK	WAS	JDBC	Pers. Server	WCS Studio
Basic functions	■[1]	■	■					
Job scheduler	■	■						
Auctions	■	■	■					
JSP pages	■	■		■	■[2]	■		■[3]
Personalization	■	■	■	■	■[4]	■	■	■[5]
Multiple shopping carts	■	■	■					
Catalog improvements	■	■	■					
Order flow improvements	■	■	■					
Mass import		■[6]						

Notes:

[1] With WCSuite V4.1, the attachment facility between the Director code and DB2 has been change from CAF to RRSAF. Therefore, any function that requires the Director code to be involved requires RRS.

[2] JSP page support uses servlets, which requires WAS. WAS is also required if you “publish” JSP pages from Studio environment, since a servlet acts as the “catcher” for the publishing process.

[3] Strictly speaking, you don’t *have* to use the Studio to create JSP pages. But the references to the databeans are in the Studio’s pulldowns, so it makes it a much easier process.

[4] The personalization server is a Java program, and as such doesn’t itself require WAS. However, the Rules Builder is part of the WebSphere Commerce Studio, and to “publish” the rule sets to the OS/390 system you need to have WAS. This is because a servlet acts as the “catcher” for the publishing process, and servlets require WAS.

[5] To construct the rule sets or personalization, you must use the Blaze Rules Builder, which is part of the WebSphere Commerce Studio package.

[6] The observant reader may wonder why WAS isn’t required for the XML support of Mass Import. The reason is that the Mass Import utility uses the C++ XML parser and therefore the XML parser from WAS isn’t required.

We discuss how to verify that all these pieces of the puzzle are working together in Chapter 2, “Prerequisites, installation and configuration” on page 29.

1.2.4 The overall architecture

We have now come to the point where we’ll draw a big complex picture that tries to incorporate all the main points. Figure 17 on page 24 is this picture, and it illustrates the architecture of the WebSphere Commerce Suite V4.1 for OS/390 product.

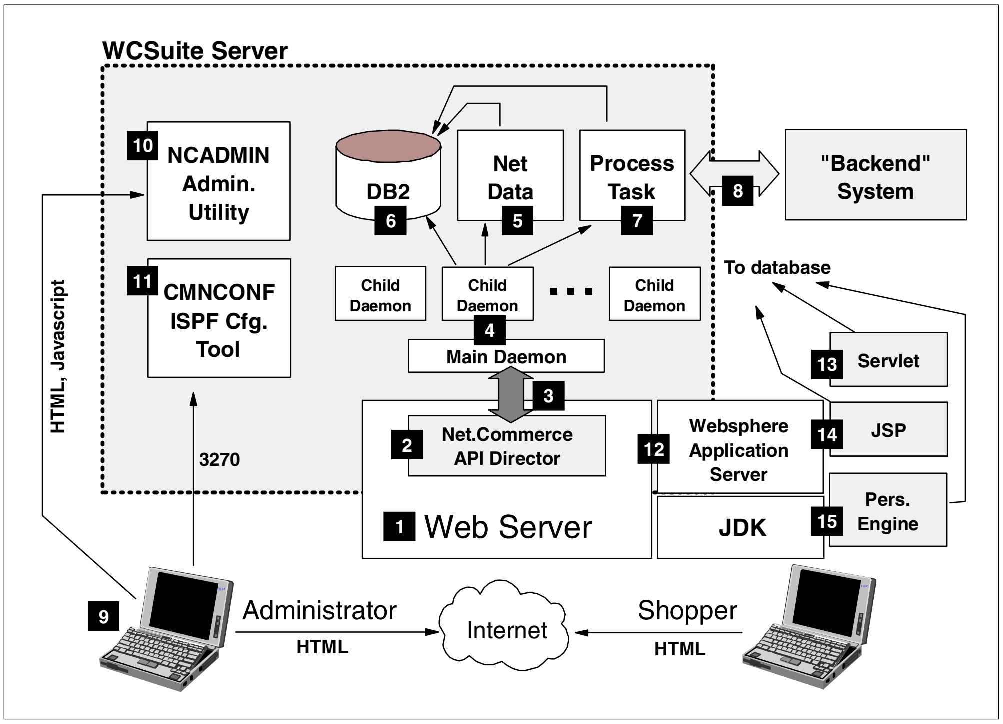


Figure 17. WebSphere Commerce Suite V4.1 for OS/390 overall architecture

Let's walk through each of the highlighted components shown in Figure 17:

1. The front-end of the entire system is the Webserver. (At one time called the Domino Go Webserver, it has been renamed the IBM HTTP Server, which is part of the WebSphere Application Server and which has been bundled with OS/390 V2R6). The Webserver receives the inbound HTTP requests from the browser and determines what should be done with each request:
   - Requests for static HTML pages are handled by the Webserver, which retrieves the HTML (and whatever image files might be associated with it) and delivers the page to the browser. In this scenario WCSuite is not involved at all.
   - Requests that include a WCSuite command are passed from the Webserver to the WCSuite Director, which is discussed next, or to the WebSphere Application Server if the command is one of the new Java commands (see #12).
2. The WCSuite Director is an implementation of the Go Webserver API interface (GWAPI), and is the portal through which the Webserver communicates with the WCSuite application. A GWAPI implementation is provided with WCSuite rather than a more traditional CGI implementation because of the superior performance characteristics of GWAPI. The WCSuite director actually runs in the Webserver's address space.
3. The WCSuite Director communicates across TCP sockets to the WCSuite application itself. The ports across which this communication occurs are determined by a configuration parameter set with the CMNCONF configuration utility. Generally, these ports are set at some high number (above 10000). It registers the ports in the `/etc/services` file.

4. The processing inside of WCSuite is handled by a main daemon and some number of child daemons. The main daemon passes off all requests to the child daemons on a round-robin basis. The number of child daemons is determined by a configuration parameter set by the configuration program. The default is 2 children, and the maximum number is 50. The more child daemons defined, the more work that can be accomplished.

   The child daemon, when executing a command, will do one of three things:

   - Access the database directly to extract information
   - Run a Net.Data macro that extracts information from the database and combines it with HTML to be issued back to the browser
   - Run a program, known as an API task, that either accesses the database or accesses other systems to complete the sales transaction.

5. Net.Data is a program that, at a high level, issues a SQL query against the database and wraps HTML around the results.
6. The WCSuite database consists of more than 200 tables arranged in a very structured model. The WCSuite database contains not just information about categories and products, but also information used to control the operation of the WCSuite application itself.
7. An API Task (now called a “Process Task”) is a program that runs to perform some function, such as checking the level of inventory, or calculating shipping charges.
8. Much of the business data in the world today exists on MVS or OS/390 systems, and many customers wish to access that data directly using exits from the WCSuite system. The API tasks just discussed are valuable for that reason: they allow programmers to come out of WCSuite processing and access these systems, returning control to WCSuite with the desired information.
9. Administrators are simply people with the knowledge and authority to add and modify information in the WCSuite database. The primary tool used to do this is a browser that accesses the administrative utility of WCSuite.
10. The NCADMIN administrative utility is a set of Javascript and HTML forms that provide an interface to the information contained in the database.
11. The CMNCONF configuration utility is a 3270-based tool that configures the “instances” of WCSuite. Instances are simply copies of WCSuite that run on the OS/390 system. Multiple instances may run at the same time, and they may either have their own database, or share a common database.
12. If the inbound URL specifies one of the new Java-based commands, the Webserver will pass the command over to the WebSphere Application Server so that the servlet that represents the command may be invoked. With WCSuite V4.1, two Java servlet commands are provided: ProductDisplay and CategoryDisplay.
13. The command is implemented in the form of a Java servlet, which will be invoked and run under the control of WAS. The servlet will contain the logic that implements the behavior of the command. The servlet will have access to the WCSuite database to query for necessary information from the database.

14. For the two servlet-based commands supplied with WCSuite V4.1, a JSP will be invoked by the servlet to render the dynamic HTML that will be presented back to the shopper.

15. For the Personalization feature of WCSuite, the “rules engine” runs as a Java program. If a rule is interrogated and invoked, it will be through this rules engine.

1.3 Why use OS/390 instead of other platforms?

This question really boils down to what’s important to the customer. All of the platforms on which WebSphere Commerce Suite runs have their strengths and weaknesses. OS/390 is no different. The following list summarizes what we think are the key strengths of OS/390 and should be considered when evaluating the platform choice for WCSuite:

- **Proximity to the backend data system**

  If the business data resides on OS/390 and you can locate your commerce application on the same system, then you can take advantage of shorter code path network access protocols to get at that data. The more efficiency you can squeeze from the transfer of data the better the site will perform.

- **Proven application platform**

  Many S/390 customers have years of experience running rock-solid applications on S/390. To them, the skill necessary to keep an application up and running on S/390 is relatively second-nature, so the inclusion of a web-based commerce application is a good fit.

- **Highly secure environment**

  Through the years the OS/390 environment has been equipped with a very strong set of security tools (such as RACF). Because the WCSuite for OS/390 product operates in this same environment, it uses these services. Properly configured, an OS/390 Web environment is practically impenetrable.

- **Very high “top end” for throughput**

  S/390 systems are renowned for having impressive throughput characteristics. With sufficient attention paid to sizing and tuning a WCSuite solution on S/390, you can construct a site with impressive performance characteristics. And if the need exists for true 24 x 7 uptime, Parallel Sysplex can enable that better than any other clustering technology available today.

Hosting a WCSuite solution on OS/390 provides many benefits, as this outline suggests, but for some customers another platform might be a better choice. IBM offers solutions across the entire spectrum of server platforms.

1.4 Our assumption of your understanding of WCSuite functions

This first chapter was dedicated to providing an overview of the WebSphere Commerce Suite product, but it was never intended to be the definitive source of information. As we were planning this redbook, we faced a fairly common dilemma when writing any documentation: how much prior knowledge can we assume our readers will have? The trap authors of technical documentation often fall into is feeling as if they should explain *everything*, and in doing so create an enormous document that is difficult to read or use.

We have intentionally decided to *not* fall into that trap. Instead, we concentrated this redbook on several specific areas based on the assumption of your skills as shown in Figure 18.

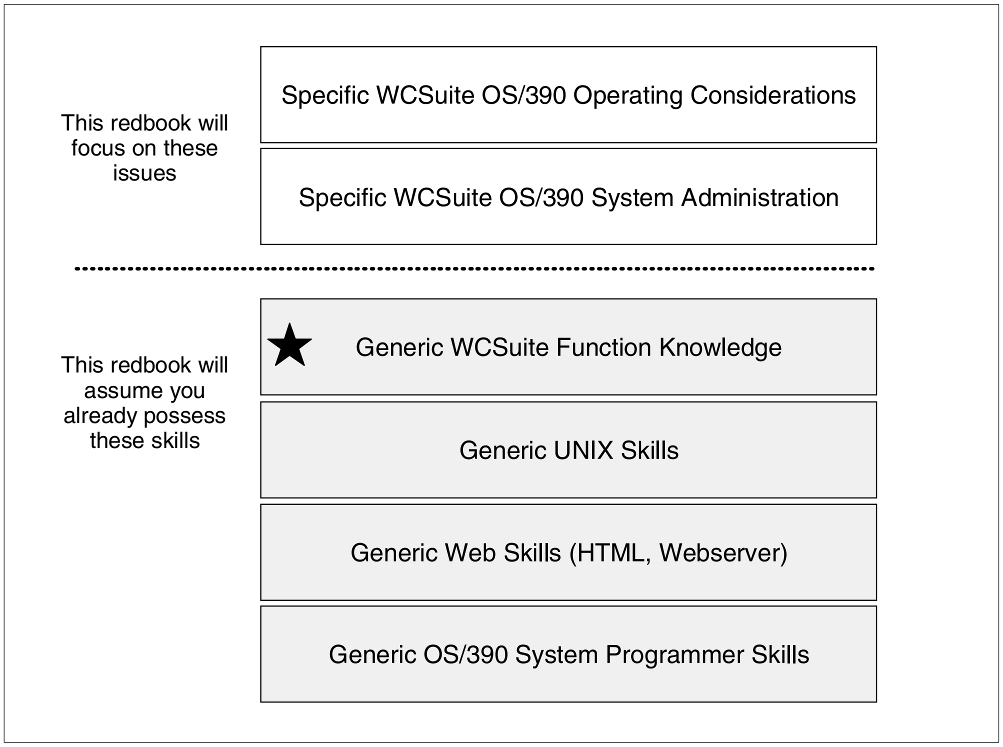

This redbook will focus on these issues
Specific WCSuite OS/390 Operating Considerations
Specific WCSuite OS/390 System Administration
This redbook will assume you already possess these skills
Generic WCSuite Function Knowledge
Generic UNIX Skills
Generic Web Skills (HTML, Webserver)
Generic OS/390 System Programmer Skills

Figure 18. Focus of this redbook

The fourth box from the bottom (the one marked with a star) is the one we struggled with the most. We decided to assume you already have a good knowledge of WebSphere Commerce Suite functions like auctions, personalization, the use of the administrative function, and so forth. We reached this decision because a good deal of excellent documentation and education already exists that covers those topic (see Appendix D, “Related publications and resources” on page 209). We make some reference to these things, particularly in Chapter 5, “Using WebSphere Commerce Studio” on page 127, but we chose not to make this redbook a complete tutorial on every functional aspect of WCSuite.

Chapter 2. Prerequisites, installation and configuration

This chapter is dedicated to the topics of installing, configuring, and verifying the function of the various components of a WebSphere Commerce Suite for OS/390 system. There are quite a few pieces to the puzzle (as we illustrated in Figure 1.2.3 on page 21), so the key to this is to take a logical approach. We show you how to install the product, and then configure and verify in a logical manner that things work the way they're supposed to.

2.1 Prerequisite software

To permit your solution to operate, some prerequisite software is needed. Figure 19 illustrates what's required.

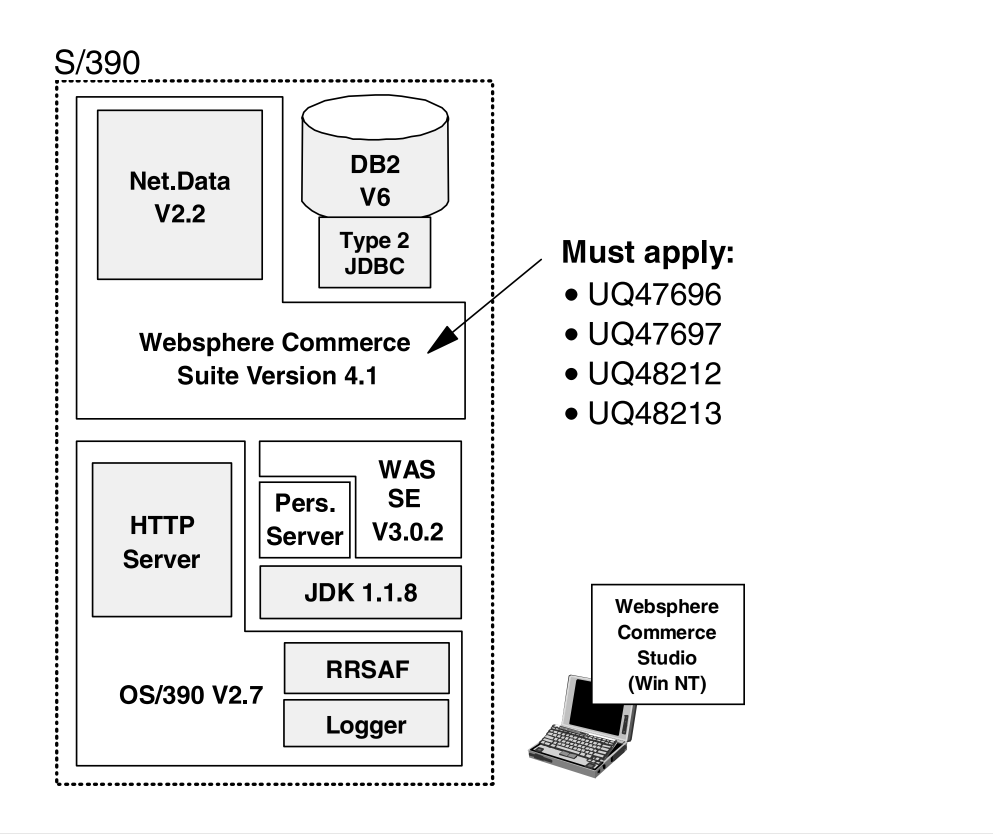


Figure 19. Prerequisite software for WebSphere Commerce Suite V4.1 for OS/390

The picture tells the story graphically. Table 2 provides some supporting information.

Table 2. Prerequisite software information

Prerequisite Software	Program Number	When or Why Required
OS/390 V2R7, R8, R9	5647-A01	Documented minimum level. A lower level of the operating system (down to V2R4) would *probably* function, but the minimum level of OS/390 that is supported with WCSuite V4.1 is OS/390 V2R7.

Prerequisite Software	Program Number	When or Why Required
RRSAF (Recoverable Resource Services Attachment Facility)	(part of OS/390 operating system)	Both the WebSphere Commerce Suite and the WebSphere Application Server use RRSAF to access the DB2 database. This facility must be started on the OS/390 system to configure and use the WCSuite product.
Logger	(part of OS/390 operating system)	Required by RRSAF. If you have a Sysplex environment, you can log to the Coupling Facility; if you have a standalone system you can log to DASD.
HTTP Server V5.1 or V5.2	(part of OS/390 operating system)	Minimum level required by WebSphere Commerce Suite V4.1 in support of it's "GWAPI" API code. The Webserver comes bundled with the OS/390 operating system; with OS/390 V2R7 came HTTP Server V5.1. OS/390 V2R8 had HTTP Server 5.2 bundled with it; OS/390 V2R9 came with HTTP Server 5.3.
JDK 1.1.8	5655-A46	Provides Java runtime environment for WAS SE V3.0.2, which is required for JSP page support and publishing from the Studio to OS/3290. Also provides Java runtime environment for Blaze Rules Server used for personalization support.
WebSphere Application Server Standard Edition Version 3.0.2 Requires PTFs UQ43991 and UQ44522	5655-A98	Required by servlets that provide JSP page support; required by servlet that provides "catcher" duties when publishing material from Studio to OS/390. Not strictly required for Blaze Rules Server (Java program, not servlet), but it is required if you wish to publish Rulesets from the NT-based Studio to OS/390.
IBM DB2 Version 6 Requires PTFs UQ41672, AQ39665, UQ43899, AQ39411 and UQ45727.	5645-DB2	Supplied database schema is defined with triggers, which requires DB2 V6 at a minimum. Trying to run WCSuite V4.1 with DB2 V5 will not work: the database-create step will fail because the trigger definitions in the script will be unrecognized. Removing the trigger definitions from the schema script would disable necessary function, so that is not a workaround. Use DB2 V6.
Type 2 JDBC (If you apply PTF UQ41672 to DB2 V6 with a Type 1 JDBC driver, it provides Type 2 JDBC support)	(part of DB2 subsystem)	Required to support database access by Java servlets and Databeans in JSP page support; required to support database access by Blaze Rules Server for personalization support. Also required by Servlet that provides "catcher" duties when publishing material from Studio to OS/390.

Prerequisite Software	Program Number	When or Why Required
WebSphere Commere Studio on Windows NT	5648-C35	This is the development environment on which such tools as the Store Creator Wizard, Visual Age for Java and the Blaze Rules Creator are run. If you wish to use those tools for your site, then this is required. However, it is *possible* to avoid the use of this suite of tools, but it would be *very difficult* to use the new Java/JSP and Personalization functions.

Check with the system administrator to insure the various pieces of this puzzle are installed and available.

2.2 The installation process

This step focuses on the SMP/E installation process. It is a fairly straightforward process, particularly for system programmers comfortable with SMP/E. Consult the Program Directory, GI10-8233, that comes with WebSphere Commerce Suite for detailed instructions on the installation process.

Note

The TSO userid from which you perform the install needs to be UID(0) to effectively complete the CMNISMKD job. This job creates the HFS subdirectories and external links, and without UID(0), the job will not complete properly.

The data sets and HFS system from this installation process will be those described in Table 3.

Table 3. Data sets and HFS data sets created by installation process

Data Set Name	Type	Contains
hlq.SCMNBASE	PDS	The jobs received from the installation tape and used by the installation process.
hlq.SCMNDBRM	PDS	The Commerce Suite DBRM library
hlq.SCMNLMOD	PDS	The Commerce Suite executables
hlq.SCMNMENU	PDS	The CMNCONF configuration utility message library
hlq.SCMNPENU	PDS	The CMNCONF configuration utility panel library
hlq.SCMNSAMP	PDS	The Commerce Suite samples
/usr/lpp/CommerceSuite	HFS	The HFS in which the Commerce Suite programs reside, as well as sample HTML files, Net.Data macros and other files used by WCSuite.

Important!

After installation is complete, apply the PTFs UQ47696, UQ47697, UQ48212 and UQ48213. The operation of WCSuite is seriously hampered without these PTFs. This redbook is written with the assumption these PTFs are applied.

When you have completed the installation process, you simply have the files off the distribution tape and on your system. Further system preparation and configuration is needed, as discussed next in this chapter.

2.3 Post-installation configuration and verification process

After the SMP/E installation process has been completed, there remains a list of things that you need to do to prepare and configure your system.

The approach we outline in this book involves working through the configuration process in a logical way that builds upon each prior step. At key intervals we show how to verify that the work done to that point is functional. The process is illustrated in Figure 20.

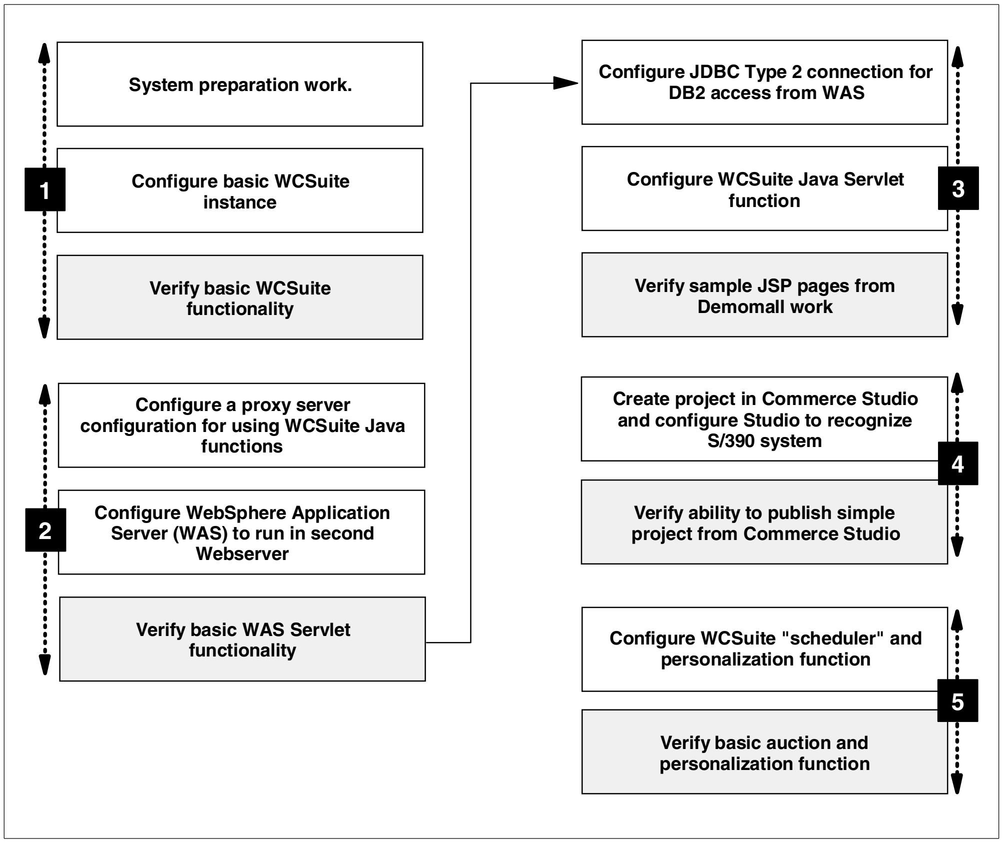

System preparation work.
Configure basic WCSuite instance
1
Verify basic WCSuite functionality
Configure a proxy server configuration for using WCSuite Java functions
Configure WebSphere Application Server (WAS) to run in second Webserver
2
Verify basic WAS Servlet functionality
Configure JDBC Type 2 connection for DB2 access from WAS
Configure WCSuite Java Servlet function
3
Verify sample JSP pages from Demomall work
Create project in Commerce Studio and configure Studio to recognize S/390 system
4
Verify ability to publish simple project from Commerce Studio
Configure WCSuite "scheduler" and personalization function
5
Verify basic auction and personalization function

Figure 20. Logical process for configuration and verification of WCSuite system

As you can see in Figure 20, we are suggesting a five-step approach, with verification steps at the end of each step. This process will confirm the basic operation of the WCSuite system. Any further customization you wish to do for your site would be in addition to the steps outlined here.

2.4 Phase 1: Basic WCSuite function verification

In this phase you verify that the basic system components are present, perform some system administration work, then test access to WCSuite using the basic functions provided by product. Follow the steps provided here.

2.4.1 Provide PUBLIC access for users

Defining PUBLIC on the Userid directive of the Webserver's configuration file (typically found at /etc/httpd.conf) provides a surrogate ID under which documents will be served to browsers. A commerce solution will imply many users — most of whom you do not know — accessing your site. Therefore, you will want to make certain the surrogate ID *PUBLIC* is used. PUBLIC is defined with very little authority.

The section of httpd.conf where the Userid directive appears looks like this:

```
# Example:
UserId      PUBLIC
# UserId     %%CLIENT%%
# UserId     %%CERTIF%%
# UserId     %%SERVER%%
# UserId     %%CLIENT%%
```

This assumes that the ID PUBLIC has been established. If is has not, here are the RACF commands to do that:

```
ADDGROUP GENERAL SUPGROUP(OMVSGRP) OMVS(GID(99))
ADDUSER PUBLIC DFLTGRP(GENERAL) NOTSO OMVS(UID(99))
```

The webserver's ID (typically WEBSRV) needs to have surrogate authority over the PUBLIC ID for this all to work. So issue these RACF commands:

```
RDEFINE SURROGATE BPX.SRV.PUBLIC UACC(NONE)
PERMIT BPX.SRV.PUBLIC CLASS(SURROGAT) ID(WEBSRV) ACC(READ)
```

2.4.2 Enable program control on datasets

If you have program control already defined, issue the command:

```
RALTER PROGRAM * ADDMEM('hlq.SCMNLMOD'/'volser'/NOPADCHK) UACC(READ)
```

where *hlq* is the high-level qualifier for the WCSuite data sets you used at installation and *volser* is the volume-serial. (If this is the first time program control is turned on, issue RDEFINE rather than RALTER. The rest of the command would be the same.) To refresh the program control profile, issue:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

If not already done, you will need to turn on program control for the data sets containing the run-time libraries for DB2 (hlq.SDSNLOAD and hlq.SDSNEXIT), C++ and LE as well. Failure to program control a data set used by the HTTP Server results in an Error 500 message on the browser screen.

2.4.3 Create group ID and user ID under which WCSuite will run

The recommended default group ID and user ID are CMNGRP and CMNSRV, respectively. To create the group ID, enter the following command:

```
ADDGROUP CMNGRP SUPGROUP(SYS1) OMVS(GID(2))
```

To create the user ID, enter the following four commands in the order shown:

```
ADDUSER CMNSRV DFLTGRP(CMNGRP) OMVS(UID(0) HOME('/usr/lpp/CommerceSuite') PROGRAM ('/bin/sh'))
PERMIT BPX.DAEMON CLASS(FACILITY) ID(CMNSRV) ACCESS(READ)
PERMIT BPX.SERVER CLASS(FACILITY) ID(CMNSRV) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

2.4.4 Update STEPLIB in Webserver's start procedure

This provides the Webserver access to the data sets it needs to run the WCSuite “director” component. In this example, the high-level qualifier for WCSuite is `WCS41`, and the high-level qualifier for the C++ Class Libraries is `CBC`.

```
//STEPLIB  DD DSN=WCS41.SCMNLMOD,DISP=SHR
//         DD DSN=CBC.SCLBDLL,DISP=SHR
//         DD DSN=DSN610.SDSNLOAD,DISP=SHR
```

You do not need to do this if these data sets are defined in LNKLSTxx or PROGxx.

2.4.5 Update Webserver's environment variables file

The Webserver's environment variables file is typically /etc/httpd.envvars. Update the NLSPATH variable to include:

```
/usr/lpp/CommerceSuite/msg/en_US/%N
```

And update the LIBPATH variable to include:

```
/usr/lpp/CommerceSuite/lib
```

You will need to stop and restart the Webserver to pick up this change, as well as the change made to the httpd.conf file.

2.4.6 Bind WCSuite DBRM plan

Binding the WCSuite database plan allows a WCSuite instance the ability to communicate with its database. It is also critical because if you don't do it, you won't be able to create or load the database. A sample BIND job is supplied in the `hlq.SCMNSAMP(CMNBIND)` data set member. Simply modify the JCL according to the instructions provided in the comments and submit the job.

A return code of 4 is normal.

2.4.7 Update WCSuite environment variables file

The WCSuite product has its own environment variables file, which provides information about the environment to the WCSuite system. This gets a little tricky because there are four copies of the WCSuite environment variables file when the CMNCONF configuration program is finished. This step focuses on two things:

1. Updating the sample envvars file (called ncommerce.envsamp) to reflect your system environment.
2. Copying that updated sample to /etc and renaming it to ncommerce.envvars. This provides the CMNCONF utility an environment file which it will use to operate.

Do the following three things in preparation for running CMNCONF:

1. Make a backup of the original ncommerce.envsamp found in the directory /usr/lpp/CommerceSuite/install directory.
2. Edit the *original* ncommerce.envsamp and make sure the settings in this file match your environment.

   IBM WebSphere Commerce Suite Pro Edition for OS/390 Configuration Guide Version 4.1 (SC27-0696) has information on the contents of this file. Generally, you won't have to make too many changes unless you've installed the HTTP Server at something *other than* /usr/lpp/internet or WCSuite at something *other than* /usr/lpp/CommerceSuite.
3. Copy the ncommerce.envsamp to the /etc directory, and rename it ncommerce.envvars. This will provide the environment variables with which CMNCONF itself will run.

2.4.8 Create shared object files for CMNCONF

Before you run CMNCONF, you need to create shared object files that point from the HFS to load module members in the hlq.SCMNLMOD data set. This only needs to be done once, immediately after installation.

Do the following:

1. Go to an OMVS shell.
2. Switch to the /usr/lpp/CommerceSuite/lib directory.
3. Run the `setupnc2.sh` shell script located in that directory.

2.4.9 Prepare TSO environment to run CMNCONF panel application

To run the CMNCONF program, you need to provide it two things:

1. Access to the WCSuite hlq.SCMNLMOD data set
2. Access to the WCSuite message (hlq.SCMNMENU) and panel (hlq.SCMNPENU) data sets

2.4.9.1 Access to SCMNLMOD

This is accomplished either by placing hlq.SCMNLMOD in LNKLSTxx or PROGxx, or by activating it for your TSO session directly. In this redbook we'll focus on the latter. (If you have the data set in LNKLST but have not yet activated the changes, then you can issue the `TSOLIB` command to temporarily activate it while you wait for LNKLST to be refreshed).

From the `TSO READY` prompt (*not* the ISPF Option 6 command line) issue the command:

```
TSOLIB ACT DSN('hlq.SCMNLMOD')
```

Then proceed into ISPF. The CMNCONF utility will now have access to its executables.

2.4.9.2 Access to message and panel libraries

Provide your ISPF environment access to the WCSuite panel and message libraries for CMNCONF, as shown in Table 4 on page 36.

Table 4. ISPF library concatenations

ISPF Library	WCSuite Data Set
ISPPLIB	hlq.SCMNPENU
ISPMLIB	hlq.SCMNMENU

This is done by concatenating the data sets to the ISPPLIB and ISPMLIB DD statements in your TSO logon procedure. As an alternative to updating your TSO logon procedure, you may also create a CLIST that performs LIBDEF commands to give your session access to the datasets. On our test system used to write this book, that CLIST was called NETCOM and it looked like this:

```
/*REXX*/
trace e
address "ISPEXEC" "CONTROL ERRORS RETURN"
address "ISPEXEC" "LIBDEF ISPPLIB DATASET ID('CMN.SCMNPENU')"
address "ISPEXEC" "LIBDEF ISPMLIB DATASET ID('CMN.SCMNMENU')"
address "ISPEXEC" "LIBDEF ISPLLIB DATASET,
         ID('CMN.SCMNLMOD','DSN610.SDSNLOAD')"
```

Consult your system programmer if you are not certain where your logon procedure is, or how to create and invoke CLISTs.

2.4.10 Enable superuser if TSO ID is not UID(0)

When you installed the WCSuite product, the directory tree off the mount point of /usr/lpp/CommerceSuite will have been created with permissions of 755 (“write” for the owner, “read” for the group and “read” for others). The CMNCONF program will attempt to create directories under /usr/lpp/CommerceSuite, and unless the TSO ID under which CMNCONF is running has the authority, the process will fail. The first indication of a problem will be a “Cannot create directory” message on the second System Configuration panel of CMNCONF.

The easiest way around this is to run CMNCONF from a TSO ID with UID(0). However, even if your TSO ID’s OMVS segment does not have UID(0), it is possible to achieve superuser status for the TSO ID. This is done by turning on the BPX.SUPERUSER facility class of RACF and then defining your ID with READ access to the facility. Once done, you can toggle to superuser in ISHELL or OMVS.

To run CMNCONF you must get yourself back to the ISPF Option 6 panel. Unfortunately, doing so *sometimes* results in your losing superuser status. To work around this and guarantee superuser for CMNCONF, use the process illustrated in Figure 21 on page 37.

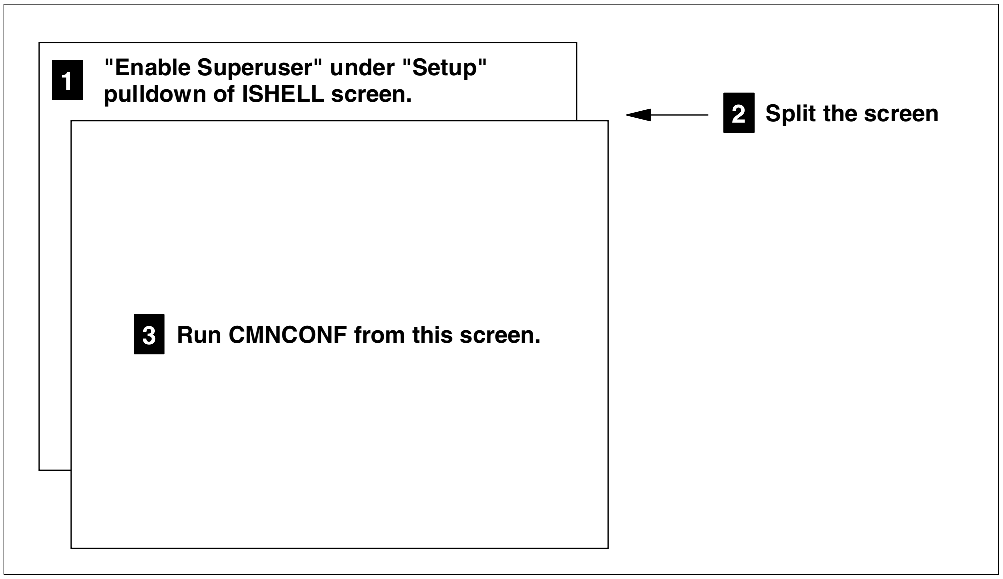

1 "Enable Superuser" under "Setup" pulldown of ISHELL screen.
2 Split the screen
3 Run CMNCONF from this screen.

Figure 21. Splitting the screen and running CMNCONF

2.4.11 Prepare separate HFS and mount-point for configuration files

By *default*, the CMNCONF utility will scatter configuration files into several different directories. This is illustrated in Figure 22.

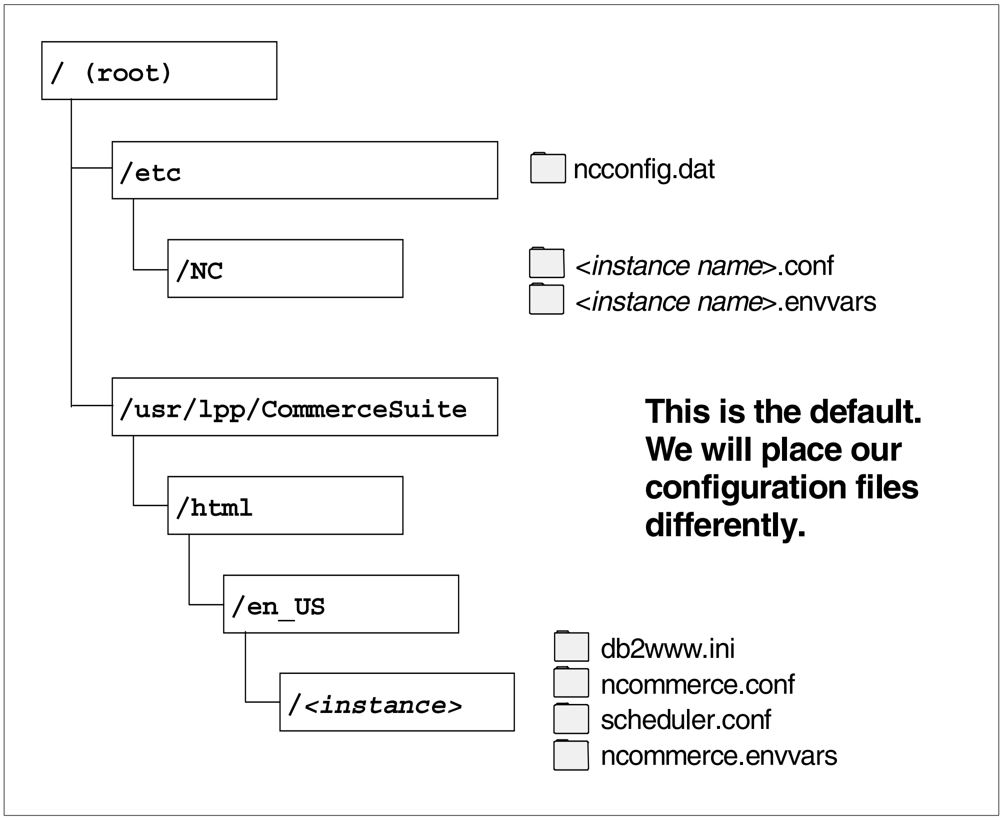


Figure 22. Where CMNCONF placed configuration files by default

These are the configuration files for just the basic WCSuite functions. There are more when you bring into play the Java functionality. This gets to be a concern for customers for the following reasons:

- It is confusing.
- Customers generally want to locate all the configuration files in a single, separate directory, and have that directory be its own HFS system. This aids in backup and recovery, and for migrating environments across systems.

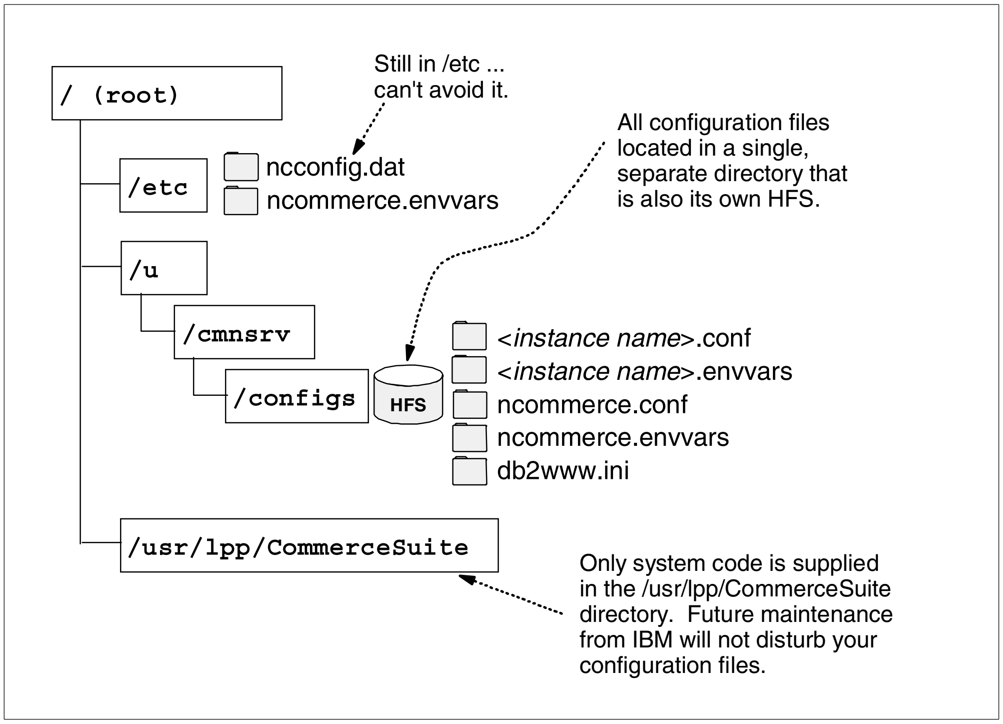
Fortunately, this is possible with WCSuite, and is fairly simple. Do the following:

1. Create a directory in your HFS system that will serve as a mount-point for a separate HFS that will hold your configuration files. This can be anywhere in the directory structure, but the recommendation is that it *not* be in the /usr/lpp/CommerceSuite directory, and that it be located somewhere relatively low in the tree.

   For the sake of illustration, assume that directory is /u/cmnsrv/configs.

2. Make sure that directory has permission bits of 755, and all the directories leading up to that new directory have 755 or higher. When WCSuite goes looking for its configuration file, it has to be at read and execute on the files to operate.
3. Allocate a separate HFS and mount it at the /u/cmnsrv/configs directory. The HFS doesn’t have to be that large: the configuration files will end up being only a few thousand bytes.

Now you have an isolated location for your configuration files, as illustrated in Figure 23.



/ (root)
Still in /etc ...
can't avoid it.
/etc
ncconfig.dat
ncommerce.envvars
All configuration files located in a single, separate directory that is also its own HFS.
/u
/cmnsrv
/configs
HFS
<instance name>.conf
<instance name>.envvars
ncommerce.conf
ncommerce.envvars
db2www.ini
/usr/lpp/CommerceSuite
Only system code is supplied in the /usr/lpp/CommerceSuite directory. Future maintenance from IBM will not disturb your configuration files.

Figure 23. Single directory for configuration files

The files do not yet exist, as you have not yet run the CMNCONF utility. But when you do run the utility, you will point to this directory as the location into which the files will be written.

Note 1 You may wish to configure your system to automount this HFS at IPL time. Otherwise, it will require you to manually mount it each time the system is IPLed.

Note 2 The example we've shown you here (/u/cmnsrv/configs) works fine when you have only one instance of WCSuite configured. If you wish to utilize separate instances you will have to utilize separate directories for each instance's configuration files. This is because each instance will use a file called ncommerce.conf, and to maintain uniqueness you will have to have them in separate directories. This can be easily accomodated by having sub-directories under the configs directory in our example: /u/cmnsrv/configs/inst1 and /u/cmnsrv/configs/inst2.

2.4.12 Configure an instance using the CMNCONF utility

An *instance* of WebSphere Commerce Suite is a running copy of the program on your OS/390 system. The design of the product permits you to configure and run more than one instance within a single OS/390 operating system. The reasons for doing this vary; usually it's done to provide different testing groups their own copy of WCSuite to start and stop and modify as they please.

WCSuite instances are configured using the CMNCONF configuration utility. CMNCONF is an ISPF panel application that takes input from you and then makes the necessary updates on your system to allow an instance of WCSuite to run.

2.4.12.1 Overview of the CMNCONF configuration utility

The CMNCONF configuration program is an ISPF panel-based application. It has four main sections; you create your configuration by stepping through the sections one at a time . Figure 24 on page 40 illustrates the flow of the program.

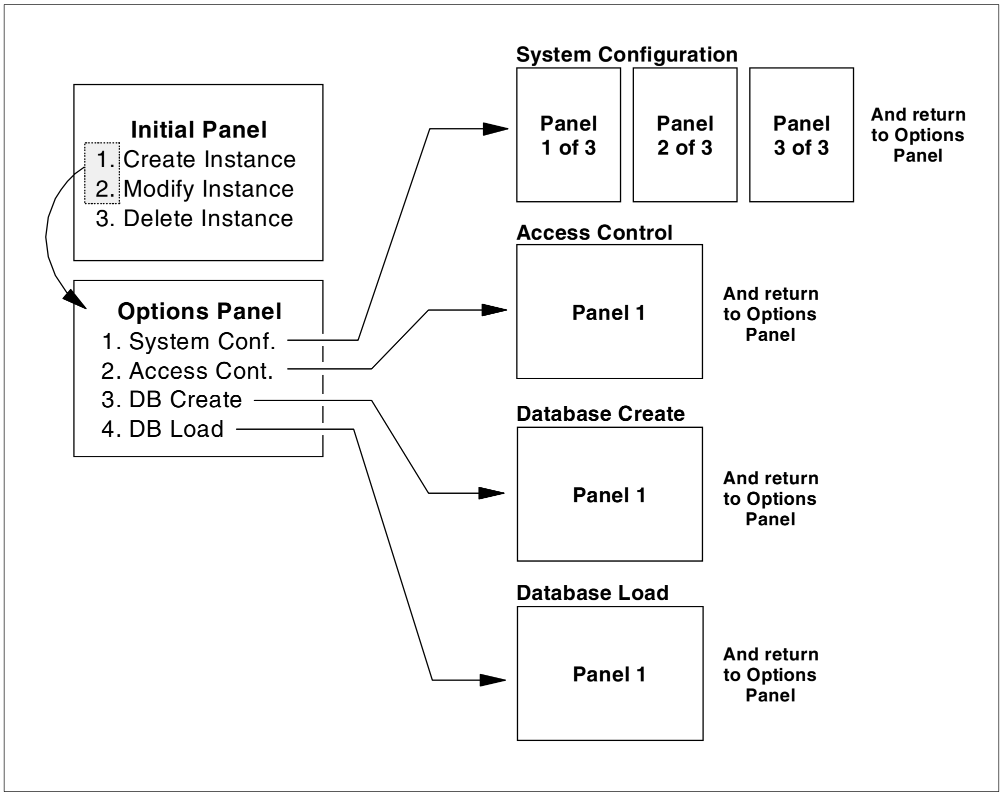


Figure 24. Flow of a CMNCONF configuration run

Note 1: Stepping through all four is typically only done on your first run, or when you are creating an entirely new instance. If you are modifying an instance, typically only options 1 and 2 are performed.

Note 2: The configuration files are created and the updates to httpd.conf made after you press **Enter** on the Access Control (Option 2) panel.

Note 3: We weren't completely truthful in this picture: Options 4 (Database Load) has a second panel if you're creating a custom database. But that's not a commonly used option and we won't cover it in this book.

The output from CMNCONF is a set of configuration files, which is discussed in the next section.

2.4.12.2 Overview of CMNCONF configuration files created

The purpose of the CMNCONF program is to create a set of configuration files that represent the instance of WCSuite. Figure 25 on page 41 illustrates the configuration file output from the CMNCONF program.

Note

The illustration we provide in this book shows the placement of all the configuration files into a single directory. That is different from the default placement of the configuration files.

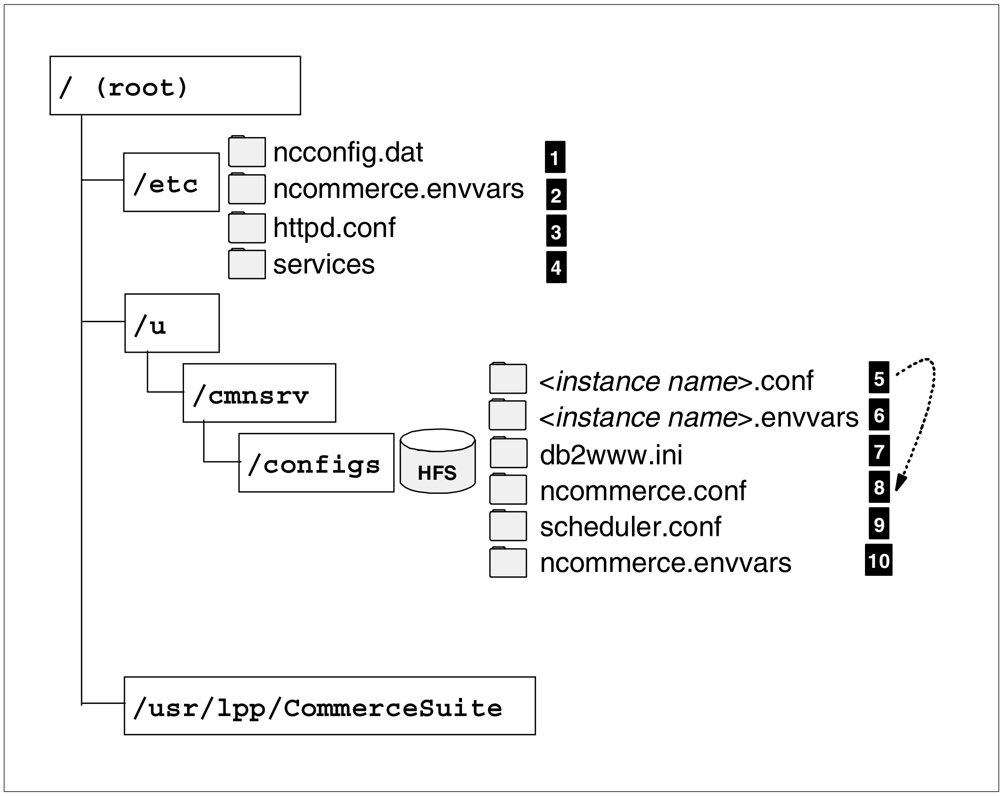


Figure 25. Directory and file output from the CMNCONF utility

Following is a brief description of these files.

1. The ncconfig.dat file is created by CMNCONF and it contains one line per WCSuite instance created. This is how CMNCONF knows what's already been configured on the system.

2. The nccomerce.envvars file is an environment variables file that you hand-copy down to the `/etc` directory to give the CMNCONF program (which is a UNIX Systems Service program) an environment in which to run. The WCSuite product comes with a sample envvars file, and back in 2.3, "Post-installation configuration and verification process" on page 32 you were given instructions on updating this to match your system and then copying it over to `/etc`.

3. The httpd.conf file is the Webserver's configuration file. The CMNCONF program will update this file with Pass and Service directives necessary to tell the Webserver how to communicate with the WCSuite product. You may point to this file anywhere in your HFS system. Typically the file resides in /etc, but many people isolate the Webserver's configuration files just as we're isolating our WCSuite configuration files. So it may be somewhere other than /etc.

4. The services file under the /etc directory is used to record TCP port numbers used by the various communication services on the system. The TCP ports across which the WCSuite director code talks with the main daemon code are registered in this file (see "The overall architecture" on page 23 for a description and picture of the director and daemon relationship). The CMNCONF program will create this file if it doesn't exist, or it will simply update it with the port numbers if the file already exists.

Note: There's a potential problem here that you should be aware of. If your system utilizes a services file in an MVS PDS file and you have no /etc/services file, then CMNCONF will create the /etc/services file with its port values. Unfortunately, the search order used by OS/390 to locate the port information is UNIX Systems Servics first, and then MVS PDS. And if it finds the USS file it'll stop looking and therefore not find your MVS PDS file. That means your TCP services defined in the MVS PDS file will not work

5. The <instance name>.conf file is a configuration file for the "Server Controller." It's not really important that you understand what a Server Controller is, but it is important that you know this file exists, and that it is in this file that a pointer is made to the primary configuration file shown at points #8 and #9 in Figure 25 on page 41. The CONTROL_POOL_CONFIG directive in the <instance>.conf file is what points to the scheduler.conf configuration file, as well as the ncommerce.conf file

   There is a field on a CMNCONF utility panel that allows you to specify the directory into which this file will go. Since our objective is to place all of our configuration files into a single, separate directory, we will use that panel input field to specify the /u/cmnsrv/configs directory.

6. In the same directory in which the <instance name>.conf file resides there is another copy of the environment variables file for WebSphere Commerce Suite. And just like for the configuration file, the name is made unique by appending the instance name to the file name. This envvars file supplies the Server Controller with the environment it needs to operate.

7. The initialization file for Net.Data is called db2www.ini. It is in this file that Net.Data finds information about the directories in which it will search for macros, as well as other information used by Net.Data.

8. The ncommerce.conf file is the primary configuration file for your instance. The "ncommerce" portion of this name is a reference to the Net.Commerce roots of the WebSphere Commerce Suite product. Much of the information you put into the fields of the CMNCONF ISPF panel application finds its way into this file.

   This file is referenced by a pointer in the <instance>.conf file. The CONTROL_POOL_CONFIG directive in the <instance>.conf file has a reference to the ncommerce.conf configuration file, as well as the scheduler.conf file.

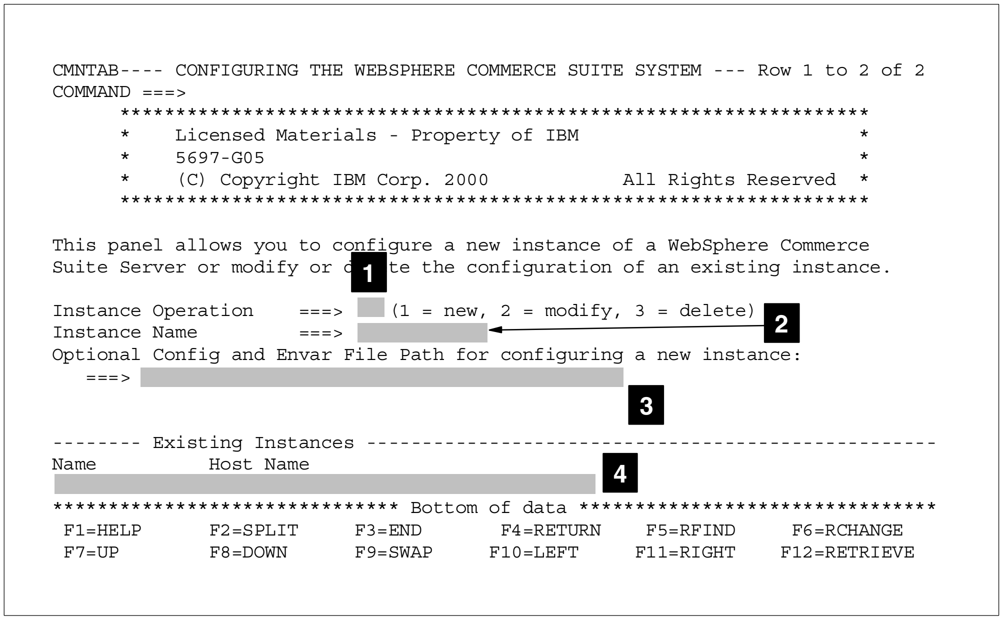
9. The scheduler.conf file is used by the WCSuite Job Scheduler (see 1.2.2.1, "Job scheduler" on page 13 for a description of the function of the Job Scheduler).

   This file is referenced by a pointer in the <instance>.conf file. The CONTROL_POOL_CONFIG directive in the <instance>.conf file has a reference to the scheduler.conf configuration file, as well as the ncommerce.conf file.

10. Finally, there is yet another copy of the environment variables file and it is located in this directory as well (that makes a total of three copies of the envvars file). It should be identical to the other copies of the envvars file, and indeed the CMNCONF program uses a single "model" as its source for these files.

2.4.12.3 Starting CMNCONF and creating an instance

You start CMNCONF by going to the ISPF Option 6 panel and entering the command `CMNCONF`. If you have performed all the tasks identified up to this point, you should get the screen shown in Figure 26.



```
CMNTAB---- CONFIGURING THE WEBSPHERE COMMERCE SUITE SYSTEM --- Row 1 to 2 of 2
COMMAND ===>
      ****************************************************************
      *    Licensed Materials - Property of IBM                      *
      *    5697-G05                                                  *
      *    (C) Copyright IBM Corp. 2000           All Rights Reserved  *
      ****************************************************************

This panel allows you to configure a new instance of a WebSphere Commerce
Suite Server or modify or delete the configuration of an existing instance.

Instance Operation    ===>    (1 = new, 2 = modify, 3 = delete)
Instance Name         ===>
Optional Config and Envar File Path for configuring a new instance:
   ===>

-------- Existing Instances --------------------------------------------------
Name           Host Name

******************************* Bottom of data ********************************
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 26. CMNTAB: The first panel of CMNCONF

The CMNTAB panel is the “Initial Panel.” This panel allows you to create new instances, modify existing instances, or delete instances. There are four input fields on this panel (referenced by number in Figure 26):

1. **Instance Operation**: This is where you specify what you wish to do with this invocation of CMNCONF. The choices are `1` for a new instance, `2` for modifying an existing instance (which would require you to then name the instance in the Instance Name field), and `3` for deleting an instance (which again would require you to name the instance).

   For a new instance, use `1`.

2. **Instance Name**: You can name an instance anything that you would like, up to eight characters. Since the name will be used in the file name of two of the configuration files, the instance name cannot contain special characters that Unix Systems Services doesn’t allow for file or directory names.

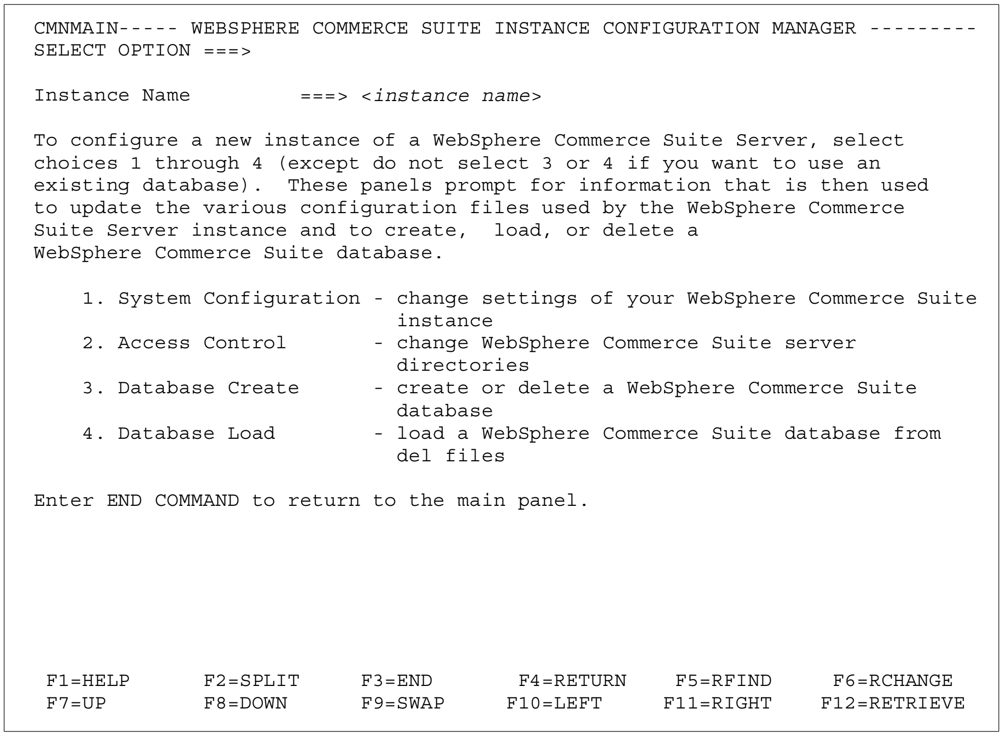
   In our example the instance name is wcs41. By the way, like all Unix systems, the name is case-sensitive.

3. **Optional Config and Envvars File Path**: This is used to specify a directory in which a model configuration file resides. The idea is you can have a model configuration file populated with information for your site, and it’ll save you time filling in fields when you run CMNCONF. However, we have found that some of the information in the model input file doesn’t make it to the final configuration file, which means there’s a bug in the CMNCONF code. So we won’t make use of this feature.

4. **Existing Instances**: This field lists all the other instances already configured on the system. You can use F7 and F8 to scroll up and down in this list if it is

long. The CMNCONF utility knows about other instances based on the information it finds in /etc/ncconfig.dat.

When you press the enter key after filling in this information you are taken to the primary options panel of the CMNCONF utility, which is illustrated in Figure 27.



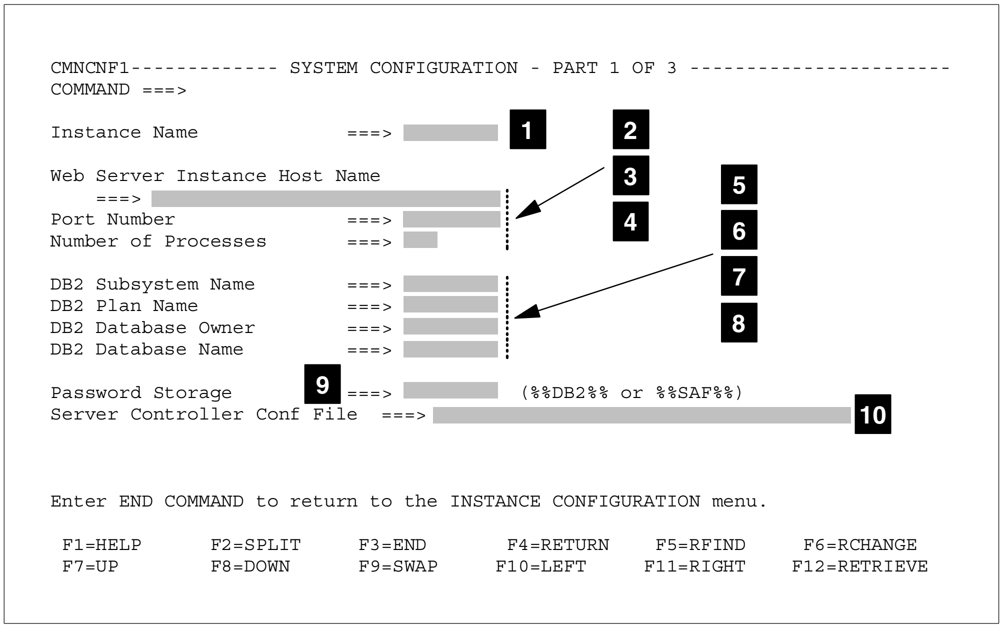
```
CMNMAIN----- WEBSPHERE COMMERCE SUITE INSTANCE CONFIGURATION MANAGER ---------
SELECT OPTION ===>

Instance Name         ===> <instance name>

To configure a new instance of a WebSphere Commerce Suite Server, select
choices 1 through 4 (except do not select 3 or 4 if you want to use an
existing database).  These panels prompt for information that is then used
to update the various configuration files used by the WebSphere Commerce
Suite Server instance and to create,  load, or delete a
WebSphere Commerce Suite database.

    1. System Configuration - change settings of your WebSphere Commerce Suite
                              instance
    2. Access Control       - change WebSphere Commerce Suite server
                              directories
    3. Database Create      - create or delete a WebSphere Commerce Suite
                              database
    4. Database Load        - load a WebSphere Commerce Suite database from
                              del files

Enter END COMMAND to return to the main panel.






 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 27. CMNMAIN: The main option panel

The CMNMAIN panel is the same thing as the “Options Panel” shown in Figure 24 on page 40. From here you simply select the option you wish to go to. You will return to this panel after completing each of the configuration steps.

2.4.12.4 CMNCONF System Configuration option

This option has two panels associated with it. The first panel is illustrated in Figure 28.



```
CMNCNF1------------- SYSTEM CONFIGURATION - PART 1 OF 3 ----------------------
COMMAND ===>

Instance Name             ===>

Web Server Instance Host Name
   ===>
Port Number               ===>
Number of Processes       ===>

DB2 Subsystem Name        ===>
DB2 Plan Name             ===>
DB2 Database Owner        ===>
DB2 Database Name         ===>

Password Storage          ===>           (%%DB2%% or %%SAF%%)
Server Controller Conf File  ===>


Enter END COMMAND to return to the INSTANCE CONFIGURATION menu.

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 28. CMNCONF System Configuration Part 1 of 3 panel

A description of the fields in this panel follows. (The numbers included with the definitions refer to the notations in Figure 28.)

1. **Instance Name**: This field will be filled in for you based on the instance name you specified on the CMNTAB panel.
2. **Webserver Instance Host Name**: This field is used to specify the host name URL to be used by this instance of WCSuite. It must be in the form of a host name (*not* dotted decimal IP address).

   Many people will simply use the host name of their IP stack for this field, and that is perfectly acceptable. However, for testing purposes you may wish to use a different host name. The Webserver will see the URL on the inbound request and associate it with the directives in httpd.conf for that host name.

   For the sake of illustration, assume you use the host name of the IP stack of your system. At the ITSO where this redbook was written the IP host name was wtsc54oe.itso.ibm.com. We used that for our “Webserver Instance Host Name” as well.
3. **Port Number**: This is the local TCP port across which the director code will communicate with the WCSuite server. If you look back to Figure 17 on page 24, this is the “3” reference tag on the illustration. The CMNCONF utility will pick a number relatively high in value as the starting point for the range of ports it will use. CMNCONF wil look in the /etc/services file (if one exists) to see if anything else on the system is using that range of port numbers. WCSuite will use as many ports as “Processes” (see next item in this list), plus one.

   In almost all cases you can simply accept the value CMNCONF comes up with. The only issue would be if you have another process on your system that uses a port in the range CMNCONF assigned.
4. **Number of Processes**: This refers to the number of child processes that will be run within the WCSuite server address space to handle work requests. This

number defaults to 2, and may be up to 50 (although the online help says 99, don’t let that number go above 50). The more processes employed, the more work that can be handled. The more processes, the more resources consumed. (At this point we do not have information on the system resources consumed per process.)

For your first configuration, accept the default of 2.

5. **DB2 Subsystem Name**: This is the DB2 subsystem name on which the WCSuite database will reside. Your system may have multiple DB2 subsystems, but the WCSuite database can only reside on one. Specify the subsystem name. For this redbook, the subsystem name was DB2P.

6. **DB2 Plan Name**: This value should match the value used when you bound the database plan back in 2.4.6, “Bind WCSuite DBRM plan” on page 34. The sample BIND job (CMNBIND) has a default value of mserver, so the default value of mserver in CMNCONF will match. If you changed the value in the BIND job, then change it here to match.

7. **DB2 Database Owner**: This value should match the name of the ID you created back in 2.4.3, “Create group ID and user ID under which WCSuite will run” on page 33. The example we provide in this redbook is CMNSRV.

8. **DB2 Database Name**: This value may be anything you wish, as long as the database name doesn’t already exist. For your first time, use DEMOMALL since that’s the sample mall we will have you load.

9. **Password Storage**: This value specifies where WCSuite will go to verify the password provided when a shopper logs into the system. There are two options:

   - `%%DB2%%` means WCSuite will go look in the SHOPPER table of the WCSuite database for the password. The userid/password combination is stored in SHOPPER when a user registers.
   - `%%SAF%%` means WCSuite will go look in RACF (or equivalent) for the password. Using this option implies additional manual updates to the httpd.conf file, which are not covered in this redbook.

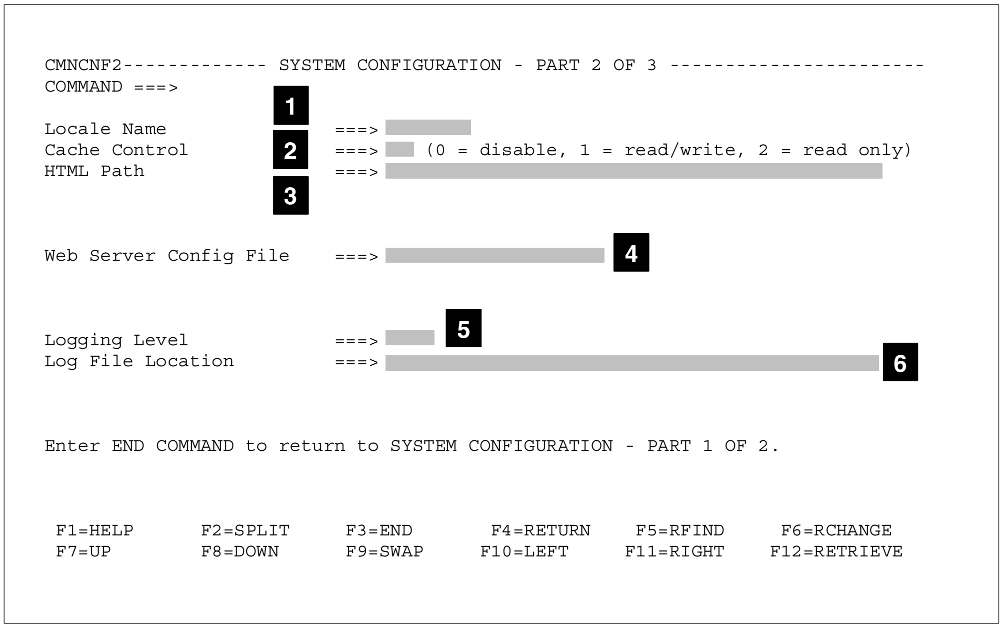
   When you first install WCSuite, you should use `%%DB2%%`. It is far simpler and allows you to validate all the connections of the system first. Later, you may change this to access RACF if you wish.

10. **Server Controller Conf File**: This field is used to specify the name and location where CMNCONF will write out the “server controller configuration file”, which is the file referenced with the number 5 tag in Figure 25 on page 41.

    The default for this is /etc/NC/<instance name>.conf, but since we are trying to force all the configuration files into a single directory, you should specify that directory. So for this example, this field would be filled in with:

    ```
    /u/cmnsrv/configs/wcs41.conf
    ```

When you are finished filling in the fields on this panel, press the Enter key to move to the next panel — panel 2 of 3 for System Configuration — which is shown in Figure 29.



```
CMNCNF2------------- SYSTEM CONFIGURATION - PART 2 OF 3 ----------------------
COMMAND ===>

Locale Name              ===>
Cache Control            ===>     (0 = disable, 1 = read/write, 2 = read only)
HTML Path                ===>


Web Server Config File   ===>


Logging Level            ===>
Log File Location        ===>



Enter END COMMAND to return to SYSTEM CONFIGURATION - PART 1 OF 2.



 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 29. CMNCONF System Configuration Part 2of 3 panel

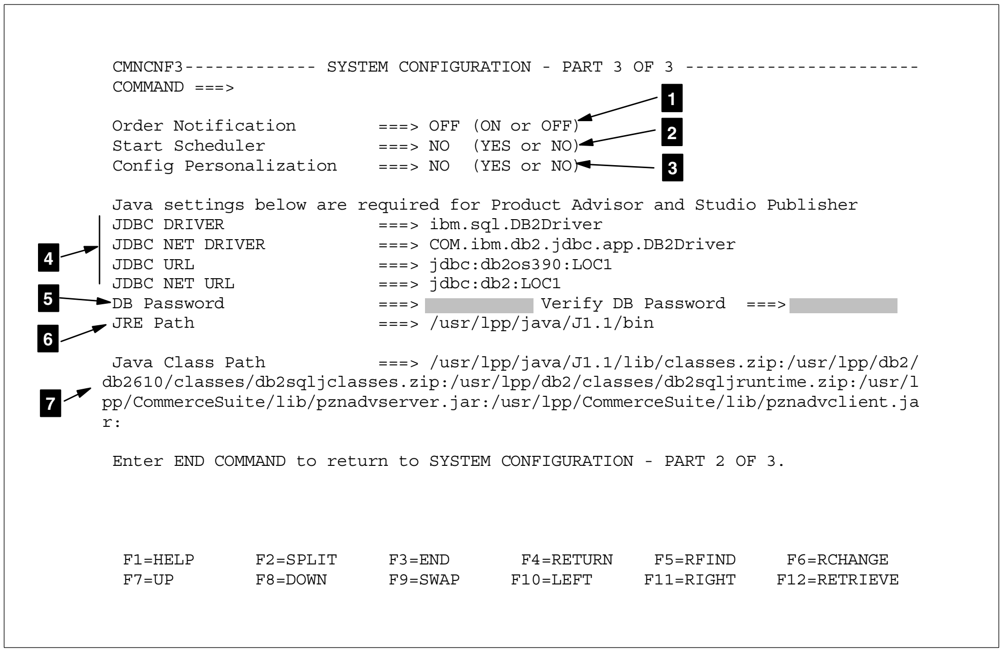
The fields of the CMNCNF2 panel have the following meanings:

1. **Locale Name**: This defaults to en_US, and unless you're very well versed in international language issues, leave it at en_US.
2. **Cache Control**: This controls the feature that caches the HTML output from Net.Data macros. It is a very nice feature for improving performance, but it is not something you want activated when you first start out with WCSuite. This value defaults to 1, which means caching will take place.

   Our recommendation is to set this to 0 initially, and change it to 1 or 2 only after you are comfortable with the operation of the WCSuite site.
3. **HTML Path**: This tells CMNCONF where to place the configuration files. It also defines the "document root" for the WCSuite instance.

   To achieve the single location for configuration files, you must specify that directory location here. In our example, that would be /u/cmnsrv/configs.
4. **Webserver Config File**: This tells CMNCONF where to go to get the HTTP Server's configuration file. By default this is /etc/httpd.conf, but your system may have a different location and file name. Simply point to the file you use.
5. **Logging Level**: This controls the trace level for the WCSuite server. This defaults to 0 and should remain 0 unless you need to do some debugging.
6. **Log File Location**: By default the output from the server tracing ("logging level" set greater than zero) will go to the SRVOUT and SYSPRINT DD locations for the started task. You can reroute that trace output to HFS files by specifying a directory location here. For now, leave this blank.

When you press the enter key you will be taken to the third panel of "System Configuration," which is shown in Figure 30.



```
CMNCNF3------------- SYSTEM CONFIGURATION - PART 3 OF 3 ----------------------
COMMAND ===>

Order Notification        ===> OFF (ON or OFF)
Start Scheduler           ===> NO  (YES or NO)
Config Personalization    ===> NO  (YES or NO)

Java settings below are required for Product Advisor and Studio Publisher
JDBC DRIVER               ===> ibm.sql.DB2Driver
JDBC NET DRIVER           ===> COM.ibm.db2.jdbc.app.DB2Driver
JDBC URL                  ===> jdbc:db2os390:LOC1
JDBC NET URL              ===> jdbc:db2:LOC1
DB Password               ===>           Verify DB Password  ===>
JRE Path                  ===> /usr/lpp/java/J1.1/bin

Java Class Path           ===> /usr/lpp/java/J1.1/lib/classes.zip:/usr/lpp/db2/
db2610/classes/db2sqljclasses.zip:/usr/lpp/db2/classes/db2sqljruntime.zip:/usr/l
pp/CommerceSuite/lib/pznadvserver.jar:/usr/lpp/CommerceSuite/lib/pznadvclient.ja
r:

Enter END COMMAND to return to SYSTEM CONFIGURATION - PART 2 OF 3.



 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 30. CMNCONF System Configuration 3 of 3

The fields of the CMNCNF3 panel are described as follows:

1. **Order Notification**: Controls whether the administrator will receive order notification messages if that facility is also configured using the NCADMIN tool. This redbook will not cover this topic. Leave this at the default of “Off.”
2. **Start Scheduler**: This is used to configure the scheduler function, which is critical to the auction facility and automated ordering. In 2.8, “Phase 5: WCSuite scheduler and personalization validation” on page 99 you will do this. For now, leave this set at its default of “No.”
3. **Config Personalization**: This is used to configure the personalization function. In 2.8, “Phase 5: WCSuite scheduler and personalization validation” on page 99 you will do this. For now, leave this set at its default of “No.”
4. **JDBC information**: These four fields are used to configure the JDBC connectivity for the instance of WCSuite. Review the information with the DB administrator to ensure the information is correct. Modify it to fit the JDBC environment at the location.
5. **DB Password**: The password for the ID under which the WCSuite instance runs (typically CMNSRV) will have a password in RACF. The WCSuite configuration file ncommerce.conf will have an encrypted string for DB_PASS that *must* match the password value found in RACF. These two fields are used to supply the CMNCONF program with the password so that it can be encrypted and placed in ncommerce.conf. Check with the security administrator to coordinate how the password for CMNSRV (which is a UID(0) ID and whose password will be guarded closely) can be entered here.
6. **JRE Path**: This field is used to point to where the “Java Runtime Environment” (JRE, also known as the JDK) is installed on this system. If the default value is not correct, modify it to match the system's JRE installation location.

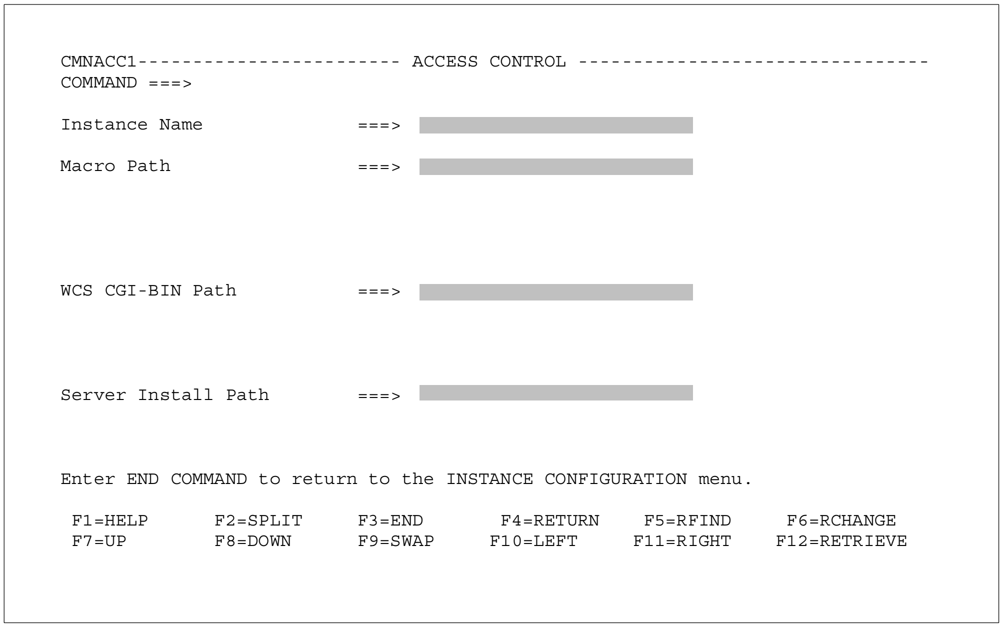
7. **Java Class Path**: This field is used to provide a classpath for Java classes needed by WCSuite. The field is populated with directories and ZIP or JAR file names. Review the string to make sure the *directories* are correct.

Press the Enter key and you will be returned to the CMNMAIN panel with a message indicating “Server Config Successful.”

Warning

No output is written after completion of the “System Configuration” steps of CMNCONF. That occurs after the “Access Control” option is completed. If you make changes to the information contained in the System Configuration panels, always remember to do the Access Control step as well to make sure the changes are updated in the configuration files.

2.4.12.5 CMNCONF Access Control option



```
CMNACC1------------------------ ACCESS CONTROL --------------------------------
COMMAND ===>

Instance Name              ===>

Macro Path                 ===>



WCS CGI-BIN Path           ===>



Server Install Path        ===>


Enter END COMMAND to return to the INSTANCE CONFIGURATION menu.

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 31. CMNCONF Access Control panel

The Access Control step is important because the output to the configuration files takes place after Access Control has completed. The fields on this panel default to what you should need, *unless you have installed WCSuite in a directory other than /usr/lpp/CommerceSuite*.

Assuming you have not modified the default install path for WCSuite, just press the Enter key. You should get a message saying “Access Control Succeeded” and find yourself back at the CMNMAIN panel.

2.4.12.6 CMNCONF Database Create option

There are two important things that must be done *prior* to attempting to create the database using CMNCONF:

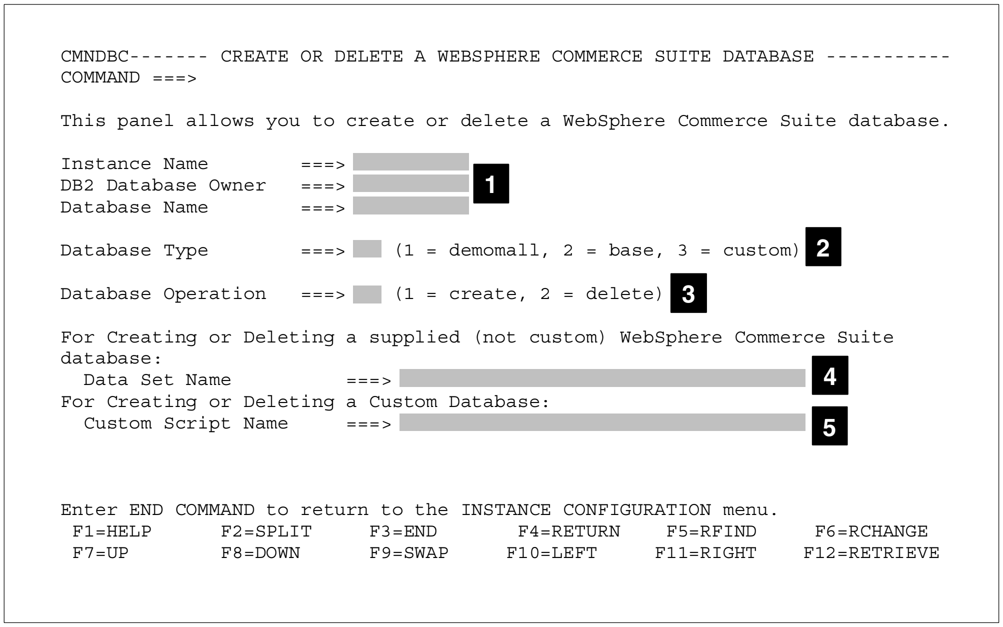
1. The database plan must be bound. See 2.4.6, “Bind WCSuite DBRM plan” on page 34 for a description of that process.

2. The DB2 subsystem must be running.

Note

> At the time of the writing of this Redbook, there appears to be a problem introduced with PTFs UQ47696, UQ47697, UQ48212 and UQ48213. A member of the hlq.SCMNSAMP dataset called DEMOMALL is missing. Without that member, you can't create the database using this step. You will need to acquire a copy of that member before you can execute this step.

The panel used for creating the database is illustrated in Figure 32.



```
CMNDBC------- CREATE OR DELETE A WEBSPHERE COMMERCE SUITE DATABASE -----------
COMMAND ===>

This panel allows you to create or delete a WebSphere Commerce Suite database.

Instance Name        ===>
DB2 Database Owner   ===>
Database Name        ===>

Database Type        ===>     (1 = demomall, 2 = base, 3 = custom)

Database Operation   ===>     (1 = create, 2 = delete)

For Creating or Deleting a supplied (not custom) WebSphere Commerce Suite
database:
  Data Set Name           ===>
For Creating or Deleting a Custom Database:
  Custom Script Name      ===>



Enter END COMMAND to return to the INSTANCE CONFIGURATION menu.
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 32. CMNCONF Database Create option panel

1. **Instance Name**, **DB2 Database Owner** and **Database Name**: We have combined these three fields into one explanation because on this panel they are not open for update. The values for these fields are filled in automatically by CMNCONF based on information you supplied in "System Configuration," so you need not do anything with these fields on this panel.

2. **Database Type**: This field is used to specify the type of database you wish to create. WCSuite supplies two sample databases:

   - **Demomall**: This database will be populated with a half-dozen or so stores and some sample products when you run the "Database Load" option. We recommend tat you start with this database because it's an excellent way of validating the basic functionality of WCSuite without having to do any customization.

   - **Base Mall**: This is a skeleton mall used as a starting point for further customization. This should only be used if you are very familiar with the operation of WCSuite.

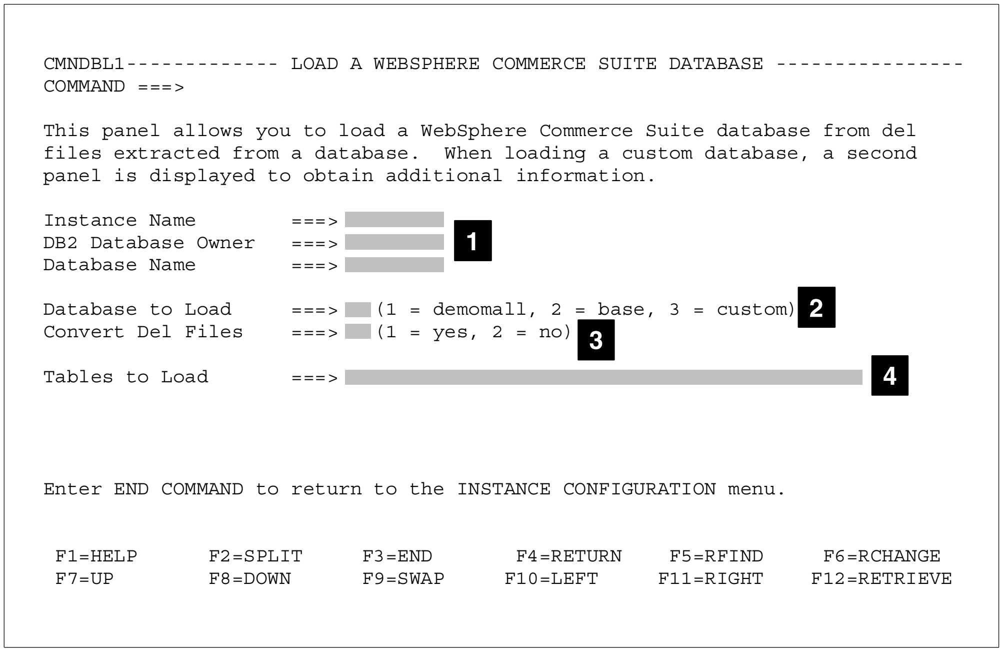
   For your first configuration you should select 1 to create the Demomall database.

3. **Database Operation**: You may either create or delete a database using this panel of CMNCONF. When you are doing your first configuration, select 1 to create the database.
4. **Data Set Name**: This field is used to point to the data set that contains the definitions of the tables that comprise the database. This defaults to `hlq.SCMNSAMP`. The member name of `DEMOMALL` will be automatically assumed by CMNCONF. Therefore, change this to reflect the high-level qualifier you used when you installed WCSuite.
5. **Custom Script Name**: This field is used to point to a custom database SQL script to use to create a custom database. Custom databases are an advanced topic and beyond the scope of this book. Leave this field blank.

Press the Enter key to proceed with the creation of the database. It will take several minutes to complete the operation, so be patient. Your TSO ID will remain occupied during this time period.

2.4.12.7 CMNCONF Database Load option

Once the Demomall database has been created, it needs to be loaded. An empty database is of no use to WCSuite. The sample data to be loaded into the database you just created is supplied with WCSuite, and you can load it using the “Database Load” option. The panel is illustrated in Figure 33.



```
CMNDBL1------------- LOAD A WEBSPHERE COMMERCE SUITE DATABASE ----------------
COMMAND ===>

This panel allows you to load a WebSphere Commerce Suite database from del
files extracted from a database.  When loading a custom database, a second
panel is displayed to obtain additional information.

Instance Name        ===>
DB2 Database Owner   ===>
Database Name        ===>

Database to Load     ===>   (1 = demomall, 2 = base, 3 = custom)
Convert Del Files    ===>   (1 = yes, 2 = no)

Tables to Load       ===>



Enter END COMMAND to return to the INSTANCE CONFIGURATION menu.


 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 33. CMNCONF Database Load option panel

A description of these fields follows.

1. **Instance Name**, **DB2 Database Owner** and **Database Name**: We have combined these three fields into one explanation because on this panel they are not open for update. The values for these fields are filled in automatically by CMNCONF based on information you supplied in “System Configuration,” so you need not do anything with these fields on this panel.
2. **Database to Load**: Match the type of database that you created. In our example, it would be the Demomall, so choose 1.

3. **Convert Del Files**: This option is used when you are bringing in data files from another platform’s DB2 system. We are not doing that here, so take the default value of 2 for “no.”
4. **Tables to Load**: This field allows you to specify individual files you want to load. This is an advanced feature and not to be used without very careful consideration. Allow this to remain blank.

Press the Enter key to proceed with the loading of the database. Just like with creating the database, this operation can take several minutes.

2.4.13 Grant DBADM to the Webserver’s userid

This step is necessary so that the WCSuite Director code, which is provided as a Webserver API program (commonly referred to as a “GWAPI” program, which is a reference to the old “Domino **G**o **W**ebserver **API**”), can access the WCSuite database. The Director runs in the Webserver’s address space and therefore operates under the authority of the Webserver’s userid.

To do this, issue the following DB2 SQL command:

```
grant dbadm on database <database name> to <Webserver userid>
```

Failure to do this will result in some rather random-looking failures. (There’s nothing random about them, of course, but it’ll appear that way when some things work and others don’t). This is an easy step to overlook.

2.4.14 Create your JCL start procedure

WCSuite runs as a started task, so that implies a JCL start procedure. A sample is supplied with the code you installed, and you may copy that code and modify it.

2.4.14.1 Copy the sample JCL procedure

The sample JCL is supplied in two locations:

- In the hlq.SCMNSAMP(CMNMSERV) PDS member
- In the /usr/lpp/CommerceSuite/install/cmnmserv.prc HFS file

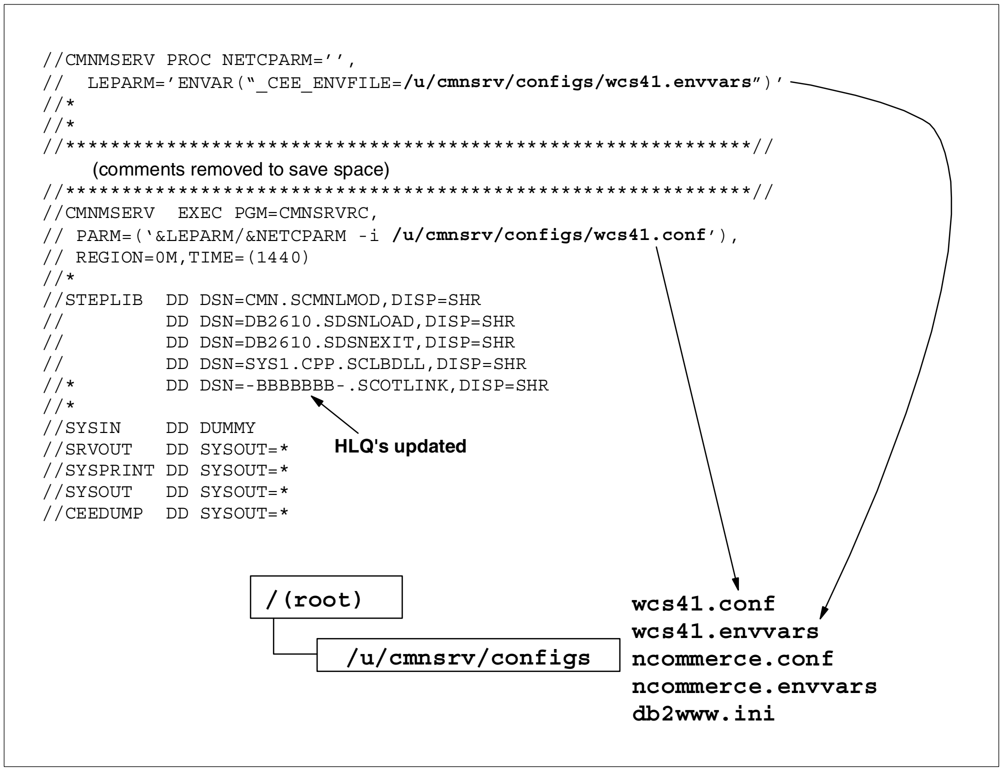
The sample JCL looks like this:

```
//CMNMSERV PROC NETCPARM='',
//  LEPARM='ENVAR("_CEE_ENVFILE=/etc/NC/-INSTNAME-.envvars")'
//*
//*
//*************************************************************//
//*                                                           *//
//* Licensed Materials - Property of IBM                      *//
//* 5697-D32                                                  *//
//*                                                           *//
//* (C) Copyright IBM Corp. 1997       All Rights Reserved.   *//
//* US Government Users Restricted Rights - Use,              *//
//* duplication or disclosure restricted by GSA ADP           *//
//* Schedule Contract with IBM Corp.                          *//
//*                                                           *//
//*************************************************************//
//* This PROC starts a Net.Commerce server instance.          *//
//*                                                           *//
//* . Change -INSTNAME- to the instance name defined for      *//
//*     this server during configuration.                     *//
//*                                                           *//
//* . Change -XXXXXXX- to the high level qualifiers for       *//
//*     Net.Commerce datasets.                                *//
//*                                                           *//
//* . Change -YYYYYYY- to the high level qualifiers for       *//
//*     DB2 datasets.                                         *//
//*                                                            *//
//* . Change -ZZZZZZZ- to the high level qualifiers for        *//
//*     Lotus Domino Go Webserver datasets.                    *//
//*                                                            *//
//* . Change -CCCCCCC- to the high level qualifiers for        *//
//*     C++ Class Library datasets.                            *//
//*                                                            *//
//* . If this instance is using the Payment Server support,    *//
//*   - change -BBBBBBB- to the high level qualifiers for      *//
//*     Payment Server datasets and un-comment the record.     *//
//*                                                            *//
//*************************************************************//
//CMNMSERV  EXEC PGM=CMNSRVRC,
// PARM=(‘&LEPARM/&NETCPARM -i /etc/NC/-INSTNAME-.conf’),
// REGION=0M,TIME=(1440)
//*
//STEPLIB  DD DSN=-XXXXXXX-.SCMNLMOD,DISP=SHR
//         DD DSN=-YYYYYYY-.SDSNLOAD,DISP=SHR
//         DD DSN=-YYYYYYY-.SDSNEXIT,DISP=SHR
//         DD DSN=-CCCCCCC-.SCLBDLL,DISP=SHR
//*        DD DSN=-BBBBBBB-.SCOTLINK,DISP=SHR
//*
//SYSIN    DD DUMMY
//SRVOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

Copy the JCL to your system proc lib.

2.4.14.2 Modify the sample

The sample JCL has commented instructions on what to change. To help clear up any possible confusion, Figure 34 illustrates which of the many configuration files you point to on the LEPARM and PARM statements in the JCL. This example assumes the directory structure we created in 2.4.11, “Prepare separate HFS and mount-point for configuration files” on page 37.



```
//CMNMSERV PROC NETCPARM=’’,
//  LEPARM=’ENVAR(“_CEE_ENVFILE=/u/cmnsrv/configs/wcs41.envvars”)’
//*
//*
//**************************************************************//
      (comments removed to save space)
//**************************************************************//
//CMNMSERV  EXEC PGM=CMNSRVRC,
// PARM=(‘&LEPARM/&NETCPARM -i /u/cmnsrv/configs/wcs41.conf’),
// REGION=0M,TIME=(1440)
//*
//STEPLIB  DD DSN=CMN.SCMNLMOD,DISP=SHR
//         DD DSN=DB2610.SDSNLOAD,DISP=SHR
//         DD DSN=DB2610.SDSNEXIT,DISP=SHR
//         DD DSN=SYS1.CPP.SCLBDLL,DISP=SHR
//*        DD DSN=-BBBBBBB-.SCOTLINK,DISP=SHR
//*
//SYSIN    DD DUMMY
//SRVOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

Figure 34. Which configuration files to point to from JCL start procedure

2.4.15 Create a STARTED class profile

To associate a RACF userid under which the started task will run, you need to add the start procedure to the STARTED class. Do this with the following command (which assumes your JCL start procedure is named CMNMSERV and the userid is CMNSRV):

```
RDEFINE STARTED CMNMSERV.** STDATA(USER(CMNSRV))
```

and then:

```
SETROPTS RACLIST(STARTED) REFRESH
```

2.4.16 Enabling SSL on your Webserver

WebSphere Commerce Suite will, by default, require you to have SSL (Secure Socket Layer) configured and working on your Webserver. While it is possible to disable WCSuite's use of SSL, it's somewhat difficult. It is also beyond the scope of this book to show you how to do it. So assume that SSL is required.

Briefly explained, SSL is a way of encrypting the data that flows between a browser and a server. Through a defined set of request and response exchanges (known as the "handshake"), the browser and the server agree on the necessary details so that both sides know how to receive and read the encrypted data. The purpose of SSL is to prevent people who may be eavesdropping on the line from intercepting information and reading it.

SSL is enabled on the Webserver through a combination of directives in the `httpd.conf` file and the existence of a certificate "keyring" on the system. The keyring is what holds the electronic certificates that are issued by the Webserver on your OS/390 system. These certificates tell the browser who the server is, and more important, it tells which "certificate authority" has "signed" the certificate. Your browser has a list of well-known "certificate authorities" coded into its security plug-in. If the certificate received from your server is signed by an authority not recognized by the browser, the browser will pop up a warning box telling you this fact.

Figure 35 on page 55 illustrates keyring location and relationship of the directives in the httpd.conf file.

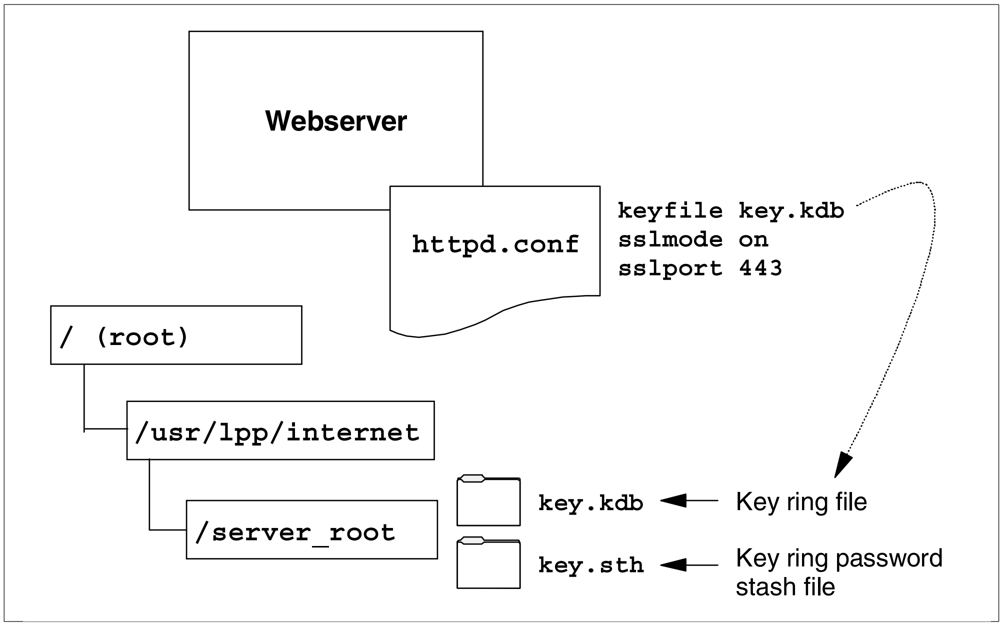
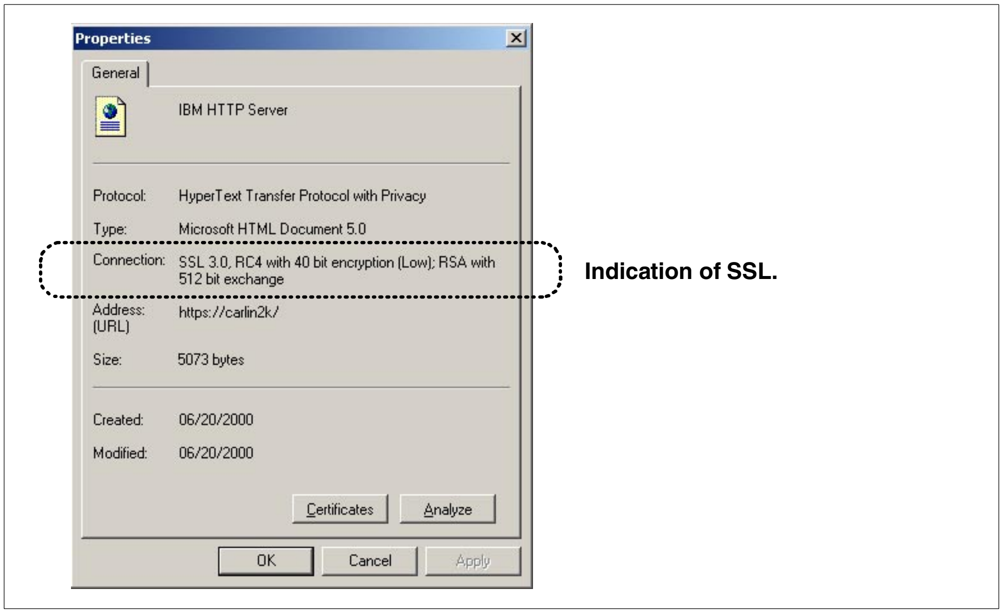


Figure 35. The Webserver's SSL keyring file locations

Note: The keyring file name doesn't have to be "key.kdb," nor does it have to reside in the `/server_root` directory of the Webserver. But these are the default settings and it illustrates the basic concept.

If your system has these directives defined in httpd.conf and the keyring files are on the system, your system *may* already be configured for SSL. Connect to the home page of your Webserver, then place an "s" at the end of "http" (to form `https://` ...) and re-issue the URL. If your Webserver is configured to support SSL, you should be able to see the session go to SSL. On Netscape you can see that by observing the little "lock" symbol in the lower-left corner of the screen. For Internet Explorer, you can right-click on the browser screen and then select the **Properties** option from the list. If you're in SSL, you'll see what's illustrated in Figure 36 on page 56.



Properties

General

IBM HTTP Server

Protocol: HyperText Transfer Protocol with Privacy

Type: Microsoft HTML Document 5.0

Connection: SSL 3.0, RC4 with 40 bit encryption (Low); RSA with 512 bit exchange

Address: (URL) https://carlin2k/

Size: 5073 bytes

Created: 06/20/2000

Modified: 06/20/2000

Figure 36. How to see if SSL is invoked on Internet Explorer browser

If SSL is not configured on your system, then you *must* configure it since WCSuite requires it. Refer to A.1, “Problems invoking SSL on Webserver” on page 183.

2.4.17 Restart the Webserver to pick up the changes to httpd.conf

The CMNCONF program will have written some Pass and Service directives to the /etc/httpd conf file, and for those changes to take effect you need to recycle the Webserver.

If the Webserver is already running, stop it with the command:

```
p imwebsrv
```

and then restart it with the command:

```
s imwebsrv
```

If it starts correctly, you should see the following console messages:

```
IMW3536I SA 16777275 0.0.0.0:80 * * READY
```

2.4.18 Start the WebSphere Commerce Suite server

Note

It is critical that DB2 be started prior to attempting to start WCSuite. If DB2 is not started, WCSuite will fail to start with the symptom described in A.3.1, “db2_load_caf failed return_code 8, reason_code f30002” on page 194.

Start the WCSuite server by issuing the command:

```
s cmnmserv
```

at the system console. If it starts correctly, you should see the following console messages:

```
CMN0411I THE Commerce Suite SERVER IS READY FOR CONNECTIONS 272
```

If you encounter problems starting WCSuite, consult A.3, “Problems starting the WCSuite server” on page 194 to see if the problem you are experiencing is explained.

2.4.19 Verify the basic operation of WebSphere Commerce Suite

Verifying the “basic operation” of WCSuite involves making sure the essential components are in place. To do this, we’ll have you exercise a few WCSuite commands against the instance you just created, and in doing so you’ll verify things are working properly. Figure 37 illustrates what we’re looking to validate.

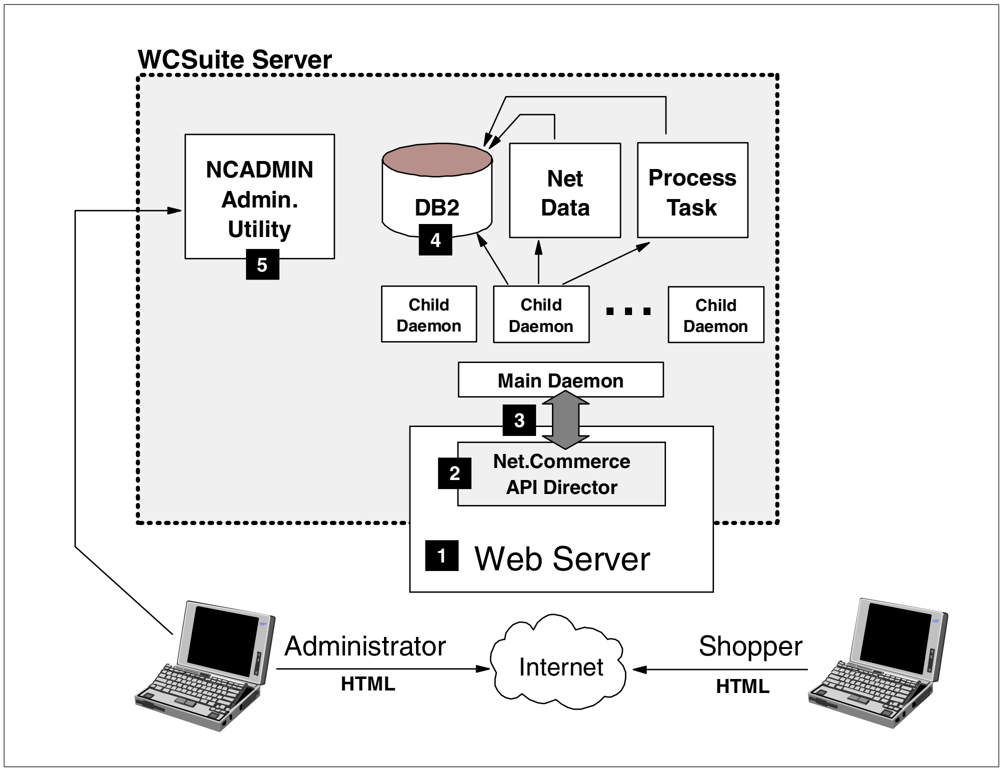


Figure 37. Verifying the basic operation of a newly configured WCSuite instance

Let’s review what the objectives of this validation phase are. The actual commands used to verify this will come after.

1. The CMNCONF program will have written some directives to the Webserver’s httpd.conf file. We need to verify that they are taking effect. Being able to get to the Demomall’s “home page” will mean the Pass directives mapping the Demomall URL have taken effect.

   We must also verify that SSL for the Webserver has been correctly configured. By hardcoding an “s” on the `http://` portion of the URL, we can test the SSL capabilities of the Webserver.

2. If the URL has a WCSuite command in it, then it will be necessary for the Webserver to pass the request over to the WCSuite Director code running in the Webserver’s address space. We need to verify that the Director is running

and that the passing of the URL is handled appropriately. Being able to successfully execute any WCSuite command will verify this.

3. The TCP ports across which the Director talks to the Main Daemon need to be checked. Successfully executing any WCSuite command will verify the connection between the Director and Main Daemon is good.
4. Getting to the DB2 database is the key to all of this. If you can successfully execute a WCSuite command that you know gets information from the database, then you'll have verified that the connection to the database is good, and that the operation of the child processes, process tasks and Net.Data tasks all worked well.
5. Finally, exercising the administrative function of WCSuite accomplishes two things: it verifies that the administrative commands are working, and it validates the authentication routine of WCSuite. You can't access this function without logging on to WCSuite, and if you can log on, then it validates the authentication process.

2.4.20 Checklist to validate basic WCSuite functionality

The following is a list of specific URLs to issue from a browser to validate the basic functionality of the WCSuite server as illustrated in Figure 37 on page 57.

Note 1: For each of these URLs, `<host>` is the host name you provided your instance on the "System Configuration Panel 1 of 2" screen of CMNCONF (see 2.4.12.4, "CMNCONF System Configuration option" on page 44 for a description of that process).

Note 2: Use Netscape for this test. Internet Explorer will work for most of these tests, but Netscape is the supported browser for use with the Administrative function of WebSphere Commerce Suite.

Note 3: If you encounter problems running these tests, consult A.4, "Problems accessing the Metropolitan Demomall" on page 195.

```
http://<host>/demomall/basemall.htm
```

If you receive the "Metropolitan Demomall" home page, this validates that the `Pass` directives in the /etc/httpd.conf file added by the CMNCONF program have been picked up by the Webserver and are active.

```
https://<host>/demomall/basemall.htm
```

Note the "s" on the https:// portion of this URL. That tells the Webserver to invoke SSL when issuing this HTML page. Watch the browser to insure SSL is invoked. On Netscape this is a small lock symbol in the lower left. See Figure 36 on page 56 for an illutration of what to look for on Internet Explorer.

See 2.4.16, "Enabling SSL on your Webserver" on page 54 for information on that process.

```
http://<host>/webapp/commerce/command/ExecMacro/mall_dir.d2w/report
```

This command will execute the `mall_dir.d2w` Net.Data macro and will query the database for the stores located in the Metropolitan Demomall. If you get the "Mall Directory," that means the Director code got the command, passed it across the local TCP port to the main daemon, Net.Data was invoked and it successfully connected to the database and received results from its query.

```
http://<host>/webapp/commerce/command/InterestItemDisplay
```

This command will display the contents of your shopping cart. Your shopping cart will be empty at this point in time, but that won't hinder the objective of this test. This command is, by default, designated as one for which WCSuite will automatically invoke SSL. Watch to insure SSL is indeed invoked. This command also results in a "Process Task" being run, so it validates that function as well.

```
http://<host>/ncadmin/
```

This command will bring up the WebSphere Commerce Suite's administrative function logon panel. If you are successful in logging onto the facility, then it validates the authentication routines of WCSuite.

The default userid and password for the administrative facility are:

Userid	`ncadmin`
Password	`ncadmin`

On your first invocation of the NCADMIN administrative utility you will be forced to change the initial password.

2.4.21 Review: where you are after running these validation tests

If you successfully ran all those tests on the first try, then you are indeed a skilled technician! Usually there's something that goes wrong and needs fixing. (We tried to capture the common things in A.4, "Problems accessing the Metropolitan Demomall" on page 195.)

Getting past this checkpoint means you may now turn your focus to the configuration issues related to the new functions of WCSuite V4.1: JSP support and personalization.

2.5 Phase 2: Proxy server configuration and validation

The purpose of this phase is to configure a second Webserver to run the WebSphere Application Server (WAS) plugin. This second Webserver is where the servlets supplied with WCSuite V4.1 will run. These servlets provide the new functions of the Java/JSP pages and the publishing capability from Commerce Studio.

This arrangement is illustrated in Figure 38 on page 60.

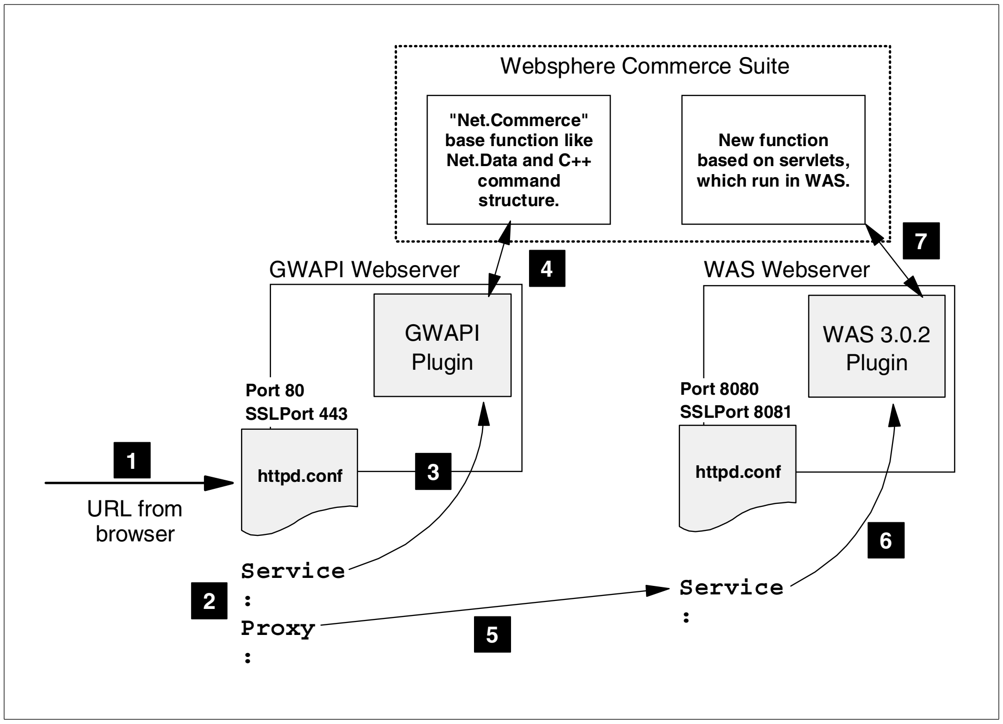


Figure 38. Proxy server configuration overview

We'll explain what all the flows mean in a moment. But first, let's answer this question: "Why in the world do we need to configure this?"

It turns out that DB2 for OS/390 restricts the mixing of JDBC and other connection types (notably CAF) from within the same OS/390 address space. The WAS 3.0.2. plugin uses the RRS Attachment Facility (RRSAF) to connect to DB2. The WCSuite GWAPI plugin uses RRSAF as well, but the recommendation is to avoid mixing plugins from the same Webserver address space. To avoid this mixing, the WCSuite GWAPI plugin resides in its Webserver, and the WAS plugin resides in another Webserver.

That being said, let's turn to the diagram shown in Figure 38 and explain each of the numbered references.

1. A URL comes in from the shopper's browser. That URL will have one of three formats:

   ```
   http://<host>/static_page.html
   http://<host>/webapp/commerce/command/ExecMacro ...
   http://<host>/webapp/commerce/servlet/ProductDisplay ...
   ```

   The first URL shown is one for a static resource, like an HTML page or JPG/GIF image file. That request will be served directly by the Webserver without WCSuite being involved. We won't focus on that type of URL.

   The second and third URLs show requests for WCSuite commands to be executed. Notice how the two URLs are different: the second one has the string "command" in it, the third has the string "servlet."

   The directives in the httpd.conf will determine what to do with the inbound URLs based on this string difference. That's discussed next.

2. There are two directives in the httpd.conf file that we will focus on here:

   - **Service**: An inbound URL that matches the value specified on a Service directive will be routed for execution to the plugin (a program running in the Webserver's address space and using the defined API of the Webserver) named on the directive.

     For WCSuite commands that conform to the "old" Net.Commerce-like style (in other words, commands that are *not* Java-based), the Service directive in the httpd.conf file looks for URLs with the string `commerce/command/*`. Any URL with that string will match and get routed to the WCSuite plugin, which is also known as the "Director code" or the "GWAPI code."

   - **Proxy**: An inbound URL that matches the value specified on a Proxy directive will be routed to the host specified on the Proxy directive. For WCSuite commands based on the newer Java architecture, the Proxy directive looks for the string `commerce/servlet/*`. Those requests get routed to the other Webserver.

3. This flow illustrates a received URL that mapped to the Service directive, and thus is sent to the WCSuite GWAPI plugin.

4. This flow illustrates the interaction between the GWAPI plugin and the WCSuite server code. This is how the bulk of the WCSuite commands are handled.

5. A URL received that mapped to the Proxy directive gets routed over to the other Webserver.

6. The httpd.conf file of this second Webserver has its own Service directives. These Service directives are looking for the `commerce/servlet/*` string. When it finds one, it routes the request to the WAS 3.0.2 plugin for execution.

7. The WAS plugin has knowledge of the location of the WCSuite Java classes that contain the new Java-based commands and functions. The Java commands, servlets and JSPs will run under the control of WAS.

2.5.1 Create proxy configuration in GWAPI Webserver

If you successfully validated the basic functions of WCSuite as detailed in 2.4.20, "Checklist to validate basic WCSuite functionality" on page 58, then you have the GWAPI Webserver functioning. However, the `httpd.conf` file of that Webserver is not yet configured to be a proxy server. Do the following:

1. Edit the `httpd.conf` file for the Webserver that has the GWAPI plugin configured and was used to validate the basic functions of WCSuite.

2. Add the following lines to the `httpd.conf` file. They must come *after* the block of updates made by CMNCONF at the top of the file (delimited by `###### IBM WebSphere Commerce Suite ######`) and *before* the `Pass /*` directive located approximately half-way down the file:

```
Proxy /redbook/verify/* http://<WAS server host name>:8080/*
Proxy /webapp/*jsp* http://<WAS server host name>:8080/*jsp*
Proxy /webapp/commerce/servlet/* http://<WAS server host name>:8080/webapp/commerce/servlet/*
Proxy /webapp/commerce/MerchantAdmin http://<WAS server host name>:8080/webapp/commerce/MerchantAdmin
Proxy /webapp/examples/* http://<WAS server host name>:8080/webapp/examples/*
Proxy *:443
```

Where `<WAS server host name>` is the host name of the Webserver that will be running the WAS plugin. It may be the exact same host name as your GWAPI plugin Webserver, or a different one.

This provides the GWAPI Webserver the ability to recognize an inbound URL and route it over to the WAS Webserver.

The first `Proxy` statement will be used to verify the proxy configuration.

3. Edit the javelin.conf file for the GWAPI Webserver (this file will be found in the same directory as the httpd.conf file). Check the value for the `PureProxy` directive: it should be set to `off`. If the directive doesn't exist, then add it; if the directive is set to `on`, then change it to `off`.
4. Stop and restart the GWAPI Webserver so the changes you have made will be picked up.

2.5.2 Create second Webserver to run WAS plugin

The IBM HTTP Server (we have been calling it the Webserver throughout this redbook) is designed to allow multiple copies to run in the same OS/390 system. The process of setting up a second running copy of the Webserver is something beyond the scope of this book, so we will leave it to you to do that job.

The key updates to the httpd.conf file of this second Webserver (which we'll call the "WAS Webserver") are as follows:

```
Userid PUBLIC
```

The PUBLIC Userid is used so that incoming Web requests do not need to require the user to log on.

```
Port 8080
sslport 8081
```

This defines the non-SSL listen port as 8080 and the SSL port as 8081. This is different from the GWAPI Webserver, which is running with non-SSL of 80 and SSL of 443.

Start this second Webserver and make certain it comes up properly. You can verify this by issuing the following URL from a browser:

```
http://<host name of WAS Webserver>:8080
```

2.5.3 Verify the proxy configuration

We are going to verify the proxy configuration by issuing a URL to the GWAPI Webserver for an HTML file we know is only over on the WAS Webserver. If the HTML page is displayed, it means the URL was routed over and that the proxy configuration works.

Do the following:

1. Browse the `httpd.conf` file of the WAS Webserver, and find the `Pass /*` directive which defines the "document root" of the this Webserver. It will have a format that looks something like this:

   ```
   Pass /* /usr/lpp/internet/pubs/*
   ```

Note the directory referenced on the Pass directive. That is where you are going to create the simple HTML file you will use to verify the proxy configuration is working.

2. Create and save a file in the document root of the WAS Webserver called `redbook.html`. Insert the following text:

```
<HTML>
<BODY>
<H2>You made it over to the WAS Webserver!</H2>
</BODY>
</HTML>
```

3. Start a browser and point it to your GWAPI Webserver (the one on which you verified the basic function of WCSuite, which is the one listening on Port 80). Issue the following URL:

```
http://<GWAPI Webserver host name>/redbook/verify/redbook.html
```

If things work properly, the `Proxy /redbook/verify/*` statement in the httpd.conf file should route this over to the WAS Webserver listening on Port 8080. You should see the “You made it over to the WAS Webserver!” message.

Let’s review where we are. Figure 39 illustrates the configuration at this point in time.

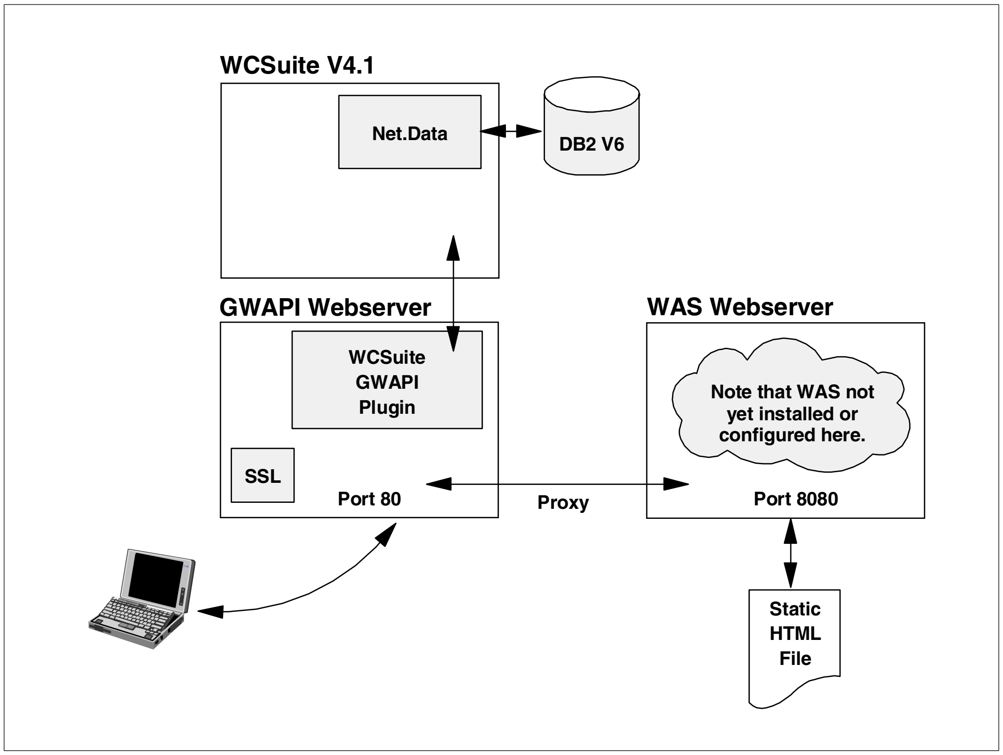


Figure 39. Checkpoint picture of configuration after verifying proxy configuration

And this is what you’ve verified:

- The GWAPI Webserver accepts inbound requests and knows about the updates to its `httpd.conf` made by the CMNCONF utility (display of `demomall/basemall.htm file` proved that).

- The SSL function on the GWAPI Webserver works properly (changing the `http` to `https` verified that).
- The WCSuite started task comes up properly, and the connection between the GWAPI plugin and the WCSuite server instance operates properly (tests from 2.4.20, “Checklist to validate basic WCSuite functionality” on page 58 proved this).
- The second Webserver (the “WAS Webserver”) works properly for serving static HTML pages (pointing browser to Port 8080 showed that).
- The proxy configuration of the GWAPI Webserver correctly routed the request over the WAS Webserver (the simple `redbook.html` file proved that).

Here is what you have left to do in this phase of the installation/verification process:

- Install the WebSphere Application Server (WAS) and configure its use as a plugin to the WAS Webserver.
- Verify the WAS configuration by exercising the sample servlets provided with WAS.

2.5.4 Install and configure WAS plugin in the WAS Webserver

Back in 2.5.2, “Create second Webserver to run WAS plugin” on page 62 you established the second Webserver that will act as the Webserver on which the WAS plugin will run. It is now time to install and configure WAS on that Webserver.

To give you a picture of how some of these configuration files relate to one another, consider the picture provided in Figure 40 on page 65.

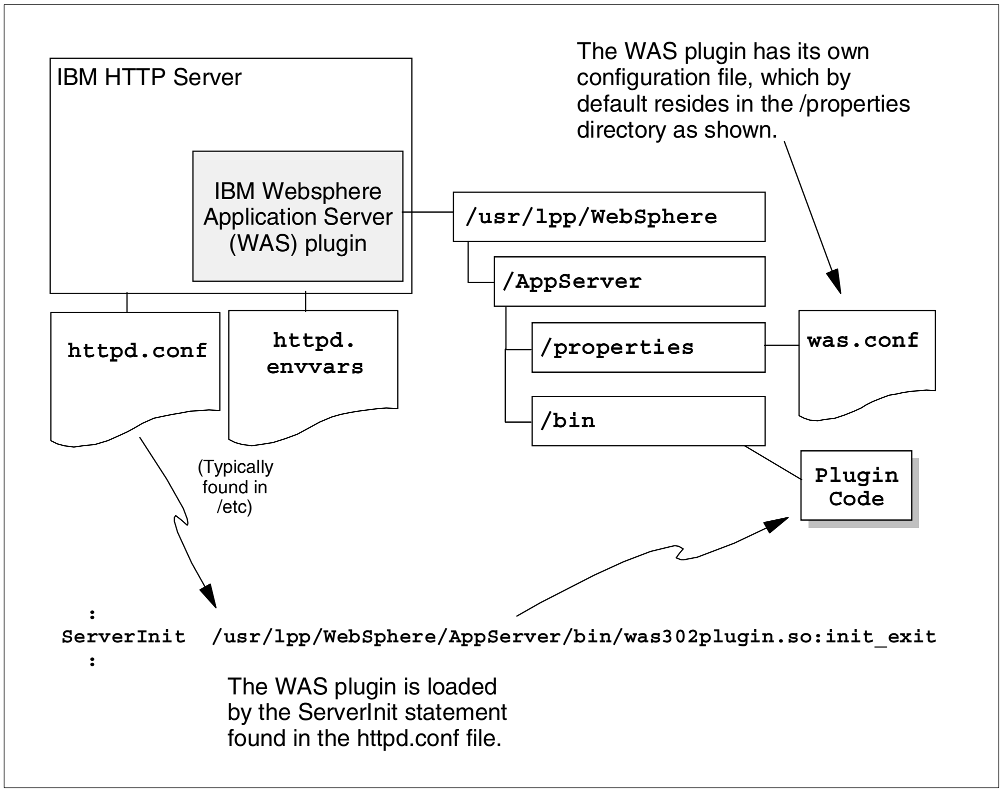

The WAS plugin has its own configuration file, which by default resides in the /properties directory as shown.
IBM HTTP Server
IBM Websphere Application Server (WAS) plugin
/usr/lpp/WebSphere
/AppServer
/properties
/bin
was.conf
Plugin Code
httpd.conf
httpd. envvars
(Typically found in /etc)
:
ServerInit /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:init_exit
:
The WAS plugin is loaded by the ServerInit statement found in the httpd.conf file.

Figure 40. Relationship of WAS to the HTTP Server and their configuration files

The installation of WAS will place the code into the /usr/lpp/WebSphere/AppServer directory (or wherever you choose to install it). The configuration process involves updating several different files shown in the picture. We wanted to give you this picture so you could visually see the relationship of the Webserver's configuration files and the WAS configuration files. In addition, we show you what tells the Webserver to load the WAS plugin into its address space.

We'll introduce even more configuration files as you work through this document.

To configure the WAS plugin, do the following:

1. Using the *WebSphere Application Server Standard Edition Planning, Installing, and Using Manual* (GC34-4806), install the WAS code onto your system. Typically this is installed into `/usr/lpp/WebSphere`.
2. Manually create the following subdirectories under the `/usr/lpp/WebSphere/AppServer` directory:
   - logs
   - work
3. Add the program control bit by issuing the following command on all the files in the /usr/lpp/Websphere/AppServer/bin directory:

   ```
   extrattr +p *.so
   ```

   You can verify that the program control bit is set by issuing the command:

   ```
   ls -E *.so
   ```

from the /usr/lpp/WebSphere/AppServer/bin directory. You should see output that looks something like this:

```
-rwxr-xr-x  -ps  2 OMVSKERN SYS1      655360 Jun 26 12:44 libascommon.so
-rwxr-xr-x  -ps  2 OMVSKERN SYS1     1097728 Jun 26 12:44 libasin.so
-rwxr-xr-x  -ps  3 OMVSKERN SYS1      589824 Jun 26 12:46 libstubj.so
-rwxr-xr-x  -ps  3 OMVSKERN SYS1      589824 Jun 26 12:46 libstubj_g.so
-rwxr-xr-x  -ps  2 OMVSKERN SYS1      745472 Jun 26 12:45 was302plugin.so
```

It's the "p" in the "`-ps`" string that indicates the program control bit is on.

4. Update the was.conf file that is located in the /usr/lpp/WebSphere/AppServer/properties directory. There are five "properties" that you need to update:

   - **appserver.logdirectory=**

     Set this to: `/usr/lpp/WebSphere/AppServer/logs`

   - **appserver.workingdirectory=**

     Set this to: `/usr/lpp/WebSphere/AppServer/work`

   - **appserver.jvmpropertiesfile=**

     Set this to:
     `/usr/lpp/WebSphere/AppServer/properties/default_global.properties`

   - **appserver.classpath=**

     This property is used by WAS to know what classes to load at initialization time. To validate the basic operation of the WAS servlet support, you may leave this blank. The WCSuite *Configuration Guide* suggests:

     ```
     /usr/lpp/WebSphere/AppServer/properties/default_mimetype.properties
     ```

     For now it doesn't seem to matter, so provide what the *Configuration Guide* suggests. Later, when we configure this WAS plugin to run the WCSuite servlets, we will add quite a bit more to this classpath.

   - **appserver.loglevel=**

     Set this to `Warning`. This will log activity to the directory specified on the `appserver.logdirectory` property.

5. Now go to the http.conf file in use by the WAS Webserver (*not* the GWAPI Webserver), edit and add the following directives. Again, add these *before* the `Pass /*` directive found lower in the file:

```
ServerInit  /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:init_exit <WAS home>,<WAS conf>
Service     /*.jhtml   /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
Service     /*.shtml   /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
Service     /*.jsp     /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
Service     /servlet/* /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
Service     /webapp/*  /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
Service     /commerce/*      /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:service_exit
ServerTerm  /usr/lpp/WebSphere/AppServer/bin/was302plugin.so:term_exit
```

   Where:

   `<`**`WAS home`**`>` is the directory in which you installed the WAS code. Typically this is /usr/lpp/WebSphere.

   `<`**`WAS conf`**`>` is the directory and name of the file that will serve as the configuration file for the WAS plugin. By default this file is /usr/lpp/WebSphere/AppServer/properties/was.conf. You could place this file somewhere else. If you're running multiple copies of WAS on your system, then you probably will locate this file in its own directory. For example, on the system upon which we tested WCSuite to write this redbook, our file was located at /web/comm2/was.conf.

This block of directives serves two purposes:

- The `ServerInit` statement is read by the Webserver and used to start the WAS plugin at the time the Webserver is initialized. The ServerTerm does the opposite: it stops the WAS plugin when the Webserver is being brought down.
- The `Service` directives are used to map inbound URL strings to the WAS plugin. Take for example a URL format of a WCSuite command that exercises the new Java-based function:

  ```
  http://<host>/webapp/commerce/servlet/ProductDisplay?prrfnbr=12345 ...
  ```

  The Webserver would see this, parse through its directives, and find a match with the `Service /webapp/*` directive. This in turn would invoke the WAS plugin and execute the command, which is implemented as a Java servlet run under WAS.

6. Determine the home directory locations of the following components. We have included here the typical locations, but your system may be different.

Table 5. Home directory locations of key system components

Home	Your Location	Typical
Java		/usr/lpp/java/J1.1
HTTP Server		/usr/lpp/internet
JDBC		/usr/lpp/db2/db2610
WCSuite		/usr/lpp/CommerceSuite

In the next step we'll refer to these as `<Java Home>`, `<HTTP Home>`, `<JDBC Home>` and `<WCSuite Home>`. You will substitute your actual values.

7. Turn your attention to the httpd.**envvars** file (not the "conf" file, but the "envvars" file). This is the environment variables file for the WAS plugin Webserver, and it needs to be updated to reflect the system environment in which it will run. Do the following:

Note

The information on any single environment variable must be typed out on one line, even if that means scrolling to the right. *Do not* break a line in the envvars file. Many envvar files have lines that string out several hundred characters to the right of the screen margin.

- To the **PATH** variable, add:

  ```
  <Java Home>/bin:<JDBC Home>/bin
  ```

- To the **LIBPATH** variable, add:

  ```
  <Java Home>/lib/mvs/native_threads:<JDBC Home>/lib
  ```

- Set the **JAVA_HOME** variable to your `<Java Home>` directory

- To the `CLASSPATH` variable, add (again, on one line— it is broken here just to so it would fit on the page):

```
<Java Home>/lib/classes.zip:
<HTTP Home>/server_root/CAServlet:
<JDBC Home>/classes/db2jdbcclasses.zip:
<WCSuite Home>/www/Servlets/Public
```

2.5.5 Validate the WAS plugin

The picture of your environment should now be what we show in Figure 41.

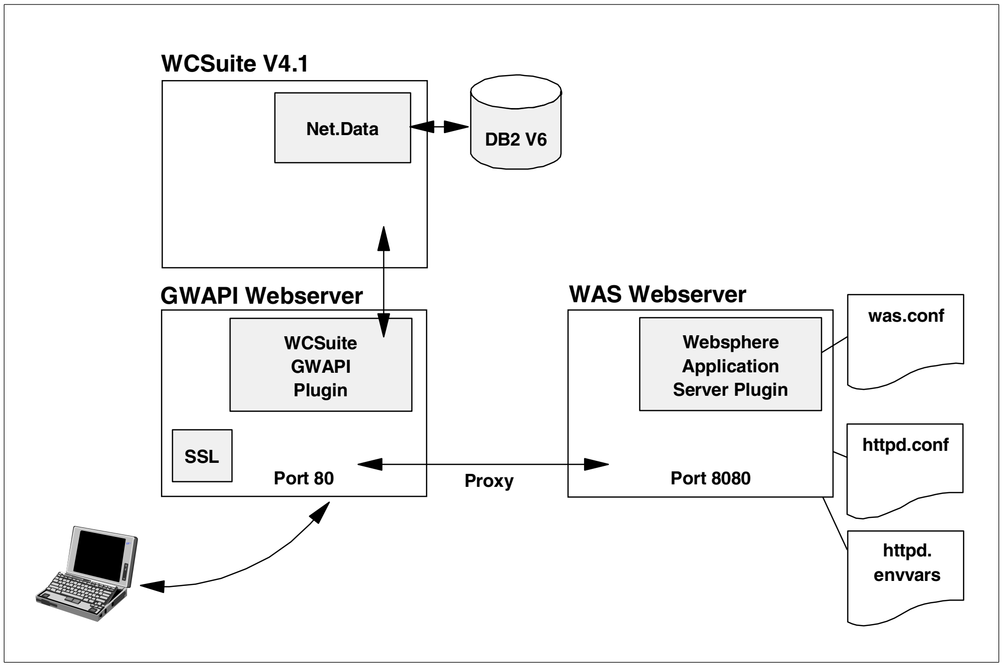


Figure 41. Environment after WAS plugin configured into WAS Webserver

To verify that the WAS plugin is working properly, do the following:

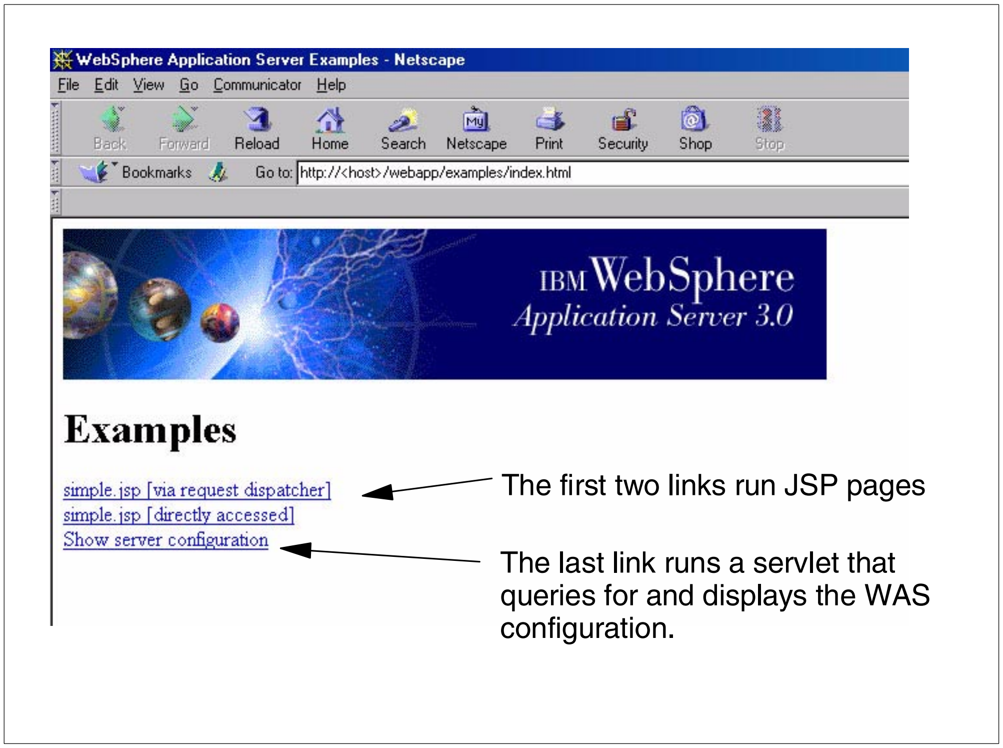
1. Stop and restart the WAS Webserver so the changes you have made to the httpd.conf, httpd.envvars and was.conf file can be picked up.
2. Make sure the GWAPI Webserver is also up and running. This will provide the proxy routing of the request over to the WAS Webserver listening on port 8080.
3. From a browser, issue the following URL:

   `http://<host>/webapp/examples/index.html`

   This will go to the GWAPI Webserver first, but the `Proxy` statements you placed in the httpd.conf of this server will route the request over to the WAS Webserver.

   The WAS Webserver’s `Service /webapp/examples/*` directive will map this request to the WAS plugin.

   The WAS examples page should display the screen shown in Figure 42 on page 69.


WebSphere Application Server Examples - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Go to: http://<host>/webapp/examples/index.html
IBM WebSphere
Application Server 3.0
Examples
simple.jsp [via request dispatcher]
simple.jsp [directly accessed]
Show server configuration

The first two links run JSP pages

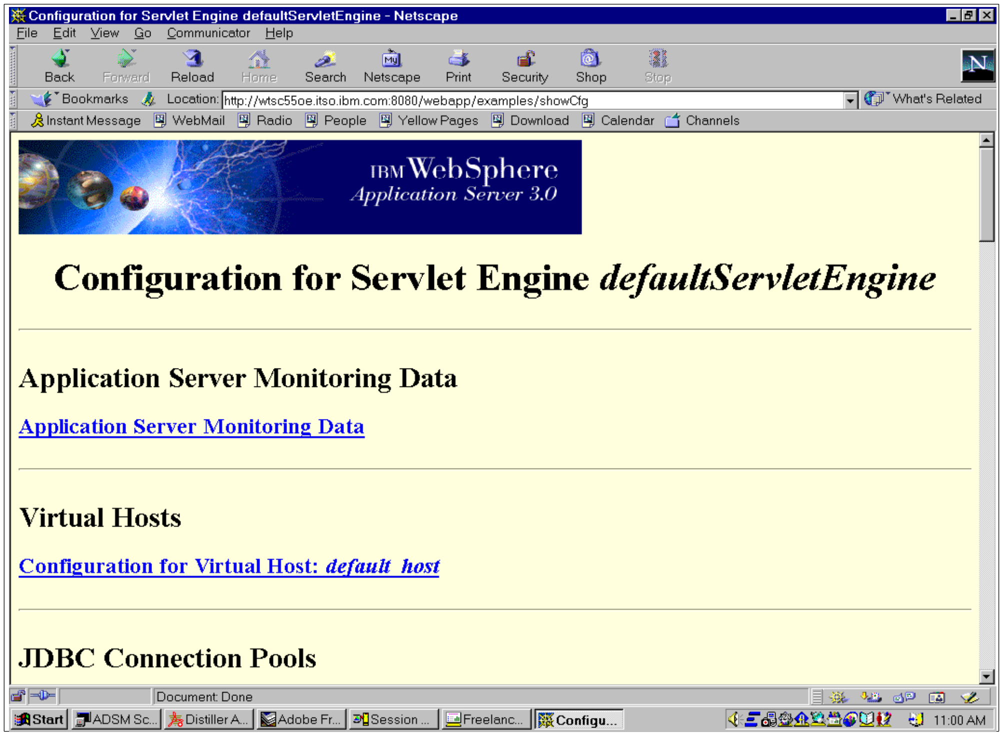
The last link runs a servlet that queries for and displays the WAS configuration.

Figure 42. The WAS examples HTML page

This is a static HTML page, so do not think that your WAS configuration has been fully tested yet. All this tested was the Service directive in httpd.conf.

4. Click on the link that says “Show server configuration.” This will run a servlet called `showCfg`. If that runs properly, then it validates the ability of the WAS plugin to locate its supplied sample servlet and run it.

   The result should look like that shown in Figure 43 on page 70.


Configuration for Servlet Engine defaultServletEngine - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://wtsc55oe.itso.ibm.com:8080/webapp/examples/showCfg
What's Related
Instant Message WebMail Radio People Yellow Pages Download Calendar Channels
IBM WebSphere Application Server 3.0
Document: Done
Start ADSM Sc... Distiller A... Adobe Fr... Session ... Freelanc... Configu...
11:00 AM

Configuration for Servlet Engine *defaultServletEngine*

Application Server Monitoring Data

Application Server Monitoring Data

Virtual Hosts

Configuration for Virtual Host: *default_host*

JDBC Connection Pools

Figure 43. WAS “showCfg” servlet output

2.5.6 End of phase 2 review

In Phase 1 you configured and validated the basic function of the WCSuite product. That validated the GWAPI Webserver’s configuration, and the ability of the WCSuite GWAPI plugin to communicate with the WCSuite server. The WCSuite server was able to communicate with the DB2 subsystem, and the basic functions of the WCSuite product were validated.

In Phase 2 you enabled the GWAPI Webserver to be a proxy server and route certain requests over to a second HTTP Server, which we are calling the WAS Webserver. The proxy configuration was validated by accessing a static HTML page known to exist only on the WAS Webserver, but not on the GWAPI Webserver you pointed your browser at.

Also in Phase 2 you installed and configured the WAS plugin to the HTTP Server, and by running the sample servlet `showCfg`, you validated the WAS plugin’s operation.

Here is what you must now do:

1. Configure a JDBC Type 2 connection for DB2.
2. Modify the WAS Webserver’s configuration to recognize WCSuite components.
3. Validate the ability to run the supplied sample JSPs that come with the WCSuite Demomall. Doing so will validate the servlet functionality of WCSuite, and position you to move to Phase 4.

2.6 Phase 3: JDBC configuration and WCSuite servlet validation

2.6.1 Update the environment variable and configuration files

There are several files on your system that need to be updated. Figure 44 shows those files.

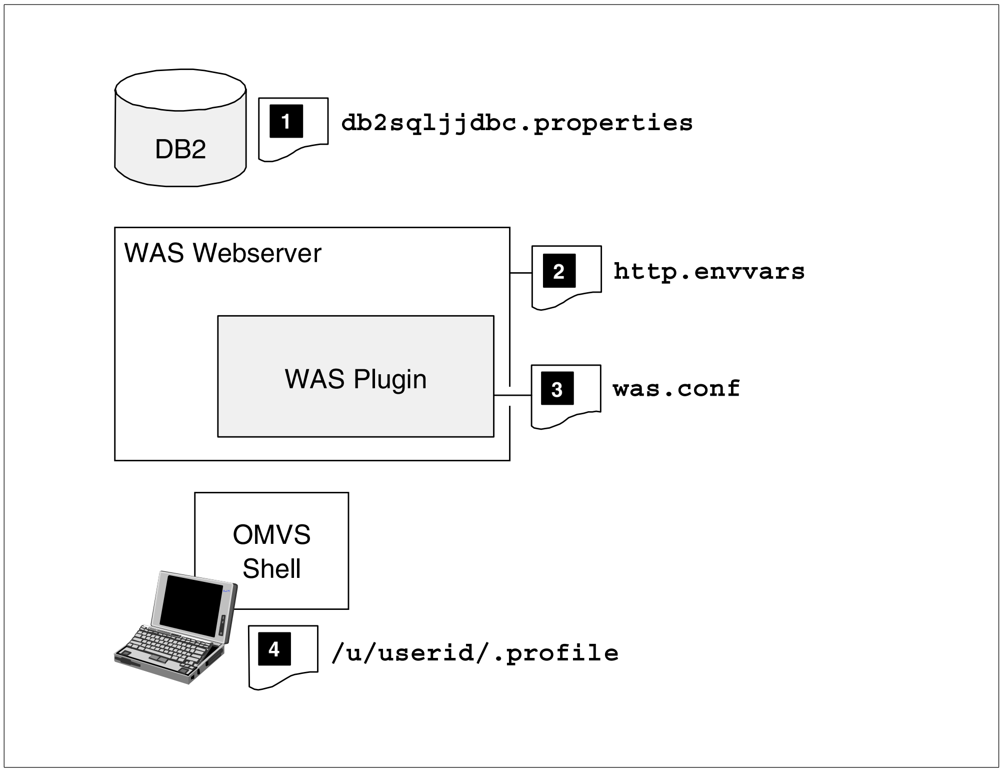


Figure 44. Environment variable, configuration and properties files updated for JDBC configuratio

The updates to these files are described next.

1. **db2sqljjdbc.properties**
   This file is typically located in the /usr/lpp/db2/db2610/classes directory. Whether you wish to change the default file or copy the file to a user directory (such as /u/cmnsrv/configs in our example here in this redbook) is up to you.

   For the WCSuite environment, two processes make use of this file:

   - Your OMVS shell environment will need to know where this file is when you run the `db2genJDBC` utility (that process will be explained later, in 2.6.2, “Bind the DBRM plans” on page 73). The db2sqljjdbc.properties file has a pointer to the PDS data set that will house the output from `db2genJDBC`.
   - The WAS plugin has a pointer in its configuration file (was.conf) to the db2sqljjdbc.properties file so WAS can understand the JDBC properties it is to work with.

   Edit the copy of db2sqljjdbc.properties that you will use (the default copy in the DB2 directory, or one that you have copied to another directory) and make sure it has the following directives:

   ```
   DB2SQJSSID=<your DB2 subsystem ID>
   DB2SQLJDBRMLIB=CMN.DBRMLIB.DATA
   ```

```
DB2SQLJATTACHTYPE=RRSAF
DB2SQLMULTICONTEXT=YES
```

A few notes on this:

- The `DB2SQLJSSID` value is pointing to the *Subsystem ID* of your DB2, not the location name, which may be different. Later, in the was.conf file and the ncommerce.conf file, we'll make reference to the *location name*. If the two are the same, then things are easier. If they are different, then you must be careful.
- The `DB2SQLJDBRMLIB` value may point to any MVS PDS. To keep things easy to remember, you should probably have the high level qualifier of the data set the same as the installation HLQ you used when you installed WCSuite.

  This PDS must be allocated *prior* to running the `db2genJDBC` utility. The allocation parameters are provided in 2.6.2, "Bind the DBRM plans" on page 73.

2. **httpd.envvars**
This is the environment variable file for the HTTP Server that is running the WAS plugin, and *not* the HTTP Server that is acting as the proxy server (see Figure 41 on page 68 for a picture of the relationship of these two Webservers.) To verify exactly which file is being used, check the `LEPARM` value of the JCL start procedure used to invoke the server. That will point to the file you should modify.

   Update the LIBPATH, LD_LIBRARY_PATH and PATH variables as follows (while maintaining whatever existing values may already be there):

   LIBPATH

   Provide a pointer to the SQLJ/JDBC library directory, which typically resides in:

   /usr/lpp/db2/db2610/**lib**

   LD_LIBRARY_PATH

   Provide a pointer to the SQLJ/JDBC library directory, which typically resides in:

   /usr/lpp/db2/db2610/**lib**

   PATH

   Provide a pointer to the SQLJ/JDBC *binary* directory (careful: *not* the library directory like LIBPATH and LD_LIBRARY_PATH):

   /usr/lpp/db2/db2610/**bin**

   Provide a DB2SQLJPROPERTIES variable to the file:

   ```
   DB2SQLJPROPERTIES=/u/cmnsrv/configs/db2sqjjdbc.properties
   ```

   Note: This should point to wherever you copied the db2sqljjdbc.properties file back in the first step of this process.

3. **was.conf**
This is the configuration file for the WAS plugin that is running in the WAS Webserver as shown in Figure 41 on page 68. The directive in this file that will be updated is the `appserver.classpath` directive. This directive, like all directives in was.conf, may not be split across lines. So be prepared to have your values stretch well beyond the right side of the screen.

It requires two updates:

- A pointer to the db2sqljclasses.zip file, for example:

```
appserver.classpath= ... /usr/lpp/db2/db2610/classes/db2sqljclasses.zip ...
```

- A pointer to the *directory* in which the db2sqljjdbc.properties file resides. Back in the first step of this process we suggested you might want to copy that file to the directory in which all of the other configuration files resided (in our example, `/u/cmnsrv/configs`). Wherever your copy of that file resides, put a reference to it on the `appserver.classpath` directive:

```
appserver.classpath ... /u/cmnsrv/configs ...
```

4. **`/u/userid/.profile`**
This is the file that establishes the environment when you go into the OMVS shell. The /etc/profile file also provides this function. If your installation maintains a common environment for all users, then place these changes down in /etc/profile. Otherwise, put them in /u/userid/.profile.

WCSuite does *not* use the .profile for your userid, nor do any of the HTTP Servers or the WAS plugin. We are updating this to provide the ability to run the `db2genJDBC` utility from *your* OMVS shell. Update the following:

CLASSPATH

Point to /usr/lpp/db2/db2610/classes/db2sqljclasses.zip

PATH

Point to /usr/lpp/db2/db2610/bin

This provides access to the db2genJDBC utility without having to type out the fully qualified directory path.

DB2SQLJPROPERTIES

Create this variable and set its value to the directory and file name of where you copied the db2sqljjdbc.properties file back in step 1 of this process. In our example we have that set to:

```
DB2SQLJPROPERTIES=/u/cmnsrv/configs/db2sqljjdbc.properties
```

You must export this variable to your environment:

```
EXPORT DB2SQLJPROPERTIES
```

This provides the `db2genJDBC` utility knowledge of the file's location, and it is in that file that the pointer to the MVS PDS data set where the output from `db2genJDBC` utility will be placed is named.

2.6.2 Bind the DBRM plans

The binding of the DBRM plans involves a few steps. Back in 2.4.6, "Bind WCSuite DBRM plan" on page 34 you bound one plan — `mserver` — but that didn't have any JDBC/SQLJ information in it so it is not enough for WCSuite's new function that uses Java. So now you must rebind `mserver` to include the JDBC/SQLJ information, as well as binding a default DB2 plan of `DSNJDBC`.

2.6.2.1 Running the db2genJDBC utility

The first step in the binding process is running a DB2 SQLJ utility called `db2genJDBC`, which resides in the /usr/lpp/db2/db2610/bin directory. Let's start to pull some of the pieces together. Figure 45 on page 74 shows the various files that help you run the db2genJDBC utility, and the output from the `db2genJDBC` utility.

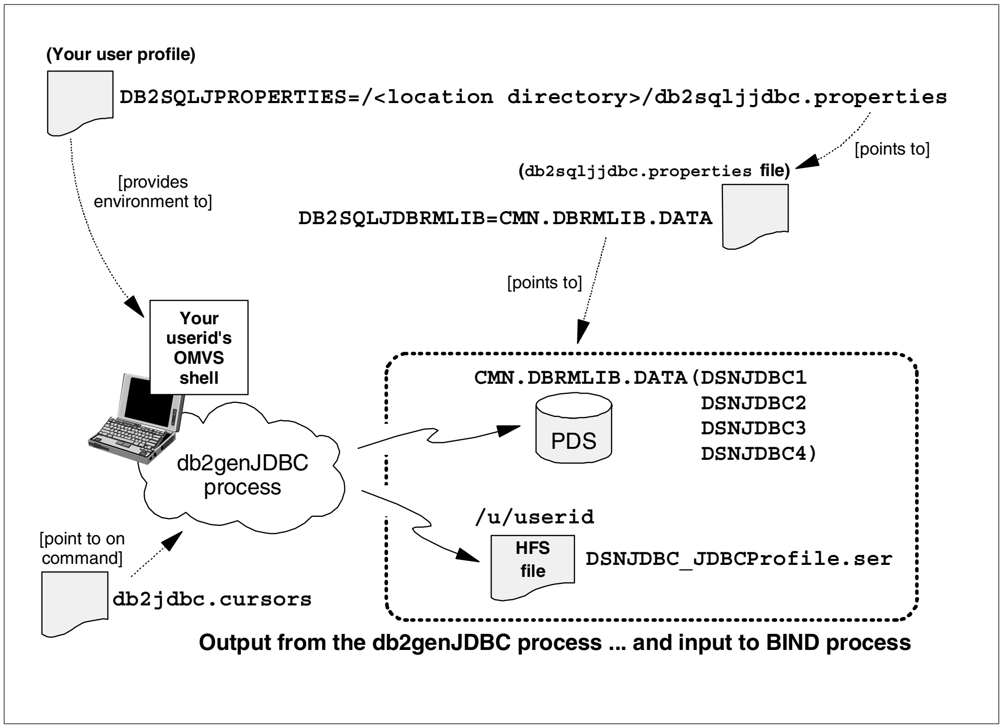

(Your user profile)
DB2SQLJPROPERTIES=/<location directory>/db2sqljjdbc.properties
[points to]
(db2sqljjdbc.properties file)
[provides environment to]
DB2SQLJDBRMLIB=CMN.DBRMLIB.DATA
[points to]
Your userid's OMVS shell
CMN.DBRMLIB.DATA(DSNJDBC1
DSNJDBC2
DSNJDBC3
DSNJDBC4)
PDS
db2genJDBC process
/u/userid
HFS file
DSNJDBC_JDBCProfile.ser
[point to on command]
db2jdbc.cursors
Output from the db2genJDBC process ... and input to BIND process

Figure 45. Environment supporting the running of db2genJDBC utility

You can see why the updates to your user profile and `db2sqljjdbc.properties` come into play. Now, to run the utility, do the following:

1. Allocate the data set CMN.DBRMLIB.DATA as pointed to in the db2sqljjdbc.properties file. (If the data set already exists, then simply skip this step). The allocation parameters are as follows:

   ```
   Space units . . . . . BLOCK
   Primary quantity  . . 100
   Secondary quantity    20
   Directory blocks  . . 10
   Record format . . . . FB
   Record length . . . . 80
   Block size  . . . . . 6080
   Data set name type  : PDS
   ```

2. Locate the file db2jdbc.cursors. It’s typically located in the /usr/lpp/db2/db2610/classes directory. Make a note of its location because you’ll have to point to it on the command line when you invoke `db2sqljjdbc`.

3. Go to the OMVS shell, and change directories to /usr/lpp/db2/db2610 directory (or whatever the `classes` directory of your DB2 installation happens to be), and issue the command:

   ```
   db2genJDBC -cursors=db2jdbc.cursors
   ```

   If the cursors file is located any place other than the current directory, then qualify the file name with the directory path.

If things run properly, the output will look like this:

```
-----------------------------------------------------
-> DB2 6.1: Begin Generation of JDBC Profile
-----------------------------------------------------
-> JDBC program name is:             DSNJDBC
-> Number of JDBC Statements is:     150
-> Number of Callable Statements is: 5
-> Cursor Property file is:          /usr/lpp/db2/db2610/classes/db2jdbc.cursors
-----------------------------------------------------
-> Creating DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC1)'
-> Successfully created DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC1)'
-> Creating DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC2)'
-> Successfully created DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC2)'
-> Creating DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC3)'
-> Successfully created DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC3)'
-> Creating DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC4)'
-> Successfully created DBRM in dataset //'CMN.DBRMLIB.DATA(DSNJDBC4)'

 db2genJDBC SUMMARY INFORMATION
-----------------------------------------------------
-> JDBC Profile DSNJDBC contains:
->  150 Statement sections.
->  5 Callable Statment sections.
->  100 NON HOLD cursors.
->  100 HOLD cursors.
-----------------------------------------------------

 ->INFORMATIVE<-
 ->INFORMATIVE<-    ***************** IMPORTANT *****************
 ->INFORMATIVE<- -> The following DBRMs must be bound as specified into the
 ->INFORMATIVE<- -> packages with the associated Transaction Isolation:
 ->INFORMATIVE<- ->  Bind DBRM in //'CMN.DBRMLIB.DATA(DSNJDBC1)'
 ->INFORMATIVE<- ->  into Package DSNJDBC1 with Transaction Isolation UR
 ->INFORMATIVE<- ->  Bind DBRM in //'CMN.DBRMLIB.DATA(DSNJDBC2)'
 ->INFORMATIVE<- ->  into Package DSNJDBC2 with Transaction Isolation CS
 ->INFORMATIVE<- ->  Bind DBRM in //'CMN.DBRMLIB.DATA(DSNJDBC3)'
 ->INFORMATIVE<- ->  into Package DSNJDBC3 with Transaction Isolation RS
 ->INFORMATIVE<- ->  Bind DBRM in //'CMN.DBRMLIB.DATA(DSNJDBC4)'
 ->INFORMATIVE<- ->  into Package DSNJDBC4 with Transaction Isolation RR
 ->INFORMATIVE<-
 ->INFORMATIVE<- -> These packages must then be bound into the plan to be used for
JDBC applications.
 ->INFORMATIVE<-    ***************** IMPORTANT *****************
 ->INFORMATIVE<-

-----------------------------------------------------
-> DB2 6.1: End   Generation of JDBC Profile
-----------------------------------------------------
```

4. If you look in the directory in which you ran the `db2genJDBC` utility, you'll find a file named:

   DSNJDBC_JDBCProfile.ser.

   Any servlet running under the WAS plugin will need information on the location of this file. The method of telling it where that file exists is with the `appserver.classpath` directive in the was.conf file. Therefore, this file *must* reside in a directory named on appserver.classpath.

   Note: This step is *critically important*, otherwise when you run a servlet that attempts a JDBC connection it'll fail and indicate it can't find the serialized profile..

   So, copy this file to the separate directory we created for storing the WCSuite configuration files (/u/cmnsrv/configs in our example).

5. Edit the was.conf file and update the appserver.classpath directory with the directory in which you just placed the DSNJDBC_JDBCProfile.ser file. In our example that would be /u/cmnsrv/configs.

   You will need to stop and restart the WAS Webserver to pick up the changes to the was.conf file.

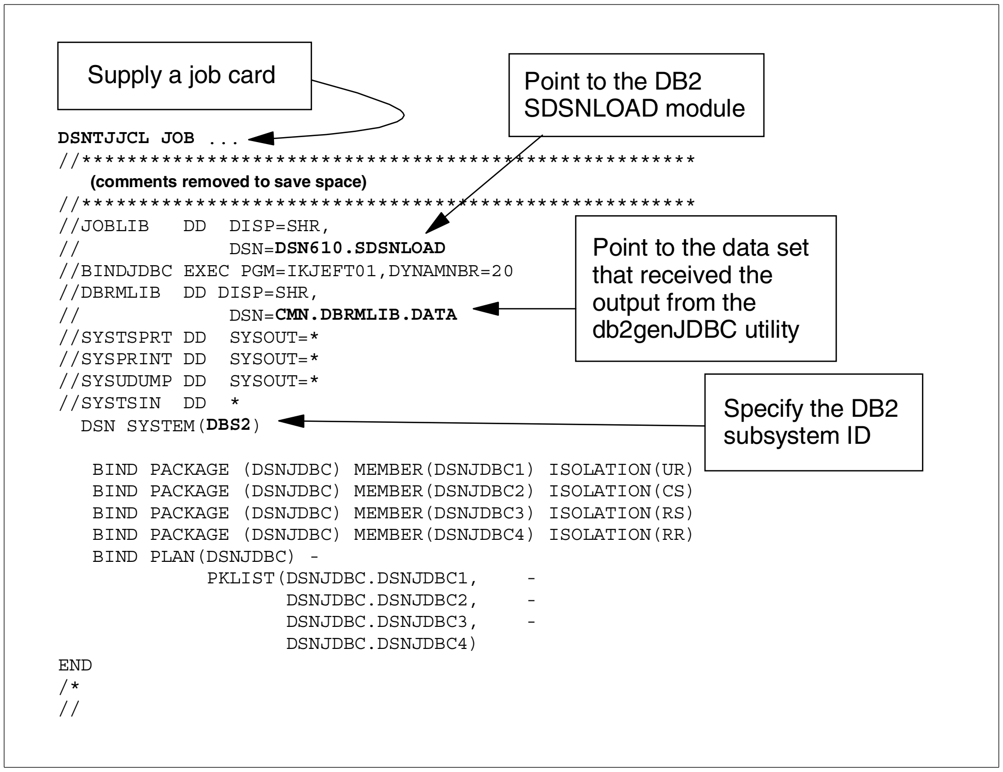
You’re almost there. Now you must bind the plans.

2.6.2.2 Binding the plans

The process of binding the DBRM plan involves running two JCL job streams. One JCL job stream is provided by DB2 (this is the DSNTJJCL member in the data set DSN610.SDSNSAMP), and one is provided by the WCSuite product (this is the file /usr/lpp/CommerceSuite/install/cmnbjdbc.jcl).

The process you should follow is:

1. Copy the member DSN610.SDSNSAMP(DSNTJJCL) to your own JCL library for modification.
2. Make the appropriate modifications. Figure 46 illustrates what needs updating.



Supply a job card

Point to the DB2 SDSNLOAD module

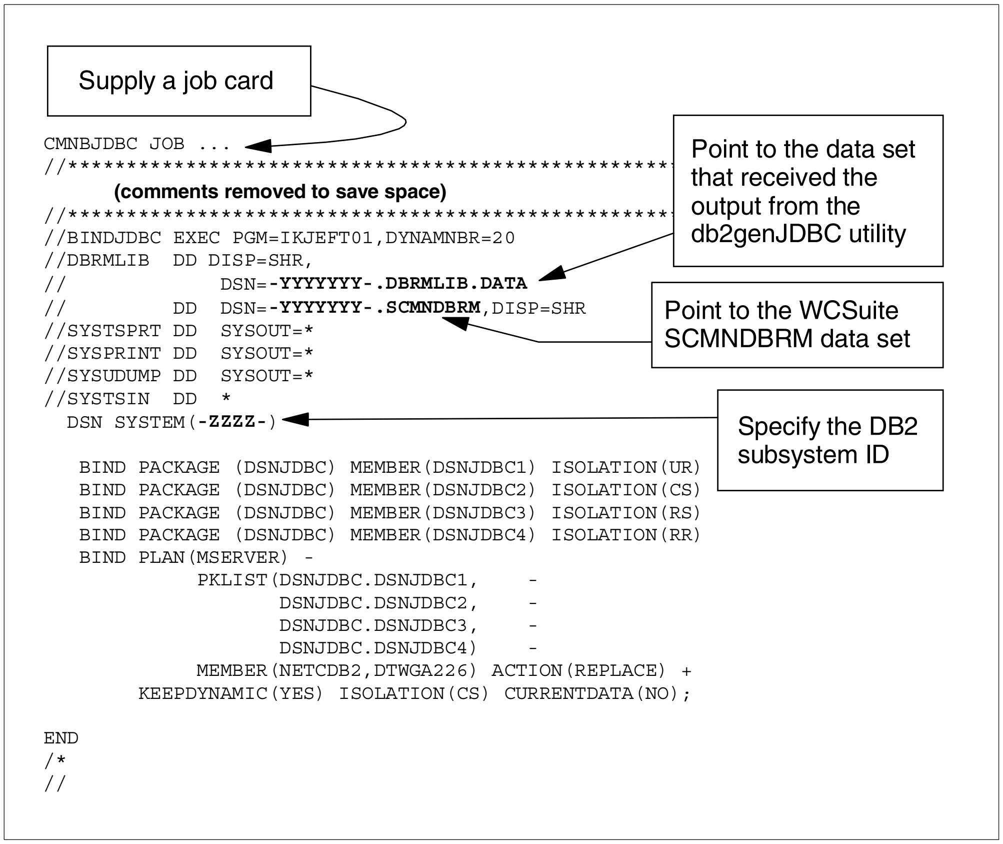
Point to the data set that received the output from the db2genJDBC utility

Specify the DB2 subsystem ID

```
DSNTJJCL JOB ...
//*******************************************************
    (comments removed to save space)
//*******************************************************
//JOBLIB   DD  DISP=SHR,
//             DSN=DSN610.SDSNLOAD
//BINDJDBC EXEC PGM=IKJEFT01,DYNAMNBR=20
//DBRMLIB  DD DISP=SHR,
//             DSN=CMN.DBRMLIB.DATA
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DBS2)

   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC1) ISOLATION(UR)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC2) ISOLATION(CS)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC3) ISOLATION(RS)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC4) ISOLATION(RR)
   BIND PLAN(DSNJDBC) -
             PKLIST(DSNJDBC.DSNJDBC1,     -
                    DSNJDBC.DSNJDBC2,     -
                    DSNJDBC.DSNJDBC3,     -
                    DSNJDBC.DSNJDBC4)
END
/*
//
```

Figure 46. Modifications to supplied DB2 sample BIND job DSNTJJCL

3. Submit the job. This will result in the plan `DSNJDBC` being bound.
4. Copy the HFS file /usr/lpp/CommerceSuite/install/cmnbjdbc.jcl to your JCL library for modification. Copying from an HFS file to a PDS can be accomplished from the ISHELL, or by using the OGET command.
5. Make the appropriate modifications. Figure 47 on page 77 illustrates what modifications are required.



Supply a job card

Point to the data set that received the output from the db2genJDBC utility

Point to the WCSuite SCMNDBRM data set

Specify the DB2 subsystem ID

```
CMNBJDBC JOB ...
//****************************************************
      (comments removed to save space)
//****************************************************
//BINDJDBC EXEC PGM=IKJEFT01,DYNAMNBR=20
//DBRMLIB  DD DISP=SHR,
//             DSN=-YYYYYYY-.DBRMLIB.DATA
//          DD  DSN=-YYYYYYY-.SCMNDBRM,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(-ZZZZ-)

   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC1) ISOLATION(UR)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC2) ISOLATION(CS)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC3) ISOLATION(RS)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC4) ISOLATION(RR)
   BIND PLAN(MSERVER) -
             PKLIST(DSNJDBC.DSNJDBC1,    -
                    DSNJDBC.DSNJDBC2,    -
                    DSNJDBC.DSNJDBC3,    -
                    DSNJDBC.DSNJDBC4)    -
             MEMBER(NETCDB2,DTWGA226) ACTION(REPLACE) +
        KEEPDYNAMIC(YES) ISOLATION(CS) CURRENTDATA(NO);

END
/*
//
```

Figure 47. Modifications to supplied WCSuite sample BIND job cmnbjdbc

6. Submit the job. A return code of 4 is normal. This will result in the plan **MSERVER** being bound.

Now you are ready to verify your JDBC configuration.

2.6.3 Verify the JDBC connection

To verify the JDBC connection, you can compile and run as a servlet some sample Java code we supply in this redbook. This servlet makes a JDBC connection to DB2 and then does a simple query against the MERCHANT database table of the WCSuite Demomall. The results are displayed on your browser.

There are several steps you must take:

1. Update your user profile.
2. Modify the sample servlet code to match your environment.
3. Compile the servlet code.
4. Copy the resulting class file to your WAS Webserver’s servlet directory.
5. Update your was.conf file to recognize the new servlet.
6. Restart the WAS Webserver.
7. Test the servlet.

These steps are discussed in detail in the following sections.

2.6.3.1 Update your user profile

Your user profile (the same .profile you updated back in 2.6.1, “Update the environment variable and configuration files” on page 71) needs to be updated to do two things:

- Provide access to the javac Java compiler, which resides in the `/bin` directory of the Java home.
- Provide access to the various Java classes that will be used to compile the sample servlet.

Update the following environment variables in your .profile:

PATH

Add **`/usr/lpp/java/J1.1/bin`** to the path variable. (If your Java install path is different than this — for example, `/usr/lpp/java18/J1.1/bin`, which is what it was on the system used to write this redbook — then use what your system has).

CLASSPATH

There is quite a list of files to add to the CLASSPATH:

```
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/ibmwebas.jar
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/jsp10.jar
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/lotusxsl.jar
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/servlet.jar
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/xml4j.jar
CLASSPATH=$CLASSPATH:/usr/lpp/WebSphere/AppServer/lib/x509v1.jar
CLASSPATH=$CLASSPATH:/usr/lpp/java18/J1.1/lib/classes.zip
```

This example illustrates a handy way to avoid having a long environment variable line that extends beyond the right side of the screen. The $CLASSPATH variable contains all the contents of CLASSPATH issues up to that point in the file. So by using $CLASSPATH at the beginning of each line, you can simply concatenate the new entry to the end of the previous entries. This method also ensures that any CLASSPATH setting established by the /etc/profile file is carried in and placed at the front of your CLASSPATH.

2.6.3.2 Modify the sample servlet code to match your environment

The sample servlet code is as follows. If you are reading the PDF version of this redbook, select the text and copy-and-paste the code to a text editor, then FTP it to your user home directory on your OS/390 system for compiling.

The file name on the OS/390 system must be TestJdbc.java

Here’s what you need to change in the source code:

- The value for `static String url =` needs to be one that can map to a “default_jdbcpool” JDBC connection pool statement in the was.conf file. Specifically, it needs to map to the “databaseurl” connection pool statement:

  ```
  jdbcconnpool.default_jdbcpool.databaseurl=jdbc:db2os390:<LOC>
  ```

  where `<LOC>` is the name of your DB2 location name (not subsystem ID).

```
// ### Start of sample code ###
import java.io.*;
import java.sql.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class TestJdbc extends HttpServlet {
  static String db2Driver = "COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver";
  static String url = "jdbc:db2os390sqlj:LOC";
  static String query = "SELECT MERFNBR,MESTNAME FROM CMNSRV.MERCHANT";
  static Connection con = null;

  public void init(ServletConfig config) throws ServletException
  {
    loadDriver(db2Driver);
    try {
      con = DriverManager.getConnection(url);
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
  }

  public void destroy() {                /* Close connection with Data Base*/
    try {
      if (con != null)
       con.close();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void doGet (HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html>");
    out.println("<title>Stores in Demomall</title>");
    out.println("<body>");
    out.println("<h1>Query of MERCHANT Table</h1>");
    out.println("<PRE>");
    executeSQL(out);
    out.println("</PRE>");
    out.println("</body>");
    out.println("</body></html>");
    out.close();
  }

  public void loadDriver(String driver) {
    try {
      Class.forName(driver);
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  public static void main(String args[]) {
    TestJdbc app = new TestJdbc();
    app.loadDriver(db2Driver);
    System.out.println("DB2 driver " + db2Driver + " is loaded.");
    try {
      con = DriverManager.getConnection(url);
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
    System.out.println("getConnection to " + url + " is OK.");
    System.out.println();
    executeSQL(System.out);
  }

  public static void executeSQL(PrintStream out) {
    Statement stmt = null;
    try {
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      while (rs.next()) {
        String sa = rs.getString(1);
        String sb = rs.getString(2);
        out.println("Table OUT  =  " + sa + "  " + sb );
      }
      out.println();
      out.println("executeSQL ended successfully.");
    }
    catch (SQLException e) {
      e.printStackTrace(out);
    }
    finally {
      try {
        if (stmt != null)
         stmt.close();
        if (con != null)
         con.close();
      }
      catch (Exception e) {
        e.printStackTrace(out);
      }
    }
  }

  public static void executeSQL(PrintWriter out) {
    Statement stmt = null;
    try {
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      while (rs.next()) {
        String sa = rs.getString(1);
        String sb = rs.getString(2);
        out.println("Merchant =  " + sa + "  " + sb );
      }
      out.println();
      out.println("executeSQL ended successfully.");
    }
    catch (SQLException e) {
      e.printStackTrace(out);
    }
    catch (Exception e) {       /* catch other exception like missing class*/
      e.printStackTrace(out);
    }
    finally {
      try {
        if (stmt != null)
         stmt.close();
      }
      catch (Exception e) {
        e.printStackTrace(out);
      }
    }
  }
}
// ### End of sample code ###
```

2.6.3.3 Compile the servlet code

With the file TestJdbc.java in the user home directory on your system, go into the OMVS shell. The work you did updating the .profile of your user should take effect, and the PATH and CLASSPATH variables should allow you to compile this code. Do the following:

1. Type `echo $PATH` and press Enter. Do you see `/usr/lpp/java/J1.1/bin` on the results that come back? If yes, then the PATH updates have taken effect.
2. Type `echo $CLASSPATH` and press Enter. Do you see all the CLASSPATH updates that you made in 2.6.3.1, “Update your user profile” on page 78 in the results that come back? If yes, then the CLASSPATH updates have taken effect.
3. Type `javac` and press Enter. This should invoke the compiler, but since you haven’t provided a program to compile, it should come back with the syntax of the command:

   ```
   use: javac [-g][-O][-debug][-depend][-nowarn][-verbose]
   [-classpath path][-nowrite][-deprecation][-d dir][-J<runtime flag>]
   file.java...
   ```

   This is a good sign: the Java compiler was found.
4. Now compile the code:

   ```
   javac TestJdbc.java
   ```

5. If all else works appropriately, you should simply get the input prompt back. Check for the existence of your class file:

   ```
   ls -al TestJdbc.class
   ```

2.6.3.4 Update your was.conf file to recognize the new servlet

The WAS plugin running in your WAS Webserver needs to be told there’s a new servlet in its environment. This is done with deployedwebapp definitions in the was.conf file. Do the following:

1. Edit the copy of was.conf applicable to the WAS plugin running for your WAS Webserver.
2. At the bottom of the file, add the following:

   ```
   # ========================================================
   #  TestJdbc servlet for validating JDBC connection
   # ========================================================
   deployedwebapp.servlet.host=default_host
   deployedwebapp.servlet.rooturi=/servlet
   ```

```
deployedwebapp.servlet.classpath=<WAS class dir>/servlets
deployedwebapp.servlet.documentroot=<WAS home dir>
webapp.servlet.servlet.TestJdbc.code=TestJdbc
webapp.servlet.servlet.TestJdbc.servletmapping=/TestJdbc
```

Where:

- `<WAS class dir>` is the directory you wish WAS to search in for your compiled servlets. You could make this your user's home directory if you wanted. Or your could point this to a directory set up under your WAS home directory where all servlets will reside
- `<WAS home dir>` is the root directory of your WAS plugin. Typically this is set as the directory in which the file was.conf is located.

Save the file.

2.6.3.5 Copy the class file to your WAS Webserver's servlet directory

Copy the compiled TestJdbc.class file to the directory named on the deployedwebapp.servlet.classpath statement in the was.conf file. If you pointed to your user's home directory, then you don't need to copy the file: WAS will look in that directory for the compiled code.

2.6.3.6 Restart the WAS Webserver

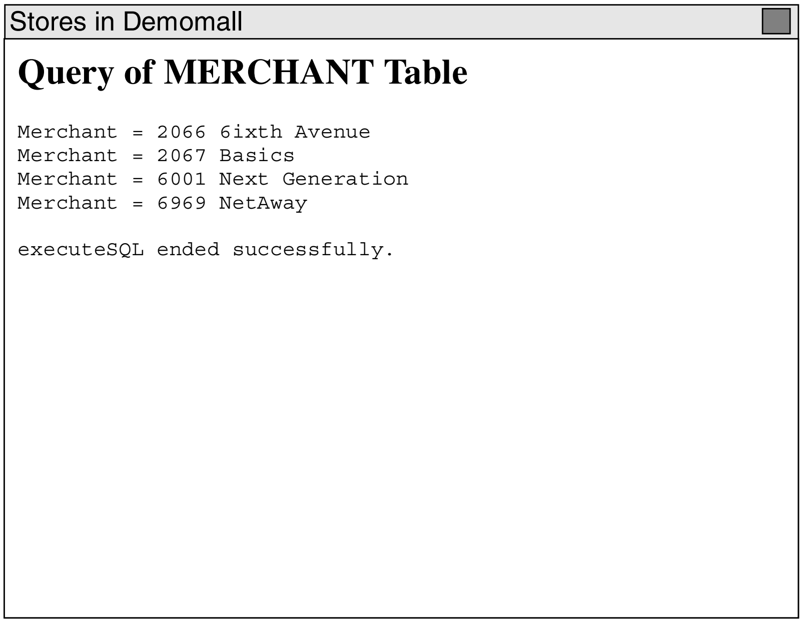
Changes to the was.conf file are picked up when the WAS Webserver is stopped and then started again.

2.6.3.7 Test the servlet

With the WAS Webserver up and running, point your browser to the following URL:

```
http://<your host name>:8080/servlet/TestJdbc
```

The output should look something like that shown in Figure 48.



Stores in Demomall

Query of MERCHANT Table

```
Merchant = 2066 6ixth Avenue
Merchant = 2067 Basics
Merchant = 6001 Next Generation
Merchant = 6969 NetAway

executeSQL ended successfully.
```

Figure 48. Output from TestJdbc servlet

If you see this output, then the servlet successfully made a JDBC connection and received query results back from DB2.

2.6.4 Enable WCSuite servlet support in the WAS plugin

Your copy of the WebSphere Application server (WAS) has been configured to run servlets and you have verified that the sample JDBC servlet (TestJdbc.java) works. Now let's configure the WAS plugin to know about the WCSuite Java code so we can test the basic Java function of WCSuite.

There is not much more to do, since much of the WAS configuration and WCSuite configuration was done earlier. To understand the new pieces of the configuration we will bring into the picture, refer to Figure 49.

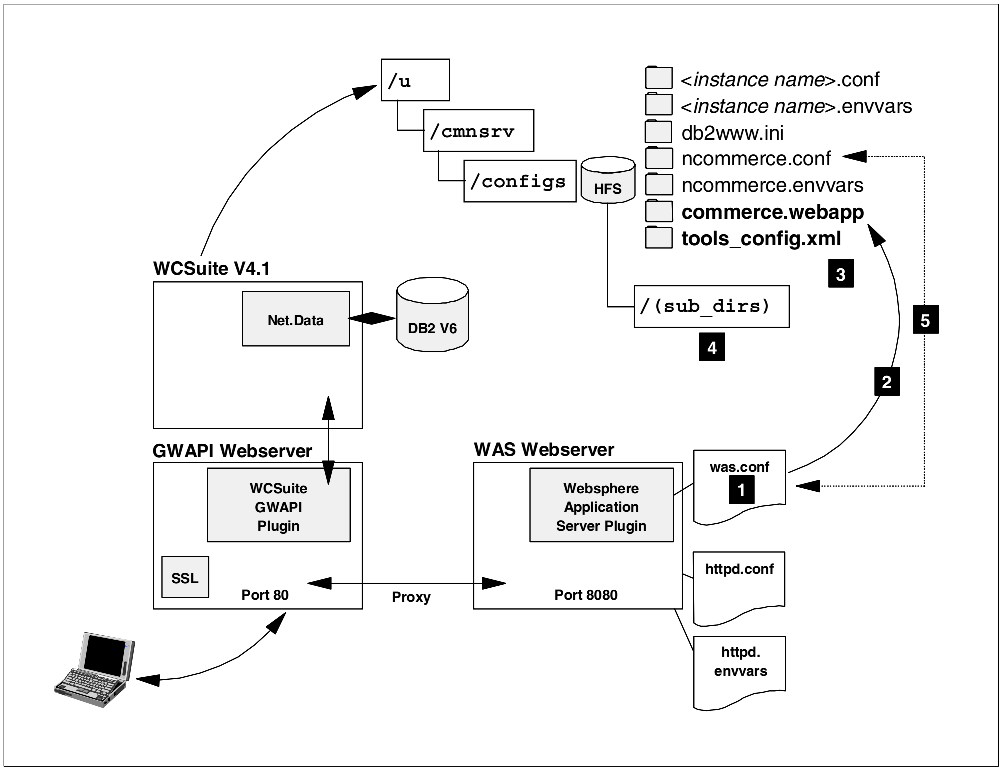


Figure 49. WCSuite-specific configuration objects for WAS plugin

The highlighted components are explained next:

1. The was.conf file will be updated to include another block of deployedwebapp statements just like you provided for the `TestJdbc.java` servlet. This new block will be for the WCSuite product.
2. If you recall the updates you made in was.conf for the `TestJdbc` servlet, you had both deployedwebapp *and* webapp statements. The deployedwebapp statements alert WAS to the existence of a Web application; the webapp statements provide specific infomation about the application.

   Advanced applications like WCSuite require quite a bit of specific information to be defined, and that would imply a great many webapp statements. Rather than cluttering up the was.conf file, WAS provides the ability to define the information in an XML-format file that will be read by WAS at the time the application is initialized. That is the way WCSuite defines its information.

Therefore, you will not see any webapp statements for WCSuite: the information is contained in a separate file called commerce.webapp.

The WAS plugin goes looking for the commerce.webapp file by searching the directories named on the deployedwebapp.<name>.classpath statement. It knows the name of the file will be commerce.webapp because the `deployedwebapp` statements have the term “commerce” in them. Finally, it knows to go looking for a separate file by the *lack* of any webapp statements associated with the deployedwebapp entries.

WCSuite provides a sample webapp.commerce file that you copy to your instance directory and update. That’s what we’ll have you do in a moment.

Note: The Configuration Guide for WCSuite contains a comment that, “When using an XML document, do not include any webapp definitions in the was.conf file.” When we first read that we were a bit confused. But now that we understand the nature of the deployedwebapp and webapp statements in was.conf, it makes more sense.

Stated a different way: WCSuite uses a separate XML document to provide its definitions to WAS. If WAS sees any webapp statements for a given application, it won’t go searching for an XML document. Therefore, for the WCSuite product do not code any webapp statements in was.conf. Doing so will disable the WCSuite Java-based functions.

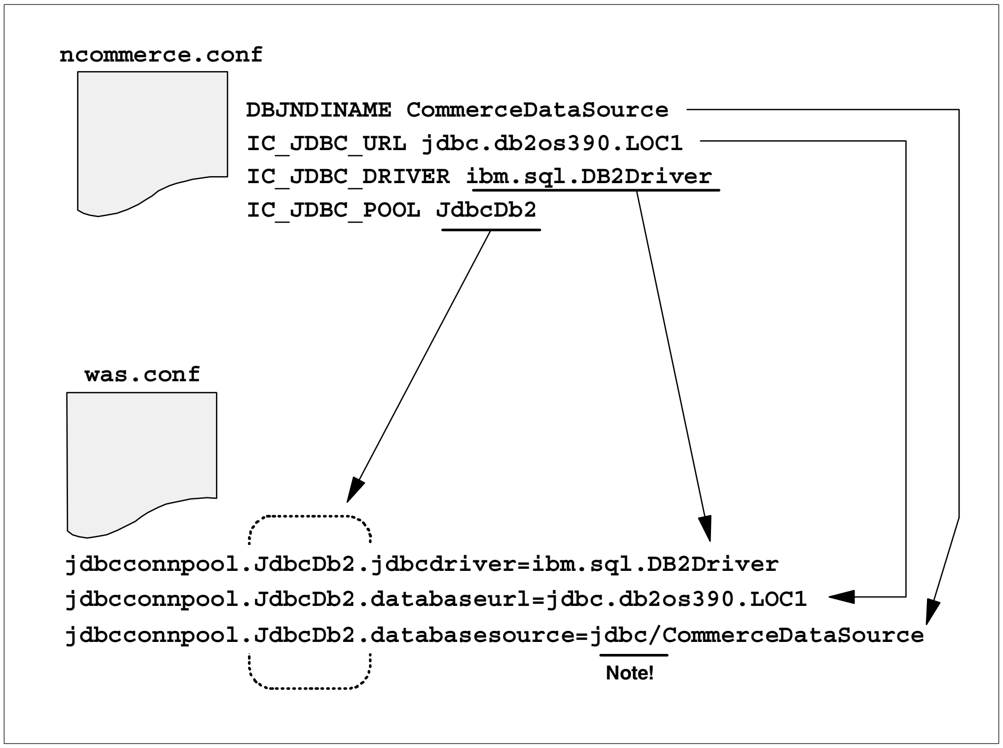
3. There is a second XML-based file associated with the WCSuite product, and its name is tools_config.xml. This file is used by the MerchantAdmin function of WCSuite. This file is pointed to by an XML tag in the commerce.webapp file that looks like this:

```
<init-parameter>
    <name>configfile</name>
    <value>/u/cmnsrv/configs/tools_config.xml</value>
</init-parameter>
```

   WCSuite supplies a sample of this as well, which you copy to your instance directory and update.

4. You must copy a set of directories and their files to your instance directory. These are files supplied with WCSuite, but they must reside under your instance directory, so it involves you manually performing the step.

5. Finally, there’s a relationship between some of the configuration parameters in the ncommerce.conf file and those found in the was.conf file. It is a relationship best illustrated with a picture (which is provided in Figure 50 on page 85).



```
DBJNDINAME CommerceDataSource
IC_JDBC_URL jdbc.db2os390.LOC1
IC_JDBC_DRIVER ibm.sql.DB2Driver
IC_JDBC_POOL JdbcDb2
```

```
jdbcconnpool.JdbcDb2.jdbcdriver=ibm.sql.DB2Driver
jdbcconnpool.JdbcDb2.databaseurl=jdbc.db2os390.LOC1
jdbcconnpool.JdbcDb2.databasesource=jdbc/CommerceDataSource
```

Figure 50. JDBC configuration relationship between ncommerce.conf and was.conf

What this is saying is the following:

- The IC_JDBC_POOL value in ncommerce.conf must map to a block of jdbcconnpool statements with the same name. It is in this way that the WCSuite servlets know which JDBC connection pool to use. You may either add another block to was.conf, or change the value of IC_JDBC_POOL in ncommerce.conf to match an existing connection pool already defined in was.conf.

  In this redbook we advocate the creation of a new block of lines in `was.conf`.
- The IC_JDBC_DRIVER value in ncommerce.conf must match the jdbcdriver value found in the block of connection pool statements in was.conf being used by WCSuite.
- The IC_JDBC_URL value in ncommerce.conf must match that found on the databaseurl statement, and the DB2 subsystem location name must match between each file *and* what's defined for your DB2 system.
- The DBJDNINAME value in ncommerce.conf is CommerceDataSource, and that value must be specified on a `databasesource` statement in the connection pool block. Note the “jdbc/” appended to the front of the value in the was.conf file.

2.6.4.1 Update was.conf file

Edit the was.conf file, and do the following:

1. `deployedwebapp` statements

   Scroll to the bottom and add the following block of statements to the file:

   ```
   # ===================================================================#
   # Added in support of WCSuite
   ```

```
# ==================================================================#

deployedwebapp.commerce.host=default_host
deployedwebapp.commerce.rooturi=/webapp/commerce
deployedwebapp.commerce.classpath=<config directory>
deployedwebapp.commerce.documentroot=/usr/lpp/CommerceSuite/stores
```

Where `<config directory>` is the directory in which your WCSuite configuration files are kept. In this Redbook example we've illustrated the use of a separate directory called /u/cmnsrv/configs. Whatever your directory is, place the directory name here.

Incidentally, as we explained back in 2.6.4, "Enable WCSuite servlet support in the WAS plugin" on page 83, the way WAS knows to look for a file called commerce.webapp is based on the word "commerce" coming immediately after "deployedwebapp" on the deployedwebapp statement

2. `appserver.classpath` updates

   Add the following statements to the `appserver.classpath` directive. Remember that the appserver.classpath line can not be split across lines, so the line will stretch well beyond the right side of a 3270 display (we show them on separate lines here simply to fit them all on one page). Values are separated by a colon (:).

   ```
   /usr/lpp/CommerceSuite/classes
   /usr/lpp/CommerceSuite/Tools/lib/StoreCreatorServer.jar
   /usr/lpp/CommerceSuite/Tools/lib/ToolsFrameWork.jar
   /usr/lpp/CommerceSuite/Tools/lib/properties.jar
   ```

3. `jdbcconnpool` statements

   Take a look at the `jdbcconnpool` statements that are already in your was.conf file. You should have a set in there with a name of default_jdbcpool. Replicate that block of lines and update them as shown in Figure 50 on page 85 so that the default values for the ncommerce.conf parameters are reflected in the was.conf file. Your was.conf file will now have at least two sets of `jdbcconnpool` statements.

Save the file.

2.6.4.2 Copy sample commerce.webapp and tools_config.xml files

This is a very simple step. If you look in the /usr/lpp/CommerceSuite/install directory, you will find two sample files:

commerce.webapp

tools_config.xml

Copy the files to the directory specified on the `deployedwebapp.commerce.classpath` statement. That's where WAS will look for the commerce.webapp file. In our example the directory was the same directory in which we are storing our WCSuite configuration files, /u/cmnsrv/configs. The tools_config.xml file will be explicitely pointed to from within the commerce.webapp file, so strictly speaking it doesn't have to reside in the same directory. But it keeps things easier to have all the configuration files together.

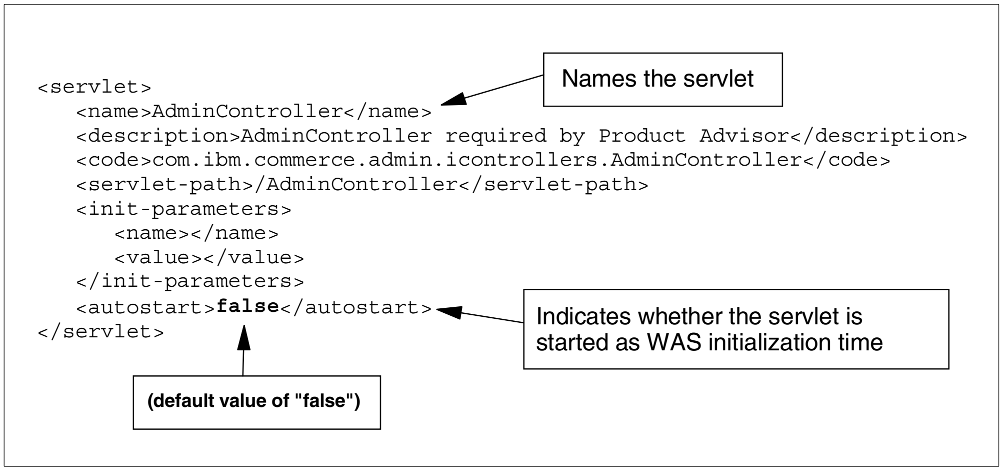
2.6.4.3 Modify the commerce.webapp file

The commerce.webapp file contains information about the servlet functions provided by WCSuite. There are eight servlets defined in commerce.webapp:

- `jsp`
- `BaseServlet`
- `CategoryDisplay`
- `ProductDisplay`
- `AdminController`
- `MerchantAdmin`
- `file`
- `ErrorReporter`

Our reason for having you edit this file is to turn on function if you plan on using the function. So do the following:

1. Edit the file you copied to your configuration directory.
2. If you plan on using the Product Advisor function, then locate the `AdminController` definition and change the value of `<autostart>` XML block from “false” to “true”. If you’re not planning on using Product Advisor, then you may leave this at the default of “false”. Figure 51 illustrates what the `AdminController` block of XML definitions looks like.



```
<servlet>
   <name>AdminController</name>
   <description>AdminController required by Product Advisor</description>
   <code>com.ibm.commerce.admin.icontrollers.AdminController</code>
   <servlet-path>/AdminController</servlet-path>
   <init-parameters>
      <name></name>
      <value></value>
   </init-parameters>
   <autostart>false</autostart>
</servlet>
```

Names the servlet

Indicates whether the servlet is started as WAS initialization time

(default value of "false")

Figure 51. Example of the “AdminController” XML block in the file commerce.webapp

The default value of “false” means AdminController won’t be started, and the Product Advisor function will not work. Changing it to “true” enables the servlet.

3. If you plan on using the Commerce Studio publishing function, then you must turn on the `MerchantAdmin` servlet. We will be illustrating the publishing function in this redbook, so turn it on:

```
<servlet>
   <name>MerchantAdmin</name>
   <description>MerchantAdmin required by Store Publishing</description>
   <code>com.ibm.commerce.tools.request_management.RequestManager</code>
   <servlet-path>/MerchantAdmin</servlet-path>
   <init-parameter>
      <name>configfile</name>
      <value>/usr/lpp/CommerceSuite/install/tools_config.xml</value>
   </init-parameter>
   <debug-mode>0</debug-mode>
   <uri-paths>default_host/webapp/commerce/MerchantAdmin</uri-paths>
   <enabled>true</enabled>
   <autostart>true</autostart>
</servlet>
```

4. In that same `<name>MerchantAdmin</name>` block of XML, change the directory specification for the tool_config.xml file as named in the `<init-parameter>` block:

```
<servlet>
   <name>MerchantAdmin</name>
   <description>MerchantAdmin required by Store Publishing</description>
   <code>com.ibm.commerce.tools.request_management.RequestManager</code>
   <servlet-path>/MerchantAdmin</servlet-path>
   <init-parameter>
       <name>configfile</name>
       <value><config directory>/tools_config.xml</value>
   </init-parameter>
   <debug-mode>0</debug-mode>
   <uri-paths>default_host/webapp/commerce/MerchantAdmin</uri-paths>
   <enabled>true</enabled>
   <autostart>true</autostart>
</servlet>
```

   Where `<config directory>` is the directory into which you copied the `tool_config.xml` file. In our example, that directory is `/u/cmnsrv/configs`.

5. Finally, go back to the top of the file and locate the `<BaseServlet>` definition block. Modify the value for the `<name>configfile</name>` definition to point to the ncommerce.conf file in your configuration directory:

```
<servlet>
  <name>BaseServlet</name>
   <description>BaseServlet required by Category/Product ...
   <code>com.ibm.commerce.server.BaseServlet</code>
   <servlet-path>/servlet</servlet-path>
   <init-parameter>
      <name>configfile</name>
      <value>/u/cmnsrv/configs/ncommerce.conf</value
   </init-parameter>
   <autostart>true</autostart>
</servlet>
```

   In this example we are illustrating the file's location as `/u/cmnsrv/configs`.

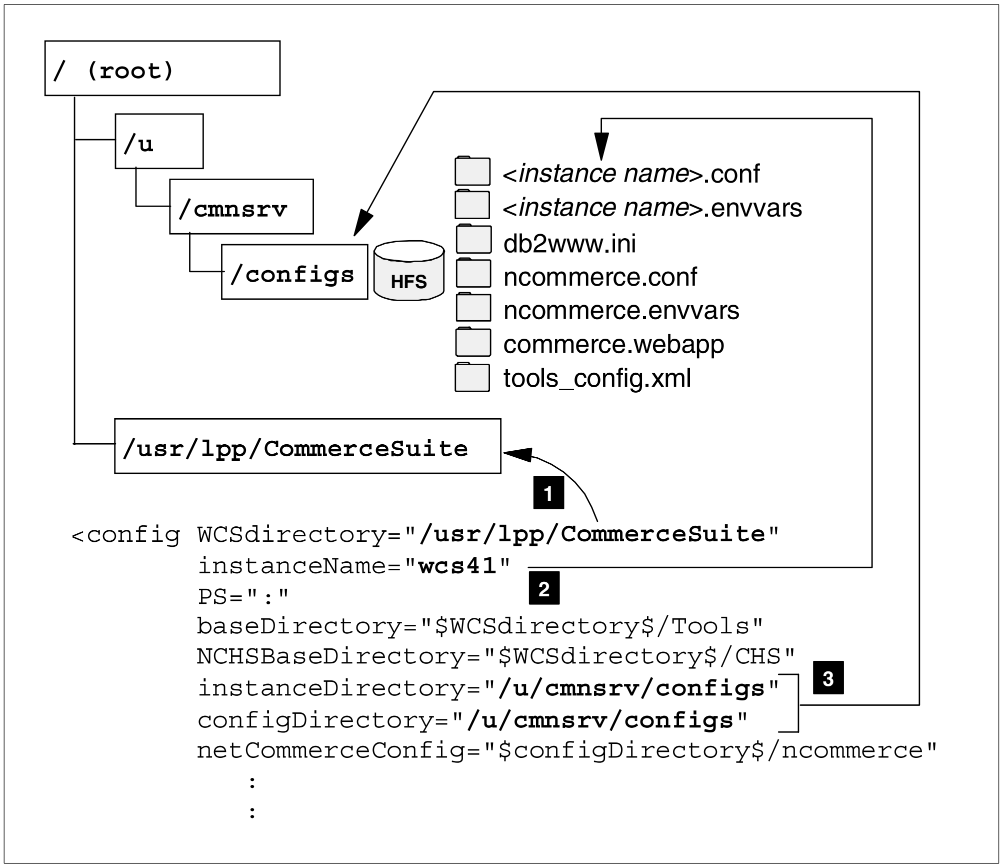
6. Save the file.

2.6.4.4 Modify the tools_config.xml file

The tools_config.xml file contains additional configuration information used by the WCSuite product. The file must reside in the directory pointed to in the MerchantAdmin section of the commerce.webapp file.

Do the following:

1. Edit the file.
2. Locate the start of the `<config>` block of XML definitions. You are going to make a few changes to this section to tell WCSuite about your environment. Figure 52 on page 89 illustrates the block of definitions and what you need to do.



```
<config WCSdirectory="/usr/lpp/CommerceSuite"
        instanceName="wcs41"
        PS=":"
        baseDirectory="$WCSdirectory$/Tools"
        NCHSBaseDirectory="$WCSdirectory$/CHS"
        instanceDirectory="/u/cmnsrv/configs"
        configDirectory="/u/cmnsrv/configs"
        netCommerceConfig="$configDirectory$/ncommerce"
            :
            :
```

Figure 52. Illustration of the changes to tools_config.xml file

Note: What Figure 52 illustrates is our configuration for this Redbook. Your configuration — particularly if you took the CMNCONF default behavior for the placement of the configuration files — may or will be different. We will point out where those differences may occur.

3. Here are the changes you need to make (the numbers refer to the numbered blocks shown in Figure 52):

   1 The `WCSdirectory=` value should point to the directory into which you installed the WCSuite product.

   2 The `instanceName=` value should name the instance of WCSuite you configured with CMNCONF. This will also be the name of the file `<instance name>.conf` in your configuration directory.

   3 The `instanceDirectory` value is used to point to the directory under which the various subdirectories will reside. (This is covered in the next section.) In this redbook example we are creating those subdirectories under the same directory in which we are storing our configuration files (/u/cmnsrv/configs).

   The configDirectory points to the directory in which the configuration files — specifically the ncommerce.conf file — resides. In our example, that is /u/cmnsrv/configs.

4. Save the file.

2.6.4.5 Create the sub-directories and populate with files

This step involves using the UNIX recursive copy function to copy the supplied sample directories and their files to your configuration directory.

From the OMVS shell, and at your user's home directory prompt, execute each of the following commands:

Note: Again, we are illustrating this using our example's /u/cmnsrv/configs directory. If your instance directory is different, then substitute that directory name for ours.

```
cp -r /usr/lpp/CommerceSuite/install/auctions /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/bpf /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/catalog /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/common /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/devtools /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/logs /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/nchs /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/optools /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/order_mgmt /u/cmnsrv/configs
cp -r /usr/lpp/CommerceSuite/install/pzn /u/cmnsrv/configs
```

The `copy` command with the `-r` flag will copy the directory and all underlying files over to your /u/cmnsrv/configs directory, creating the directory at the target on the fly.

2.6.5 End of Phase 3: Verify WCSuite servlet function

At this point you are ready to see if the WCSuite servlet function is working properly. Let's start this discussion with a picture of our completed environment (see Figure 53 on page 91).

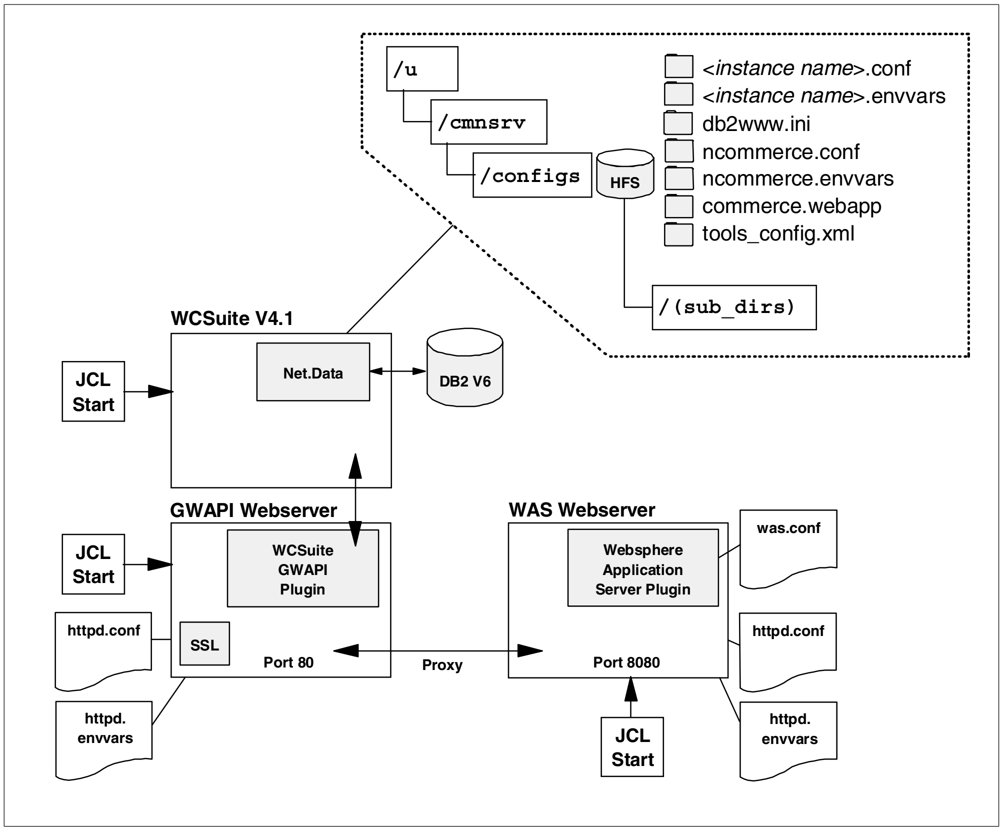


Figure 53. Phase 3 validation and environment configuration

The process of validating the WCSuite servlet function involves accessing the sample JSP pages in the “Next Generation” store of the Demomall. Doing so exercises all the components we show in Figure 53.

2.6.5.1 Clear the WAS logs directory

We are going to have you look at the WAS ncf log to validate that the WCSuite `BaseServlet` comes up. It can be confusing selecting the appropriate ncf log file, particularly if there are several in the directory. So, we are going to have you clear the log directory with the following steps:

1. Browse the was.conf file, and locate the directive `appserver.loglevel`. Make certain that it says `WARNING`.
2. Locate the directive `appserver.logdirectory`. This is where the ncf log will be written. Note the directory.
3. Go to that log directory and clear out any files that have the name ncf.log.* (you can clear the whole directory of files if you wish).

2.6.5.2 Start WCSuite and the two Webservers

Start the three started tasks in this order:

1. GWAPI Webserver

   The console message indicating a successful start is:

   ```
   IMW3536I SA 67109432 0.0.0.0:80 * * READY
   ```

2. WCSuite

```
CMN0411I THE Commerce Suite SERVER IS READY FOR CONNECTIONS 003
```

3. WAS Webserver

```
IMW3536I SA 16777834 0.0.0.0:8080 * * READY
```

2.6.5.3 Ensure “BaseServlet” is up and running

Go to the directory specified on the was.conf file’s `appserver.logdirectory` directive and browse the file `ncf.log.<date stamp>`. Scroll down until you find the following string:

```
Load group: commerce
```

This indicates the `deployedwebapp` directive in the was.conf file has been encountered, and that WAS is about to attempt to load the group. This will start the process of looking for the commerce.webapp file discussed in 2.6.4, “Enable WCSuite servlet support in the WAS plugin” on page 83.

A few lines further down the file you should see:

```
Instantiate: com.ibm.commerce.server.BaseServlet
```

This means WAS will attempt to load the `BaseServlet` servlet. If you look at the content of the log, you’ll see many of the configuration parameters from ncommerce.conf being read in. Keep scrolling down the file. If you come across another “Instantiate:” log record without encountering an “Exception” log record, then your `BaseServlet` has successfully come up.

Some problems you might encounter:

- *No “Load group: commerce” line in ncf log*

  You missed the step detailed in 2.6.4.1, “Update was.conf file” on page 85.

- *No “Instantiate ... BaseServlet” line in ncf log*

  Caused by `<autostart>` value for `BaseServlet` in commece.webapp file being set to false. The default is `true`.

- *“Cannot switch security thread ID” message in ncf log*

  This is caused by a mismatch between the password implied by the `DBPASS` parameter of ncommerce.conf being different from the RACF password value for the CMNSRV ID. They must match. The `DBPASS` parameter is set on the “System Configuration, 3 of 3” panel of CMNCONF.

- *WAS can’t find the JDBC Serialized file*

  The symptom of this would be the following in the ncf log:

  ```
  Error: JDBC is not allowed without a valid JDBC serialized profile
  The following error messages were received while trying to read the JDBC profile
  Unable to find the JDBC Serialized Profile: DSNJDBC_JDBCProfile.ser
  ```

  To correct this, make sure the file DSNJDBC_JDBCProfile.ser, created during the running of the `db2genJDBC` step as discussed in 2.6.2.1, “Running the db2genJDBC utility” on page 73, is in a directory specified on the `appserver.classpath` directive of was.conf.

Assuming everything worked properly, then proceed to the next step.

2.6.5.4 Access the Demomall

Issue the following URL from your browser:

```
http://<your host name>/demomall/basemall.htm
```

When you get the Demomall's home page (a static HTML file), you have reached the GWAPI Webserver, and it has simply pulled the HTML file from the HFS. Figure 54 illustrates this.

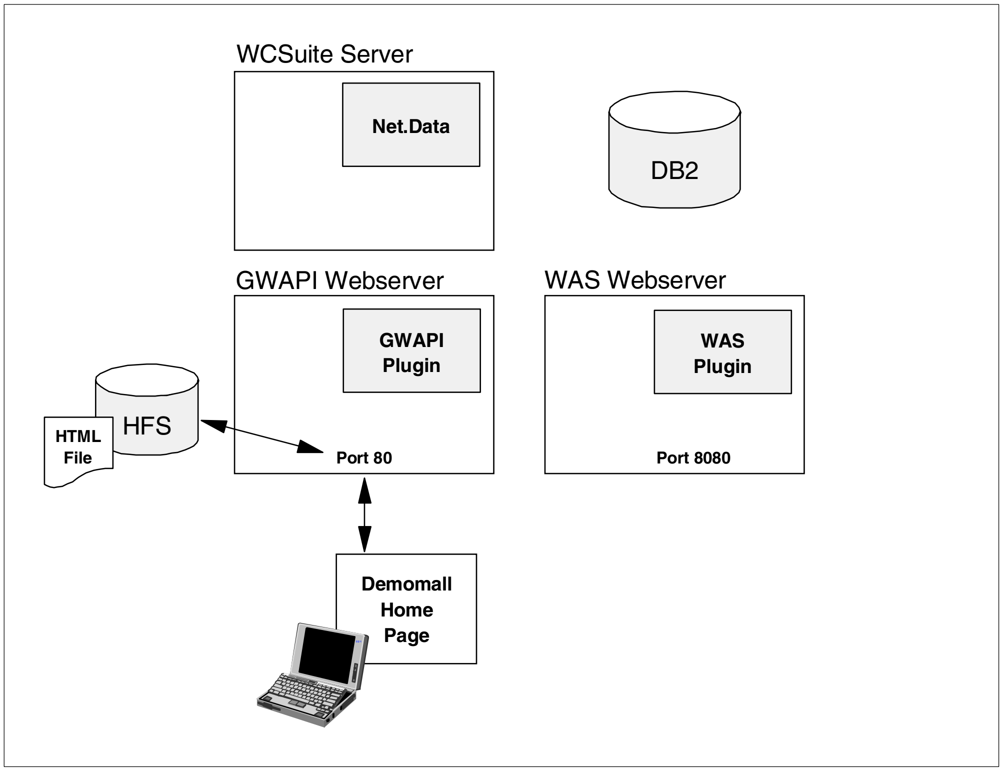


Figure 54. Accessing the Demomall home page

2.6.5.5 Access the Mall Directory

You should have the Demomall home page on your browser screen. Now click on the "Home" icon near the top of the screen. The WCSuite command under that icon is:

```
/webapp/commerce/command/ExecMacro/mall_dir.d2w/report
```

Clicking on it will run the Net.Data macro `mall_dir.d2w`. When you get the mall directory page, you will have made it up to the WCSuite Server. Figure 55 on page 94 illustrates this.

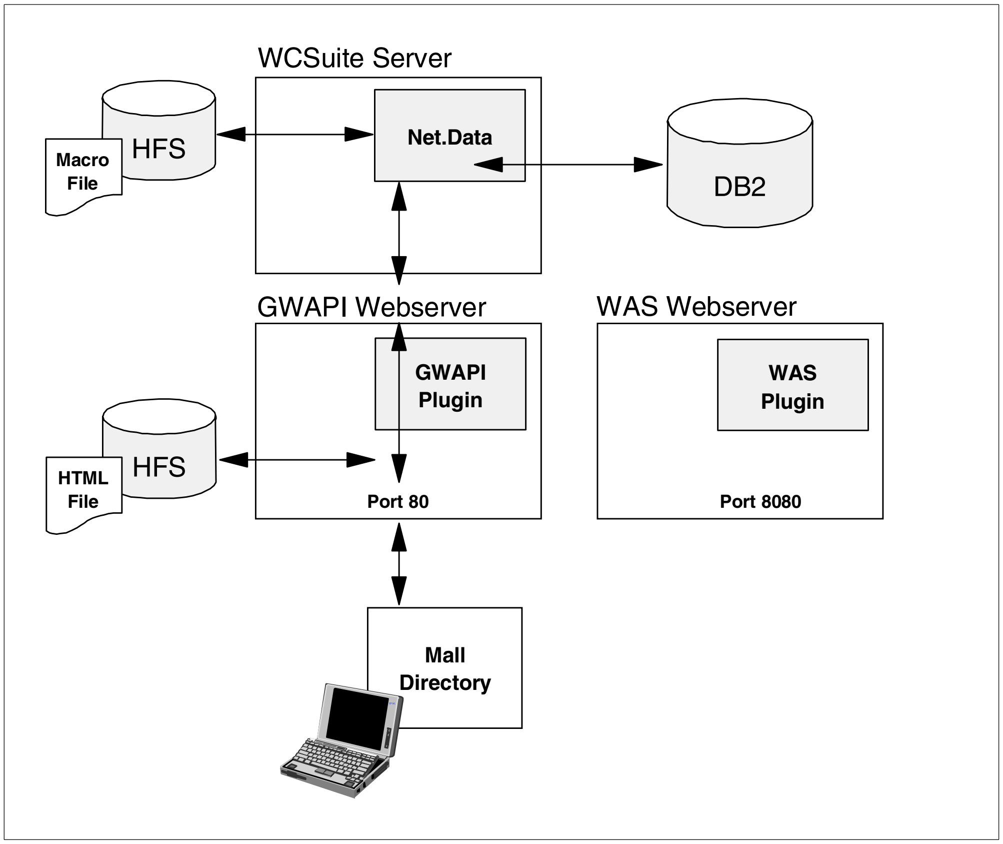


Figure 55. Accessing the Demomall's Mall Directory

2.6.5.6 Go to the "Next Generation" store and select "Thinkpads"

The Mall Directory will list a store called "Next Generation" which will, when the link for the store is selected, display a static HTML page. That page will have a link on it that reads:

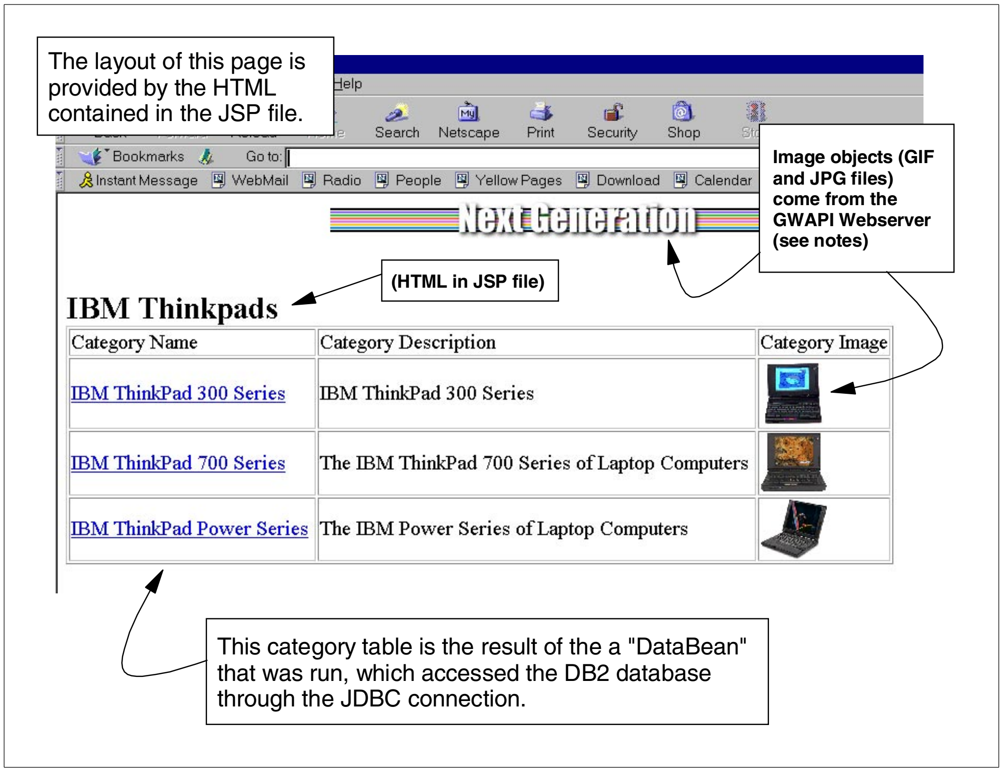
> "Click **here** to enter the Thinkpads category."

That link has a command under it that looks like this:

```
http://<your host>/webapp/commerce/servlet/CategoryDisplay?merchant_rn=6001&cgrfnbr=33
```

If you recall the explanation offered in 2.5, "Phase 2: Proxy server configuration and validation" on page 59, the `Proxy` statements in the httpd.conf file will catch the `/webapp/commerce/servlet/*` portion of that and route the request over to the WAS Webserver listening on port 8080.

If all works well, the screen that comes back should look like Figure 56 on page 95.



The layout of this page is provided by the HTML contained in the JSP file.

Image objects (GIF and JPG files) come from the GWAPI Webserver (see notes)

(HTML in JSP file)

IBM Thinkpads

Category Name	Category Description	Category Image
IBM ThinkPad 300 Series	IBM ThinkPad 300 Series	
IBM ThinkPad 700 Series	The IBM ThinkPad 700 Series of Laptop Computers	
IBM ThinkPad Power Series	The IBM Power Series of Laptop Computers	

This category table is the result of the a "DataBean" that was run, which accessed the DB2 database through the JDBC connection.

Figure 56. A JSP page being run to show categories of Thinkpads

The comment about the GIF/JPG files coming from the GWAPI Webserver (as opposed to the WAS Webserver on which the JSP is running) might surprise people. But it does make sense: each image object is brought down to the browser as the result of a separate, under-the-covers HTTP request to the Webserver. Further, the request is for the specific file that represents the graphic image. The format of the URL request for the “Next Generation” image, for example, is:

```
/nextgeneration/NG_head.gif
```

The request hits the GWAPI Webserver first (the one listening on port 80), and it is that Webserver’s `httpd.conf` directives that are interrogated for a match. If you look in the GWAPI Webserver’s httpd.conf file, you will find the following:

```
Pass /nextgeneration/*  /usr/lpp/CommerceSuite/models/demomall/html/en_US/*
```

That `Pass` directive is the one that matches, and that means the GWAPI Webserver will look for the image file in the directory shown on the directive.

So the picture of JSP operation is as shown in Figure 57 on page 96.

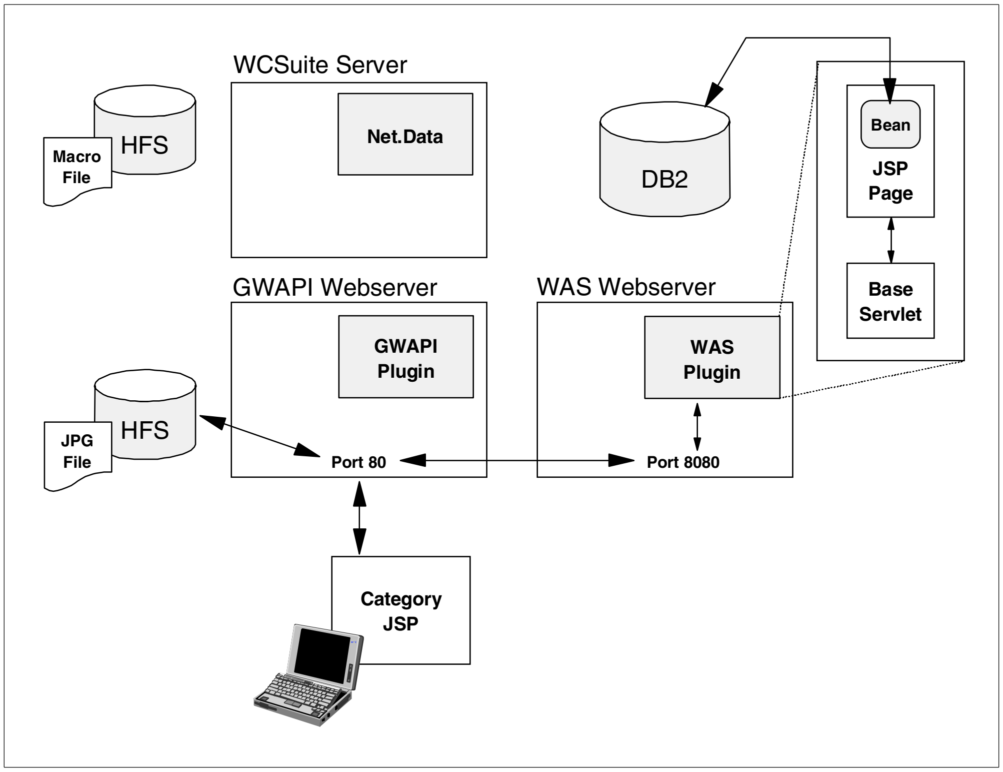


Figure 57. System diagram of running a JSP page

2.7 Phase 4: Commerce Studio publishing validation

The ability to publish “assets” (files, folders and information in the database) from the WebSphere Commerce Studio tool on Windows NT to the OS/390 server involves having a few key things in place:

- The “MerchantAdmin” servlet must be up and running in the WAS Webserver to accept the publish request from the NT machine.
- The assets need to be in a state on the NT machine where they’re ready to be published.

In this section we walk you through the verification of the readiness of the OS/390 system. We leave the second topic until Chapter 5, “Using WebSphere Commerce Studio” on page 127, which covers the process of store creation and publishing.

2.7.1 Readying the OS/390 server to accept publishing

The act of publishing assets from the NT-based Studio environment requires a few things to be in place before it will work. The process involves both FTP-ing of files and communication with a WCSuite servlet to “catch” and then place the information on the OS/390 system. Follow the instructions provided here to get all the pieces working.

2.7.1.1 Ensure MerchantAdmin servlet is set to start automatically

Back in 2.6.4.3, “Modify the commerce.webapp file” on page 86 we had you edit the commerce.webapp and, among other things, set to “true” the value of

`<autostart>` in the MerchantAdmin section of the file. Browse the file (in this redbook example we had you place that file in /u/cmnsrv/configs) and make sure you see the following:

```
<servlet>
   <name>MerchantAdmin</name>
:
:
<autostart>true</autostart>
</servlet>
```

If you find that it is *not* set, then set it to “true” and restart the WAS Webserver.

2.7.1.2 Ensure other components are working

To get publishing to work, you need to have other system components in place as illustrated in Figure 58, and discussed immediately after the figure.

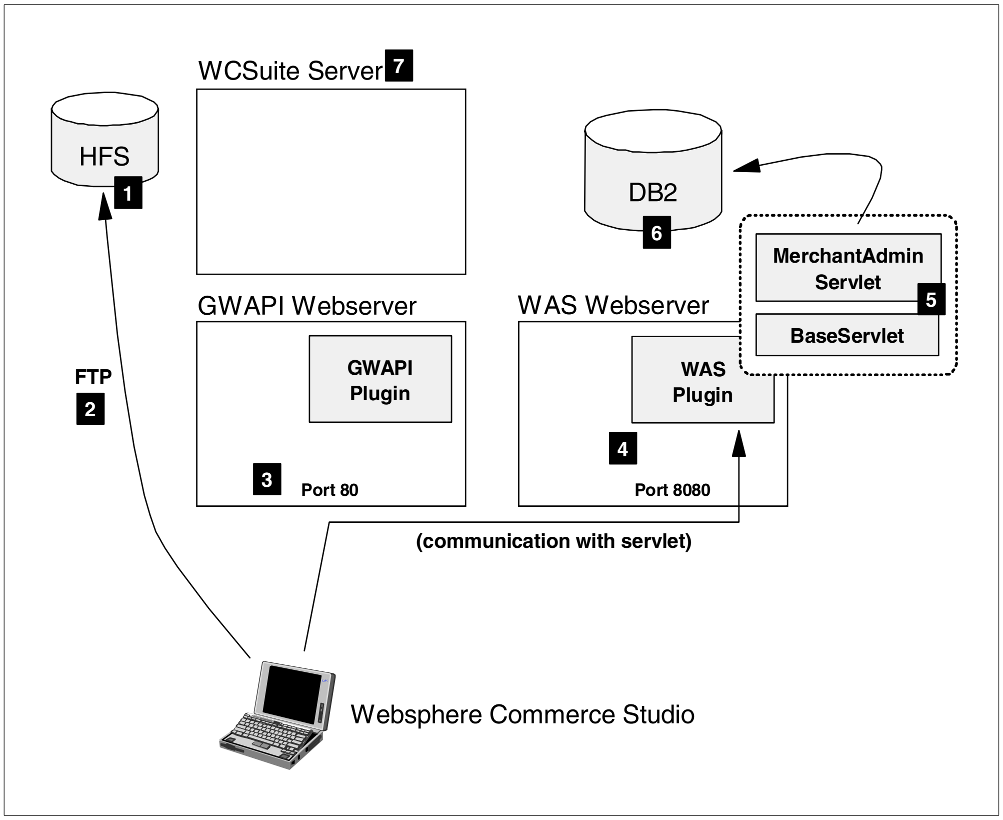


Figure 58. System requirements on OS/390 to accept publishing

On the target OS/390 system, ensure the following:

1. Write access to the target directories

   In order for the NTplatform to successfully publish to the target directories, it needs write access to them. The default directory will be /usr/lpp/CommerceSuite/stores, and it will have, by default, 755 permissions. That means the ID must either own that directory, or have UID(0).

2. FTP daemon running on OS/390 system

The Studio will publish file assets via FTP. In order to accomplish that, the FTP daemon must be up. Verify that it is, either by displaying the active tasks, or by manually FTP-ing a test file to the system.

3. GWAPI Webserver up and running

   The route the Studio platform will have to the “catcher” servlet running on the WAS Webserver is through the proxy function of the GWAPI Webserver. Therefore, the GWAPI Webserver needs to be up and running. The following is the console message indicating the Webserver is up:

   ```
   IMW3536I SA 924 0.0.0.0:80 * * READY
   ```

4. WAS Webserver up and running

   For the “catcher” servlet (which is really the MerchantAdmin servlet) to do its job, the WAS plugin needs to have a place to operate. That place is the WAS Webserver we’ve been writing about throughout this chapter. That Webserver, with the WAS plugin, needs to be up and running. The following is the console message indicating the Webserver is up:

   ```
   IMW3536I SA 924 0.0.0.0:8080 * * READY
   ```

5. BaseServlet and MerchantAdmin servlet up and running

   The way you can check to see if the servlets came up okay is to look in the logs directory for the WAS Webserver, and browse the ncf log. In there you will see the servlets being instantiated:

   ```
   Instantiate: com.ibm.commerce.server.BaseServlet
   ```

   and after that will come a screen or two of messages about the BaseServlet as it’s coming up. What you look for is the next “instantiate” message without any errors or warnings for the BaseServlet. If all you see are informational messages, and you see the next “instantiate message,” then you can assume the BaseServlet came up.

   The process for the MerchantAdmin servlet is a bit more tricky because there is no reference to it in the ncf log *when it succeeds*. It’ll show error messages when it fails, but nothing when it succeeds. So, when you scan the ncf log, if you see no mention of MerchantAdmin, it’s a *good* sign.

6. DB2 up and running

   The MerchantAdmin servlet will access DB2 through the JDBC interface.

7. WCSuite server up and running

   This is required if you wish to launch the store from the Studio environment. Having the WCSuite server up isn’t necessary to do the FTP or updates to the database.

2.7.2 Using Studio to create assets

The process for creating a store and publishing it to the S/390 server is detailed in Chapter 5, “Using WebSphere Commerce Studio” on page 127. Rather than recreating all of that here, we will rely on the information in that chapter. When you successfully get a store to publish, come back to this point.

2.8 Phase 5: WCSuite scheduler and personalization validation

The “scheduler” is a function provided with WCSuite that periodically (based on your setting) “wakes up” and launches jobs. The jobs the scheduler is to run are registered in the DB2 database. The scheduler is used by the auction support of WCSuite to provide automated updates of things like “proxy” bids and auction starts and closings.

The personalization function utilizes a licensed third party “rules engine” from Blaze Software. The configuration of this function involves many manual steps. Go to 6.2, “Providing a personalized product recommendation” on page 164 for information on this process.

2.8.1 Configuring the scheduler

The process involves re-running CMNCONF and changing one of the values on the “System Configuration 3 of 3” panel.

Important

The scheduler’s ability to operate is dependent upon the application of PTFs UQ47696, UQ47697, UQ48212 and UQ48213. Failure to apply these PTFs will result in the scheduler failing to start. The PTFs fix a corrupted CMNSCHED member delivered in the original SMP/E tape.

Do the following:

1. Go back into CMNCONF from Option 6 of ISPF
2. On the first panel of CMNCONF you should now see your instance listed. Select option 2 — modify instance — and specify the name of your instance in the provided field.
3. Select Option 1 on the CMNMAIN panel — System Configuration.
4. When the “System Configuration, 1 of 3” panel comes up, simply press Enter to go to the next panel.
5. When the “System Configuration, 2 of 3” panel comes up, simply press Enter to go to the third panel.
6. On the “System Configuration, 3 of 3” panel, change the value of “Start Scheduler” to “Yes.”
7. Press Enter to go back to the CMNMAIN panel.
8. Run through the “Access Control” function so the changes make their way into the various configuration files. You must restart WCSuite to have these changes picked up.

2.8.2 Validate scheduler activation

Validate the scheduler’s start by issuing the following command to the OS/390 console:

```
D OMVS,U=CMNSRV
```

where the value following `U=` is the RACF userid that owns the started task. This is the ID you created back in 2.4.3, “Create group ID and user ID under which

WCSuite will run” on page 33. The results you should see from this command will look something like this:

```
D OMVS,U=CMNSRV
BPXO040I 14.15.01 DISPLAY OMVS 702
OMVS     000F ACTIVE          OMVS=(9A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
CMNSRV   CMNMSERV 0056   50332461          1 HFI   12.52.23    560.189
  LATCHWAITPID=         0 CMD=CMNSRVRC
CMNSRV   CMNMSERV 0056        814   50332461 1FI   12.52.24    560.189
  LATCHWAITPID=         0 CMD=CMNSERV
CMNSRV   CMNMSERV 0056        815   50332461 1FI   12.52.24    560.189
  LATCHWAITPID=         0 CMD=CMNSERV
CMNSRV   CMNMSERV 0056   67109680   50332461 1FI   12.35.52    560.189
  LATCHWAITPID=         0 CMD=CMNSCHED
```

The example of the D OMVS,U=CMNSRV command was based on our example scenario where CMNSRV was the owning ID, CMNMSERV was the started task, and we defined two processes to the WCSuite configuration.

The existence of CMD=CMNSCHED indicates the scheduler is started.

You see four processes here because:

- The “server controller” is a separate process. The controller has a CMD=CMNSRVRC value.
- Two child processes are started, as defined during the configuration. These are shown with CMD=CMNSERV.
- One scheduler task is started. This has a value of CMD=CMNSCHED.

2.8.3 Using the auction functionality

The scheduler is an important component of the auction function of WCSuite, but enabling the scheduler is not the only thing you need to do. Refer to 6.1, “Creating an auction in your store” on page 161 for a further discussion of this process.

Chapter 3. General migration considerations

This chapter covers the general migration issues when moving from Net.Commerce 3.1.2 for OS/390 to WebSphere Commerce Suite 4.1 for OS/390. Here, we cover migrating HTML files and images, Net.Data macros, and C++ commands and functions. The biggest task when moving your site is the migration of the Net.Commerce database (DB2), which is covered in more details in Chapter 4, “Migrating the database” on page 109.

3.1 HTML and image considerations

HTML files should be relatively simple to migrate from Net.Commerce to WebSphere Commerce Suite. One consideration to remember is that files in the OS/390 UNIX System Services Hierarchical File System (HFS) are by default stored in EBCDIC format. When the OS/390 Webserver passes these files down to the browser, it translates them from EBCDIC to ASCII “on the fly”. However, it is possible to store HTML files in the OS/390 HFS in ASCII format and have the Webserver pass them down to the browser untranslated. See *WebSphere Application Server for OS/390 HTTP Server Planning, Installing, and Using*, SC31-8690 if you wish to store your files in ASCII format in the HFS.

HTML files should be transferred to the OS/390 server using copy or a file transfer program such as ftp in ASCII mode, and the image files (gif, jpeg, and so on) should be transferred using copy or a file transfer program such as ftp in binary mode. For copy in the same LPAR you may even be able to just mount the HFS from WebSphere Commerce Suite and copy the needed files over.

Figure 59 details the areas where you will find the HTML and gif files on Net.Commerce and where they should go on the WebSphere Commerce Suite server.

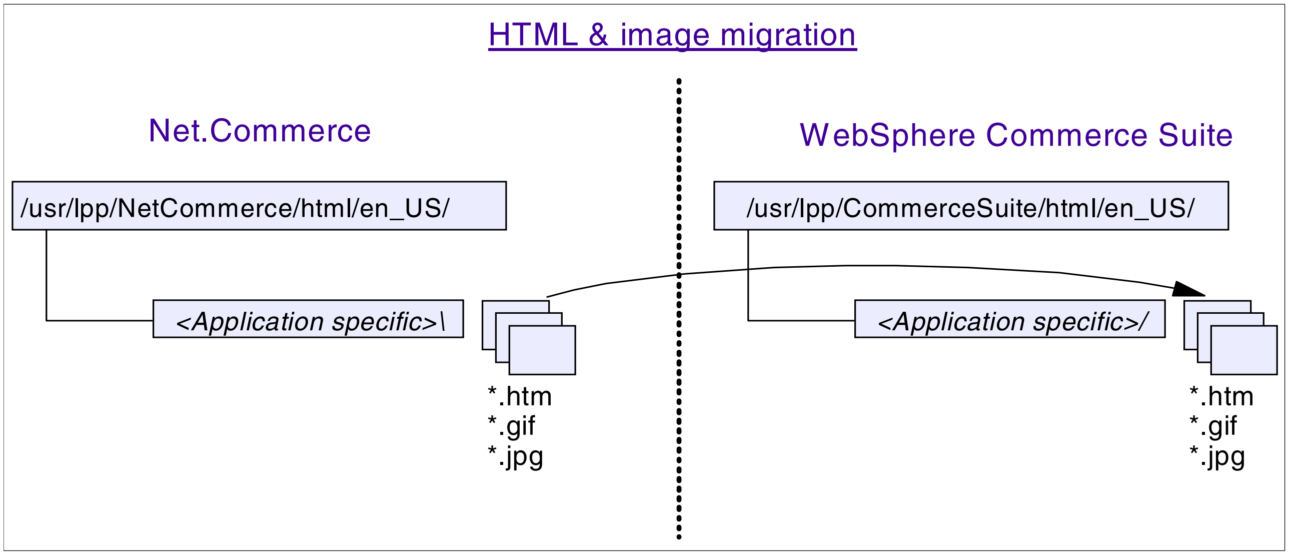


Figure 59. HTML and image migration

These files are referenced by the `Pass` directives in the Webserver's httpd.conf file. A Pass directive looks like the following:

```
Pass /demomall/* /usr/lpp/CommerceSuite/html/en_US/demomall/*
```

This means that whenever the Webserver receives a URL request with the string `demomall` immediately following the host name, it resolves the location to the full path as specified by the Pass directive.

Therefore, if you have your HTML files in a directory called /usr/lpp/NetCommerce/html/en_US/myhtml, and image files in a directory called /usr/lpp/CommerceSuite/html/en_US/myimages, and you want to refer to them just as myhtml and myimages, you would need to include two Pass directives in the httpd.conf file:

```
1. Pass /myhtml/* /usr/lpp/CommerceSuite/html/en_US/myhtml/*
2. Pass /myimages/* /usr/lpp/CommerceSuite/html/en_US/myimages/*
```

Do not put these Pass directives in the directives added by the WebSphere Commerce Suite configuration utility CMNCONF. The statements added by CMNCONF are located at the top of the Webserver configuration file (httpd.conf by default). CMNCONF overwrites this section in the Webserver configuration file every time you modify an existing instance or create a new instance with CMNCONF. These directives are between these two comment blocks in the Webserver configuration file:

```
###### IBM CommerceSuite ######
... various statements...
###### End of IBM CommerceSuite ######
```

To activate the changes, you must restart the Webserver each time its configuration file is modified. You may issue the following OS/390 console command to restart the Webserver:

```
F <Web server task name>, APPL=-RESTART
```

For more information about running the CMNCONF utility, refer to 2.4.12, “Configure an instance using the CMNCONF utility” on page 39.

3.1.1 Shopper password migration

The password information for shoppers is stored in the Commerce database (Net.Commerce and WebSphere Commerce Suite) in the *shopper* table. Since the encryption format has changed between Net.Commerce and WebSphere Commerce Suite, a straight copy of the shopper table will not work. A one way encryption utility of the old password with the new encryption routine will give us the correct password representation on WebSphere Commerce Suite.

At the time of writing this book, an update to the WebSphere Commerce Suite code is being written that will allow migration of the Net.Commerce password in a one way migration into the WebSphere Commerce Suite 4.1 format. To make sure that you will have the right code level, we strongly recommend that you install all maintenance, in particular the first 3 PTFs.

3.2 Net.Data compatibility

Net.Data capability and macro format is the same between the two different OS/390 e-commerce products, so you should be able to copy the macros over from Net.Commerce to WebSphere Suite. The schematic shown in Figure 60 is a visual representation of the location of various Net.Data macros in an existing Net.Commerce implementation and where they might move to in an OS/390 WebSphere Commerce Suite environment.

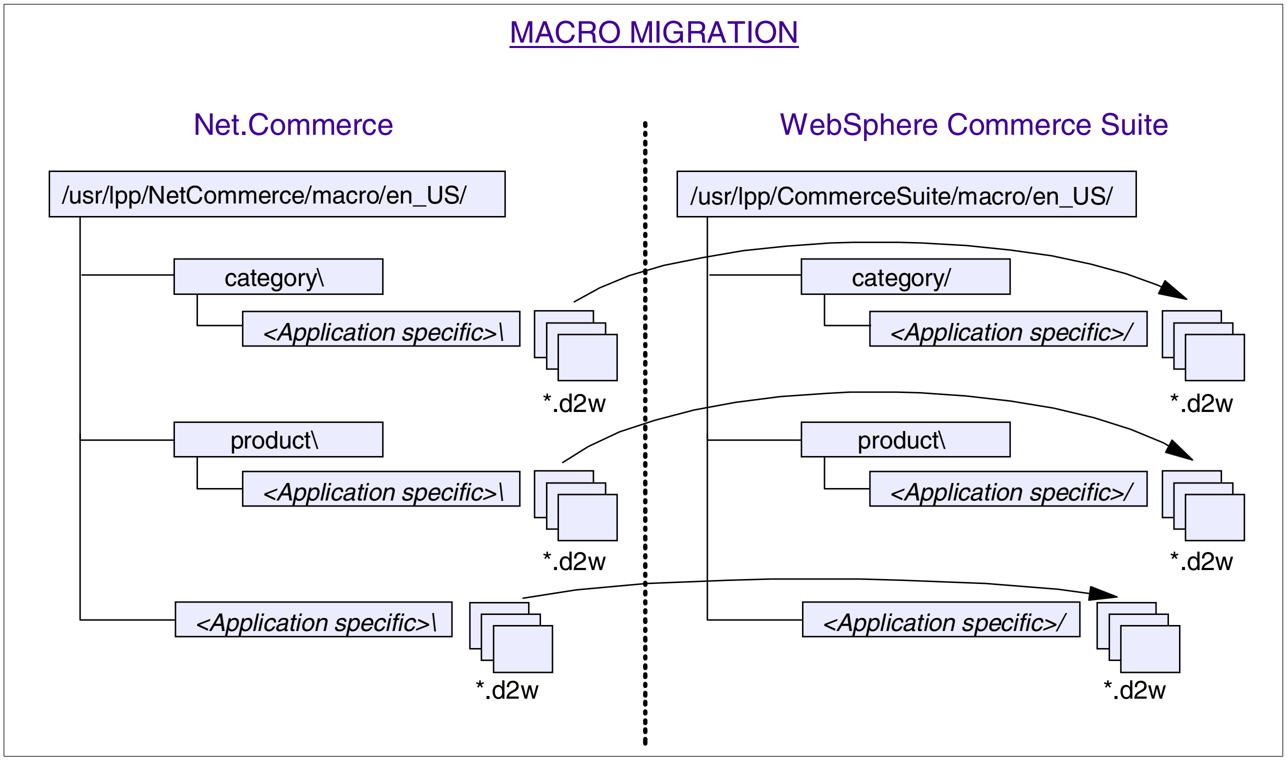


Figure 60. Migration of Net.Data macros from Net.Commerce to WebSphere Commerce Suite.

3.3 Backend integration

There are tools available for integrating an OS/390 Net.Commerce V3.1.1 site to various existing backend fulfillment systems. (These have not been tested at the 3.1.2 level or on the WebSphere Commerce Suite 4.1 level). IBM has made enterprise integration samples available for connecting Net.Commerce to CICS, IMS, MQSeries, and electronic data interchange (EDI) servers, along with miscellaneous utilities. The sample code is available on the Web at:

```
http://www.s390.ibm.com/nc/ecommerce/sampcode.html
```

The IBM-supplied function, available on the Web, is structured as follows:

- CICS and IMS integration: two high-level communication functions for executing transaction programs on both CICS and IMS. Each function will allow you to specify all input data for a given transaction program and receive all of the output in a single call.

- EDI integration: a set of functions that permit the collection of purchase order information from the Net.Commerce system and the exchange of this information and responses with an EDI service provider.
- MQSeries integration: a set of functions that allow the exchange of information with an MQSeries application program.
- Asynchronous server: a separate server daemon to manage information exchange that is asynchronous to the Net.Commerce Server daemon operation.
- Miscellaneous utilities and samples, including:
    - A family of functions that perform data transformation; for example, integer to zoned decimal and packed decimal to integer for communicating with transaction programs that are written in COBOL.
    - A set of 19 functions that are skeleton replacements for the existing Net.Commerce server API functions. Some functions are populated with sample calls to the CICS, IMS, and MQSeries communication functions.
    - Exception macros for processing errors encountered in the API task functions.
    - Documentation in HTML for online viewing and PostScript for hardcopy that describes how to make use of the supplied functions.

3.4 Commands, Tasks, and Overridable Functions

In Net.Commerce, a *Command* is an instruction to Net.Commerce to do some work. The Command is actually comprised of smaller units of work called *Tasks*. The number of Tasks associated with a Command varies by Command; this same structure applies to WebSphere as well.

Figure 61 shows a summary chart of Commands, Tasks, and Overrideable Functions and how they relate to each other.

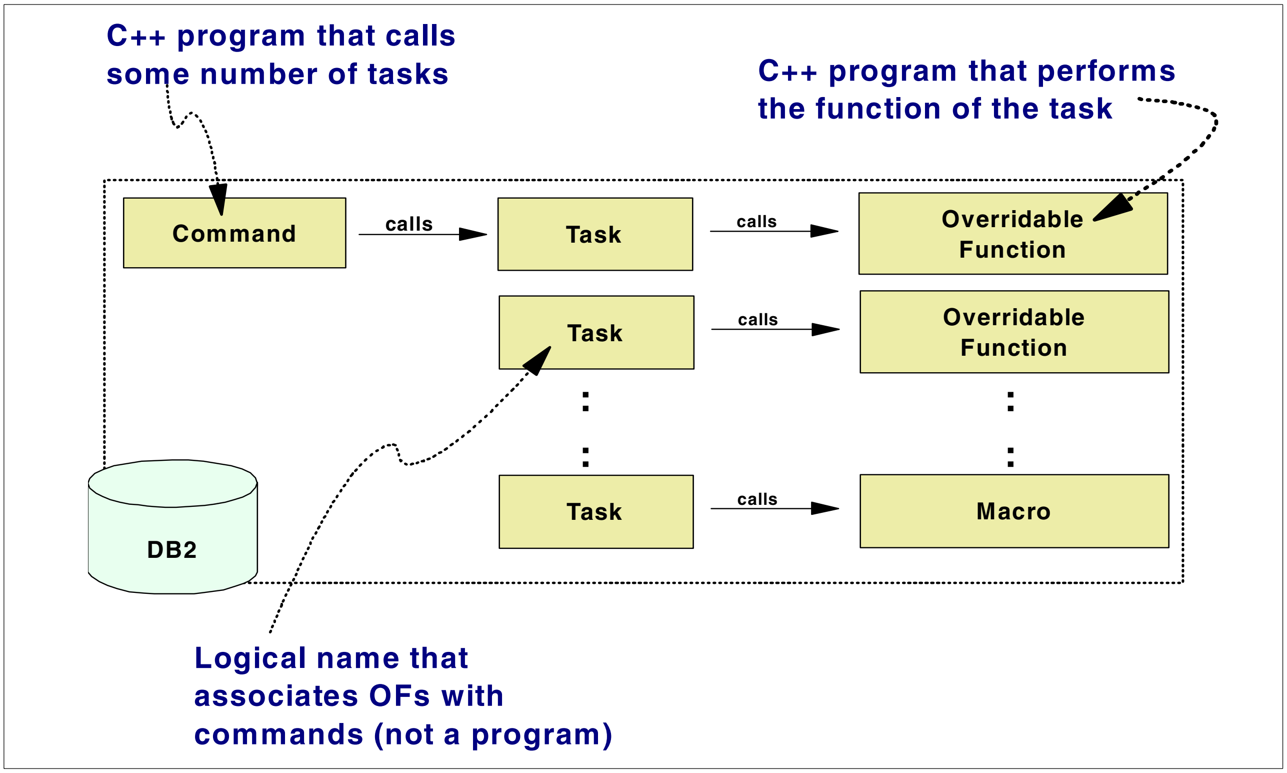

C++ program that calls some number of tasks
C++ program that performs the function of the task
Command
calls
Task
calls
Overridable Function
Task
calls
Overridable Function
Task
calls
Macro
DB2
Logical name that associates OFs with commands (not a program)

Figure 61. Overview of Commands, Tasks, and Overridable Functions

3.4.1 Commands and Overrideable Functions

Commands and Overrideable Functions are currently implemented in C++ shared objects.

The C++ source from your Net.Commerce application will have to be recompiled on WebSphere for OS/390. You place the *.so file and the *.x file in the /usr/lpp/CommerceSuite/lib directory by default. It is advisable, though, that you place them in a user code directory such as /u/code. You then add this directory to the LIBPATH statement in the <CommerceSuite instance name.envvars> file, for example:

```
PATH=/bin:.:/usr/lpp/internet/bin
SHELL=/bin/sh
TZ=EST5EDT
LANG=en_US
NLSPATH=/usr/lpp/CommerceSuite/msg/%L/%N:/usr/lpp/NetCommerce/msg/en_US/%N
LIBPATH=/usr/lpp/internet/bin:/u/ocde;/usr/lpp/CommerceSuite/lib
STEPLIB=CMN.V312.SCMNLMOD:MYCODE.LOADLIB
```

You can optionally copy your C++ Commands and Overrideable Functions from the HFS to OS/390 data sets and have CommerceSuite access them from OS/390 data sets rather than the HFS.

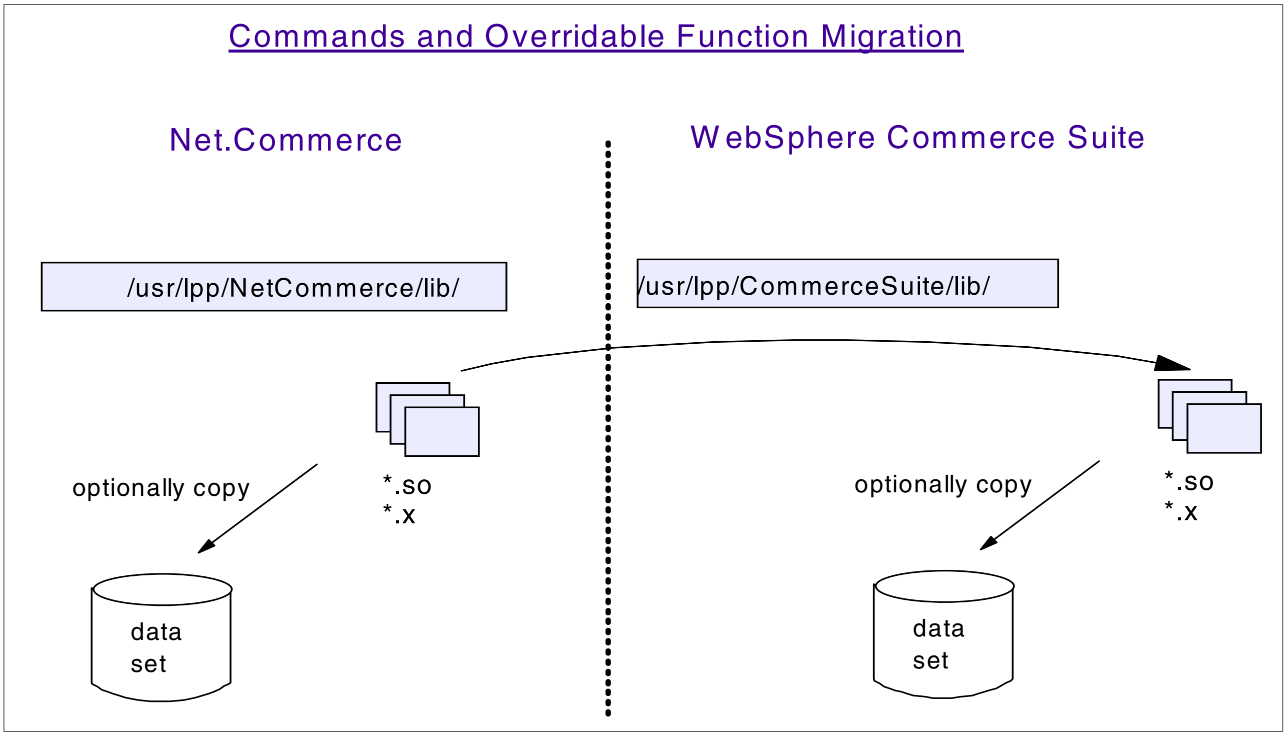


Figure 62. Migration of Net.Commerce Commands and Overrideable Functions

When migrating the Net.Commerce database to WebSphere Commerce Suite, you will insert information into the WebSphere Commerce Suite CMDS table (among others) on OS/390. See Chapter 4, “Migrating the database” on page 109 for our recommendations on how to migrate your existing Net.Commerce database to WebSphere Commerce Suite.

3.4.1.1 MAX(REFNUM) in the CMDS table

In the Net.Commerce default databases (including demomall), the maximum value for REFNUM in the CMDS table is lower than the maximum value for REFNUM in the CMDS table on WebSphere Commerce Suite. This means that any custom-written Commands that you have registered on Net.Commerce will need to be re-registered with different REFNUM values on WebSphere Commerce Suite for OS/390.

3.4.1.2 MAX(REFNUM) in the OFS table

From looking at the OFS table, we see that OS/390 WebSphere Commerce Suite has more Overrideable Functions than the Net.Commerce implementation, so you will have to re-register your OFS with higher (different) REFNUM values than those used on Net.Commerce.

Edit the OFS table to change the REFNUM. This can be achieved with SQL, such as:

```
update <owner>.OFS set REFNUM=xxx where DLL_NAME='<command_name>.so'
```

You can find the reference number (REFNUM) for the WebSphere Commerce Suite Command from an SQL query, such as:

```
select * from <owner>.OFS where DLL_NAME='<command_name>.so'
```

3.4.2 Tasks

There are three kinds of Tasks:

- Process Task – This is a Task that runs some program to do work, such as fetch the base price of a product, or determine the inventory level.
- View Task – This is a Task that runs a Net.Data macro.
- Error Task – This is actually a special kind of View Task that is associated with Process Tasks. If the process encounters an error, this Task is called so that the shopper can be presented with a screen informing them of the error.

User-written Tasks will be migrated to WebSphere Commerce Suite when you port your Net.Commerce database as described in Chapter 4, “Migrating the database” on page 109.

There are many tables associated with Tasks, such as MACROS, TASKS, TASK_MER_OF, and OFS. You should review each of these tables to ensure that the integrity of your Commands, Tasks, and Overridable Functions is maintained during the migration.

3.4.2.1 MAX(TKRFNBR) in the TASKS table

Similarly, MAX(TKRFNBR) is lower in the Net.Commerce TASKS table, so you will have to re-register your custom-written Tasks in the WebSphere Commerce Suite TASKS table with higher TKRFNBR numbers than used in Net.Commerce.

3.4.2.2 MAX(TKRFNBR) in the TASK_MER_OF table

Similarly, MAX(TKRFNBR) is lower in Net.Commerce TASK_MER_OF tables, so you will have to reassign your custom-written Overrideable Functions to their Tasks in the TASK_MER_OF table with higher TKRFNBR numbers.

3.4.2.3 MAX(TKRFNBR) in the MACROS table

MAX(TKRFNBR) is also lower in the Net.Commerce MACROS table, so you will have to reassign your custom written Tasks to their Net.Data macros in the MACROS table with higher TKRFNBR numbers.

Chapter 4. Migrating the database

Net.Commerce (NC) and WebSphere Commerce Suite (WCS) both use the DB2 database to access, manipulate, and store data and logical structures. Therefore, the process of Net.Commerce migration is largely a matter of DB2 data migration.

Information on the database schema for the Net.Commerce and WebSphere Commerce Suite systems can be obtained from the online help provided with the respective products. This online help can be accessed via the browser by specifying the following URL:

```
http://<host_name>/nchelp/
```

4.1 Overview

There are several methods by which the Net.Commerce 3.1.2 database structures can be migrated to a new WebSphere Commerce Suite 4.1 system. Within this document we will describe only one possible path. Major variations to this path could be caused by specific user requirements, such as:

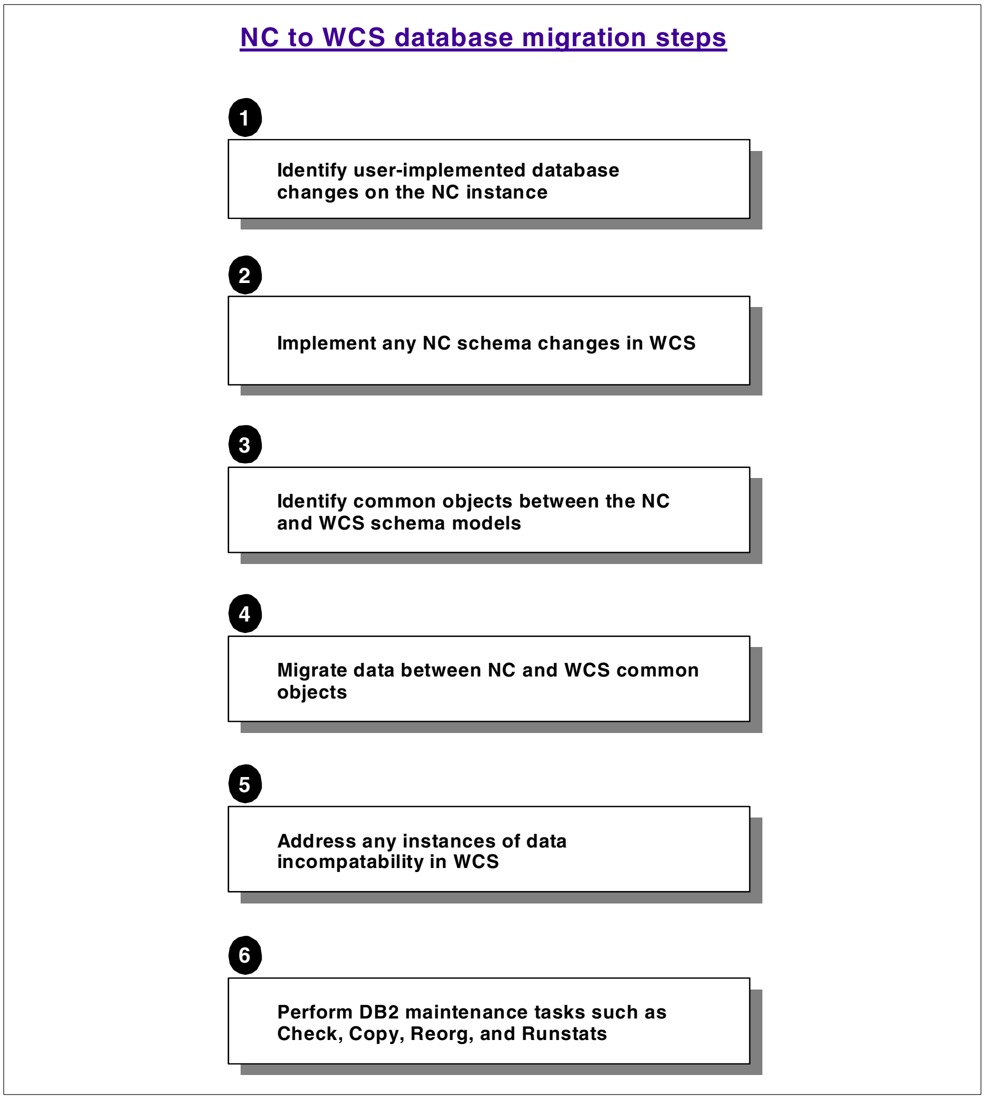
- The need to maintain separate, operational NC and WCS systems simultaneously until the WCS system is deemed ready for production
- The need to place the NC and WCS systems on separate LPARs for the migration, or to maintain both on a common OS/390 operating system
- Whether the NC and WCS systems share a common DB2 subsystem or are on separate DB2 subsystems

Each of these choices has significant ramifications for the NC to WCS migration scenario.

For the purposes of the NC to WCS migration documented here, the following environmental scenario was used:

- A single OS/390 LPAR was used to contain both NC and WCS
- Both NC and WCS shared a single DB2 V6 subsystem
- Separation of the NC and WCS objects within a single DB2 system was accomplished by using different database and creator IDs for each instance.

Regardless of the migration environment decisions you make, there are several database migration steps you must perform in an NC to WCS migration, as you can see in Figure 63 on page 110.



NC to WCS database migration steps

1. Identify user-implemented database changes on the NC instance
2. Implement any NC schema changes in WCS
3. Identify common objects between the NC and WCS schema models
4. Migrate data between NC and WCS common objects
5. Address any instances of data incompatability in WCS
6. Perform DB2 maintenance tasks such as Check, Copy, Reorg, and Runstats

Figure 63. DB2 migration overview

4.1.1 Migration steps

Following are the details of the numbered steps shown in Figure 63.

1. Identify what changes the user application has made on the existing Net.Commerce instance. This would include such things as new tables, views, referential constraints, macros, functions, and table columns that have been added to existing Net.Commerce objects. New and modified objects will need to be migrated to the WCS target instnace. Note that user-added executables, such as macros and functions, while actually maintained as data within a table, may require different treatment for migration than purely informational data such as products and customers.
2. Recreate all identified user-required NC schema changes upon the WCS instance using the required DB2 Create and Alter data definition language (DDL) statements. This will ensure that all schema objects required by the user's NC implementation will be reflected in the new WCS instance.

3. Identify all common schema objects that exist in both the NC and WCS schema implementations. The resulting set of common tables, shared by both systems, makes up the set of data that it is possible to migrate between the two instances. This set includes all product-implemented tables that are common between NC and WCS, as well as all user modifications and additions to those schemas.
4. Migrate the data from NC to WCS for those objects that are common to both schemas. This requires unloading the data from the NC objects and reloading that data into the corresponding WCS objects. This feat can be accomplished several ways, though the use of the DB2 Load utility is recommended due to its superior performance to SQL insert for large amounts of data, as well as user control of referential integrity checking, image copies, and statistics collection. The DSNTIAUL sample unload program that is shipped with DB2 is a convenient way to unload data and generate the Load utility syntax and files needed to reload that data into an identical DB2 table.
5. The next step can be the most trying. After attempting to migrate the data from NC to WCS, it is possible that some NC data will be found to be incompatable with the WCS instance. This incompatability is usually due to rows that are disallowed due to duplicate index violations. In most cases, this is not really a problem, but those disallowed rows must be examined before such a verdict can be reached. Where the incompatability is most likely to be a problem is with tables that contain executable items that the user has installed/registered on their NC system. This includes such items as macros, OFSes, and commands. Experience has shown this to be due to the unique identifiers associated with these user objects in NC being in conflict with new IBM-implemented objects in WCS.
6. Finally, after migrating the NC data and any schema changes to the target WCS system, you will need to perform some DB2 housekeeping tasks on the migrated system data. Primarily, this includes running the CHECK utility to verify that the data adheres to referential constraints, making copies of the data to ensure future integrity of the data using the COPY utility, reorganizing the data for optimal performance during SQL access using the REORG utility, and collecting new statistics with the RUNSTATS utility so that optimal access path selection can be determined by DB2 at query bind time.

4.1.2 Identify user-implemented database changes on the NC source

The first thing you must do is identify what, if any, user modifications to the NC schema will need to be migrated to the target WCS instance. To simplify this identification process as much as possible, a comparison of the existing NC schema to a pristine WCS schema is preferred.

Both WCS and Net.Commerce supply a base set of objects that are required for their operation, which we will refer to as the BASEMALL. The BASEMALL provides all the database objects and descriptors which are the minimal requirements for the operation of NC and WCS, as provided by IBM. To identify what, if any, deviation there is between the user's source NC schema and that of the WCS BASEMALL, we will use SQL joins between these two NC and WCS schemas.

Using the NCCONFIG interface, create a WCS instance using a different database and creator ID than those used in the NC instance. Using a different database name is recommended due to the fact that the creation scripts within

NC and WCS first perform an SQL Drop Database statement, and we would not want to drop both databases when creating a new instance. Different creator IDs are required for our scenario since Creator_id.Table_name must be a unique combination within a single DB2. The created WCS instance need not be operational in any way. All that is needed is to create the schema for comparison purposes.

For the instance containing the user application, the creator ID 'NETCDBU' was used, while for the WCS BASEMALL installation, the creator ID 'CMNSRV' was used. These SQL constants will need to be modified in the following queries for your particular needs.

Note that table names beginning with the characters IC, ET, NCET, and NCPAY, have been excluded from the subsequent queries. During investigation, all such tables and views were found to be representatives of the Payment Server and Product Advisor, which are not part of the described migration procedure.

4.1.2.1 Query to locate new tables and views

The following query and sample output is an outer join between the SYSIBM.SYSYSTABLES catalog table entries concerned with the NC user instance and the entries of the WCS BASEMALL instance that was created for comparison purposes. By using an outer join, tables and views that exist in one system and not the other can be distinguished.

Examining the following SQL query and output, it is evident that three tables exist in the NC312 instance that do not exist in the WCS4 instance. These three tables are user additions to the base system, which need to be recreated and repopulated by the migration.

This query could be easily changed to reveal only those three tables by deleting the 'OR NC312.NAME IS NULL' clause from the query. The rows included by this predicate represent those tables that exist in WCS and not in the NC system. This was left for informational purposes to show how the schema for WCS has grown since NC.

```
--------------------------------------------------------------------------------
 SELECT
       NC312.NAME AS NC3_TBL, WCS4.NAME AS WCS4_TBL,
       NC312.TYPE AS NC3_TYP, WCS4.TYPE AS WCS4_TYP,
       NC312.COLCOUNT AS NC3_COLS, WCS4.COLCOUNT AS WCS4_COLS
   FROM (SELECT * FROM SYSIBM.SYSTABLES A
         WHERE A.CREATOR='CMNSRV'
         AND A.NAME NOT LIKE 'IC%'
         AND A.NAME NOT LIKE 'ET%'
         AND A.NAME NOT LIKE 'NCET%'
         AND A.NAME NOT LIKE 'NCPAY%'
        ) WCS4
   FULL  OUTER JOIN
        (SELECT * FROM SYSIBM.SYSTABLES B
         WHERE B.CREATOR='NETCDBU'
         AND B.NAME NOT LIKE 'IC%'
         AND B.NAME NOT LIKE 'ET%'
         AND B.NAME NOT LIKE 'NCET%'
         AND B.NAME NOT LIKE 'NCPAY%'
        ) NC312
   ON   WCS4.NAME=NC312.NAME
   WHERE WCS4.NAME IS NULL OR NC312.NAME IS NULL;
---------+---------+---------+---------+---------+---------+---------+---------+--------
NC3_TBL             WCS4_TBL            NC3_TYP  WCS4_TYP NC3_COLS  WCS4_COLS
---------+---------+---------+---------+---------+---------+---------+---------+--------
------------------  ALCHARGE            -------  T        --------          9
------------------  AORDERS             -------  T        --------          4
------------------  APAYMENTS           -------  T        --------          6
------------------  AUCPROFILE          -------  T        --------         20
```

```
------------------  AUCTINFO            -------  T        --------        37
------------------  BEHAVIOR            -------  T        --------         3
------------------  BIDRULE             -------  T        --------        12
------------------  BIDTABLE            -------  T        --------        26
------------------  BILLITEMS           -------  T        --------         4
------------------  BROADCAST           -------  T        --------         2
------------------  BUACCTDET           -------  T        --------         9
------------------  BUADDR              -------  T        --------        13
------------------  BUCONT              -------  T        --------        11
------------------  BUYORG              -------  T        --------         6
------------------  CACHLOG             -------  T        --------         4
------------------  CCARDINFO           -------  T        --------         2
------------------  CCCHECK             -------  T        --------         5
------------------  CERT_X509           -------  T        --------         8
------------------  CIILIST             -------  T        --------         2
------------------  CMBITEMS            -------  T        --------         4
------------------  CMD_ACCESS          -------  V        --------         7
------------------  CMD_PARAM           -------  T        --------         3
------------------  CMD_TASK            -------  T        --------         2
------------------  CMDS_MER            -------  T        --------         9
------------------  CONTENTS            -------  T        --------         3
------------------  COUNTCODE           -------  T        --------         4
------------------  CPENDORDER          -------  T        --------         5
------------------  CURMSGID            -------  T        --------         1
------------------  CYMERCHANT          -------  T        --------        10
------------------  CYPAYMTHD           -------  T        --------        21
------------------  DAEMON              -------  T        --------         3
------------------  DBDELIVERY          -------  T        --------         2
------------------  DBDTFREQ            -------  T        --------         2
------------------  DBMEAS              -------  T        --------        19
------------------  DBMEASFM            -------  T        --------        12
------------------  DBTEMPLATE          -------  T        --------         4
------------------  DBVIEW              -------  T        --------         5
------------------  DISCUSSION          -------  T        --------        16
------------------  ECEOUT              -------  T        --------         6
------------------  ECETMPT             -------  T        --------         6
------------------  ECTPTITM            -------  T        --------         7
------------------  FREEGIFT            -------  T        --------         4
------------------  GL_BEHRATE          -------  T        --------         5
------------------  HMBITEMS            -------  T        --------         7
------------------  IILIST              -------  T        --------         5
------------------  LASTCG              -------  T        --------         4
------------------  LASTPR              -------  T        --------         5
------------------  LIKECG              -------  T        --------         4
------------------  LIKEPR              -------  T        --------         5
------------------  MBINFO              -------  T        --------         5
------------------  MCCARDINFO          -------  T        --------         2
------------------  MCSPINFO            -------  T        --------        10
------------------  MCSPLOCK            -------  T        --------         3
------------------  MEBUADDINFO         -------  T        --------         7
------------------  MEBUCNTINFO         -------  T        --------         4
------------------  MERPAYINFO          -------  T        --------         7
------------------  MERPROFILE          -------  T        --------         6
------------------  MESSAGES            -------  T        --------        14
------------------  METHDES             -------  T        --------         6
------------------  METHODS             -------  T        --------         4
------------------  MSGTYPES            -------  T        --------         4
------------------  NSLOG               -------  T        --------         6
------------------  NSQUEUES            -------  T        --------         6
------------------  NVSTORAGE           -------  T        --------         4
------------------  OBJREL              -------  T        --------        15
------------------  OBJTYPE             -------  T        --------         6
------------------  OBRELTYP            -------  T        --------         8
------------------  OF_PARAM            -------  T        --------         3
------------------  OF_TASK             -------  T        --------         2
------------------  OIBTB               -------  T        --------        15
------------------  ONLOG               -------  T        --------         4
------------------  ONQUEUE             -------  T        --------         2
------------------  ONSLOG              -------  T        --------         2
------------------  ORCOMMENT           -------  T        --------         4
------------------  ORDBTB              -------  T        --------        17
------------------  ORDERBID            -------  T        --------        26
------------------  ORDERMSG            -------  T        --------         5
------------------  ORDERTMPL           -------  T        --------         3
------------------  ORDISTAT            -------  T        --------        20
------------------  ORDOPTIONS          -------  T        --------         3
------------------  ORDPAYINFO          -------  T        --------         2
------------------  ORDSTAT             -------  T        --------        17
```

```
------------------  PARAMS              -------  T        --------          6
------------------  PAYINFO             -------  T        --------         12
------------------  PAYOPTIONS          -------  T        --------          6
------------------  PAYSRVRURL          -------  T        --------          2
------------------  PAYSTATUS           -------  T        --------         33
------------------  PAYSYNCH            -------  T        --------          4
------------------  PCARD               -------  T        --------          3
------------------  PREFERENCE          -------  T        --------          2
------------------  PRODIMAGE           -------  T        --------          6
------------------  PROFILES            -------  T        --------          6
------------------  PSESSION            -------  T        --------          7
------------------  PSHIPRULE           -------  T        --------         12
------------------  PTAXRULE            -------  T        --------         13
------------------  PURCG               -------  T        --------          4
------------------  PURPR               -------  T        --------          5
------------------  RECCG               -------  T        --------          4
------------------  RECIPIENTS          -------  T        --------          5
------------------  RECPR               -------  T        --------          5
------------------  REPORT              -------  T        --------         12
RRRNVPAIR           ------------------  T        -------          4 --------
RRRORDERS           ------------------  T        -------         36 --------
RRRORDS             ------------------  T        -------         38 --------
------------------  SCHCONFIG           -------  T        --------         12
------------------  SCHORDERS           -------  T        --------          2
------------------  SCHSTATUS           -------  T        --------         11
------------------  SETBATCH            -------  T        --------          3
------------------  SETMERCH            -------  V        --------          5
------------------  SETPROFILE          -------  T        --------          2
------------------  SETSTATUS           -------  V        --------         33
------------------  SHBTB               -------  T        --------          8
------------------  SHIPJURST           -------  T        --------          4
------------------  SHIPRULE            -------  T        --------         12
------------------  SHOPPERPAY          -------  T        --------          3
------------------  SHPINTR             -------  T        --------          7
------------------  STAGLOG             -------  T        --------         24
------------------  STAGSKIP            -------  T        --------          2
------------------  STBRCODES           -------  T        --------          2
------------------  STBRWSER            -------  T        --------          4
------------------  STCONF              -------  T        --------          8
------------------  STDCNTRY            -------  T        --------          2
------------------  STDOMAIN            -------  T        --------         10
------------------  STERROR             -------  T        --------         10
------------------  STFLNAME            -------  T        --------         12
------------------  STMERCNF            -------  T        --------          3
------------------  TASKRSET            -------  T        --------          6
------------------  TAXCODE             -------  T        --------          5
------------------  TAXINFO             -------  T        --------         11
------------------  TAXJURST            -------  T        --------          4
------------------  TAXRULE             -------  T        --------         13
------------------  TBSPMAP             -------  T        --------         36
------------------  TEMPLATE            -------  T        --------          3
------------------  UTMERINFO           -------  T        --------          7
------------------  UTREGIP             -------  T        --------          6
------------------  WTAXMERINFO         -------  T        --------          7
DSNE610I NUMBER OF ROWS DISPLAYED IS 136
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

4.1.2.2 Query to locate changed tables and views

The following query is a join of the same SYSIBM.SYSTABLES catalog entries as in the previous query. In this case, however, the purpose is to determine whether the user has changed any of the columns or referential constraints in the existing Net.Commerce tables. This query is not definitive, and is largely informational. It compares some basic features of tables that exist in both schemas, and displays only those where some difference exists. What is worth noticing are any cases where the NC schema shows a higher count than its WCS counterpart for some feature. This would certainly be due to some user-implemented change, since WCS added to the NC schema and no items where deleted.

If user-implemented table changes are found, they must be investigated for migration to WCS. The following query will help to identify specific user additions to the NC schema. In cases where an NC table has been modified, and that table does not exist on the WCS instance, you should investigate whether the modifications are required for a different table and/or whether the migrated application requires some redesign.

Note that if this query is executed as a join between the user's NC system and a base NC system, then *any* differences seen are surely due to user modifications of the schema. While not shown here, this is highly recommended. For our test migration, no such modifications were found. Schema differences seen in the following query, then, are entirely due to changes made in the WCS schema.

```
SELECT
        NC312.NAME AS TBLNAME,
        NC312.TYPE AS N3_TYP, WCS41.TYPE AS W4_TYP,
        NC312.COLCOUNT AS N3_COLS, WCS41.COLCOUNT AS W4_COLS,
        NC312.PARENTS AS N3_PAR, WCS41.PARENTS AS W4_PAR ,
        NC312.CHILDREN AS N3_CHILD, WCS41.CHILDREN AS W4_CHILD
    FROM SYSIBM.SYSTABLES WCS41 ,
         SYSIBM.SYSTABLES NC312
    WHERE WCS41.NAME=NC312.NAME
      AND NC312.CREATOR='NETCDBU'
      AND WCS41.CREATOR='CMNSRV'
      AND (
              NC312.COLCOUNT^=WCS41.COLCOUNT
           OR NC312.PARENTS^=WCS41.PARENTS
           OR NC312.CHILDREN^=WCS41.CHILDREN
           OR NC312.KEYCOLUMNS^=WCS41.KEYCOLUMNS
          )
      AND WCS41.NAME NOT LIKE 'IC%'
      AND WCS41.NAME NOT LIKE 'ET%'
      AND WCS41.NAME NOT LIKE 'NCET%'
      AND WCS41.NAME NOT LIKE 'NCPAY%'
   ;
---------+---------+---------+---------+---------+---------+---------+---------+--------+
TBLNAME         N3_TYP  W4_TYP    N3_COLS  W4_COLS  N3_PAR  W4_PAR  N3_CHILD  W4_CHILD
---------+---------+---------+---------+---------+---------+---------+---------+--------+
CMDS                T       T           9        9       0       0         3         5
SHOPPINGS           T       T           9       12       2       3         0         0
CATEGORY            T       T          13       14       1       1         6        16
CATEGORY_PUB        V       V          13       14       1       0         0         0
USRTRAFFIC          T       T          16       16       1       1         0         1
DISCCALC            T       T          10       11       4       4         0         0
SHIPTO              T       T          21       21       4       4         0         4
TASKS               T       T           5        5       0       0         4         7
KEYS                T       T           4        5       0       0         0         0
RATE                T       T           7        9       1       1         0         0
MALL                T       T          12       13       0       0         0         0
MERCHANT            T       T          30       31       1       1        35        68
SHOPPER             T       T          17       17       0       0         9        25
SHADDR              T       T          30       30       1       1         2         4
TAXPRCODE           T       T           4        6       1       2         2         2
PRODUCT             T       T          32       34       3       4         6        15
PRODUCT_PUB         V       V          32       34       0       0         0         0
ORDERS              T       T          17       20       3       3         3        19
ORDER_COMP          V       V          17       20       0       0         0         0
ORDER_PEND          V       V          17       20       0       0         0         0
OFS                 T       T           7        7       0       0         2         4
DSNE610I NUMBER OF ROWS DISPLAYED IS 21
```

4.1.2.3 Query to locate added columns

The following query is a join of the same SYSIBM.SYSTABLES catalog entries as in the previous query. In this case, however, the purpose is to determine whether the user has changed any of the columns or referential constraints in the existing Net.Commerce tables. This query is not definitive, and is largely informational. It compares some basic features of tables that exist in both schemas, and displays only those where some difference exists. What is worth noticing are any cases where the NC schema shows a higher count than its WCS counterpart for some feature. This would certainly be due to some user-implemented change, since WCS added to the NC schema and no items where deleted.

If user-implemented table changes are found, they must be investigated for migration to WCS. The following query will help to identify specific user additions to the NC schema. In cases where an NC table has been modified, and that table does not exist on the WCS instance, you should investigate whether the modifications are required for a different table and/or whether the migrated application requires some redesign.

Note that if this query is executed as a join between the user's NC system and a base NC system, then *any* differences seen are surely due to user modifications of the schema. While not shown here, this is highly recommended. For our test migration, no such modifications were found. Schema differences seen in the following query, then, are entirely due to changes made in the WCS schema.

```
SELECT
        NC312.NAME AS TBLNAME,
        NC312.TYPE AS N3_TYP, WCS41.TYPE AS W4_TYP,
        NC312.COLCOUNT AS N3_COLS, WCS41.COLCOUNT AS W4_COLS,
        NC312.PARENTS AS N3_PAR, WCS41.PARENTS AS W4_PAR ,
        NC312.CHILDREN AS N3_CHILD, WCS41.CHILDREN AS W4_CHILD
    FROM SYSIBM.SYSTABLES WCS41 ,
         SYSIBM.SYSTABLES NC312
    WHERE WCS41.NAME=NC312.NAME
      AND NC312.CREATOR='NETCDBU'
      AND WCS41.CREATOR='CMNSRV'
      AND (
              NC312.COLCOUNT^=WCS41.COLCOUNT
           OR NC312.PARENTS^=WCS41.PARENTS
           OR NC312.CHILDREN^=WCS41.CHILDREN
           OR NC312.KEYCOLUMNS^=WCS41.KEYCOLUMNS
          )
      AND WCS41.NAME NOT LIKE 'IC%'
      AND WCS41.NAME NOT LIKE 'ET%'
      AND WCS41.NAME NOT LIKE 'NCET%'
      AND WCS41.NAME NOT LIKE 'NCPAY%'
   ;
```

```
---------+---------+---------+---------+---------+---------+---------+---------+--------
TBLNAME             N3_TYP  W4_TYP  N3_COLS  W4_COLS  N3_PAR  W4_PAR  N3_CHILD  W4_CHILD
---------+---------+---------+---------+---------+---------+---------+---------+--------
CMDS                T       T             9        9       0       0         3         5
SHOPPINGS           T       T             9       12       2       3         0         0
CATEGORY            T       T            13       14       1       1         6        16
CATEGORY_PUB        V       V            13       14       0       0         0         0
USRTRAFFIC          T       T            16       16       1       1         0         1
DISCCALC            T       T            10       11       4       4         0         0
SHIPTO              T       T            21       21       4       4         0         4
TASKS               T       T             5        5       0       0         4         7
KEYS                T       T             4        5       0       0         0         0
RATE                T       T             7        9       1       1         0         0
MALL                T       T            12       13       0       0         0         0
MERCHANT            T       T            30       31       1       1        35        68
SHOPPER             T       T            17       17       0       0         9        25
SHADDR              T       T            30       30       1       1         2         4
TAXPRCODE           T       T             4        6       1       2         2         2
PRODUCT             T       T            32       34       3       4         6        15
PRODUCT_PUB         V       V            32       34       0       0         0         0
ORDERS              T       T            17       20       3       3         3        19
ORDER_COMP          V       V            17       20       0       0         0         0
ORDER_PEND          V       V            17       20       0       0         0         0
OFS                 T       T             7        7       0       0         2         4
DSNE610I NUMBER OF ROWS DISPLAYED IS 21
```

4.1.2.4 Query to locate modified columns

The following query is intended to determine which NC schema columns the user has modified. This query is a join between Net.Commerce and WCS entries in the SYSIBM.SYSCOLUMNS table, where a difference in column length, type, null status, or default status is sought.

Columns defined as type LONGVAR make up a sizeable portion of the query results due to a difference in the implicit lengths. LONGVAR is not defined as having any particular length in reality, and is limited by the database manager to what can be fit into the maximum page size. Note that on the OS/390 system a 4 K page size is used as the default for tablespaces. So when fields are added to a table, the maximum length of any LONGVAR fields in that table decreases.

There is also a case in the sample output where a character field became shorter in WCS. This, along with the shortened LONGVAR case, can cause migration problems if there is any data in the NC instance which will exceed the shorter maximum length in the WCS schema. You must check for all such instances and shorten the data entries sufficiently for insertion into the WCS instance. If this is not possible, the alternative would be in place the tablespaces into bufferpools with greater page sizes than 4 K. For the LONGVAR cases, DB2 will automatically allocate larger maximum lengths when using large page sizes. For CHAR or VARCHAR cases, however, the length specified in the required CREATE TABLE statements will need to be changed

After analyzing the other changed columns in this sample output, you should notice that they occur because a character field grew in length in WCS, or became nullable. Such changes will not cause problems in migration since existing NC data can be accomodated in the new WCS tables.

```
SELECT
        NC312.TBNAME AS TBNAME,
        NC312.NAME AS COLNAME,
        NC312.COLTYPE AS NC3_TYP,WCS41.COLTYPE AS WCS4_TYP,
        NC312.LENGTH AS NC3_LGTH,WCS41.LENGTH AS WCS4_LGTH,
        NC312.NULLS AS NC3_NULL, WCS41.NULLS AS WCS4_NULL,
        NC312.DEFAULT AS NC3_DFLT,WCS41.DEFAULT AS WCS4_DFLT
    FROM SYSIBM.SYSCOLUMNS WCS41,
         SYSIBM.SYSCOLUMNS NC312
    WHERE
          WCS41.TBCREATOR='CMNSRV'
      AND NC312.TBCREATOR='NETCDBU'
      AND NC312.TBNAME=WCS41.TBNAME
      AND NC312.NAME=WCS41.NAME
      AND (
              NC312.COLTYPE^=WCS41.COLTYPE
           OR NC312.LENGTH^=WCS41.LENGTH
           OR NC312.NULLS^=WCS41.NULLS
           OR NC312.DEFAULT^=WCS41.DEFAULT
          )
      AND WCS41.TBNAME NOT LIKE 'IC%'
      AND WCS41.TBNAME NOT LIKE 'ET%'
      AND WCS41.TBNAME NOT LIKE 'NCET%'
      AND WCS41.TBNAME NOT LIKE 'NCPAY%'
  ;
---------+---------+---------+---------+---------+---------+---------+---------+---------+--
TBNAME         COLNAME    NC3_TYP   WCS4_TYP  NC3_LGTH          NC3_NULL     NC3_DFLT
                                                      WCS4_LGTH       WCS4_NULL     WCS4_DFLT
------+--------+---------+---------+---------+---------+---------+---------+---------+------
CATEGORY      CGLDESC      LONGVAR   LONGVAR        2252      2004      Y       Y       Y       Y
CATEGORY_PUB  CGLDESC      VARCHAR   VARCHAR        2252      2004      Y       Y       Y       Y
APIS          APIDLLNAME   VARCHAR   CHAR            254       254      Y       Y       Y       Y
APIS          APIFUNCNAME  VARCHAR   CHAR            254       254      Y       Y       Y       Y
STATCODE      STANAME      CHAR      CHAR             40        40      N       Y       N       Y
TASKS         TKCOMMENT    VARCHAR   CHAR            254       254      Y       Y       Y       Y
MACROS        MAFILENAME   VARCHAR   CHAR            254       254      Y       Y       Y       Y
MALL          MHTHEAD      CHAR      CHAR            253       254      Y       Y       Y       Y
TAXPRCODE     TPCDESC      LONGVAR   LONGVAR        3984      3724      Y       Y       Y       Y
PRSPCODE      PSCODE       CHAR      CHAR              5        10      N       N       N       N
PRODUCT       PRLDESC1     LONGVAR   LONGVAR         718       696      Y       Y       Y       Y
PRODUCT       PRLDESC2     LONGVAR   LONGVAR         718       696      Y       Y       Y       Y
PRODUCT       PRLDESC3     LONGVAR   LONGVAR         718       696      Y       Y       Y       Y
PRODUCT_PUB   PRLDESC3     VARCHAR   VARCHAR         718       696      Y       Y       Y       Y
PRODUCT_PUB   PRLDESC2     VARCHAR   VARCHAR         718       696      Y       Y       Y       Y
PRODUCT_PUB   PRLDESC1     VARCHAR   VARCHAR         718       696      Y       Y       Y       Y
NC_CACHE      HTML         LONGVAR   LONGVAR        3778      4026      Y       Y       Y       Y
DSNE610I NUMBER OF ROWS DISPLAYED IS 17
```

4.1.3 Common tables of NC and WCS

The purpose here is to generate a list of the tables that are common to both the user's NC and the base WCS instance. The table members of the resulting list, combined with the user-added tables as determined earlier, comprise all the tables whose data must be migrated to the WCS target platform. The required query is a simple join between the DB2 SYSIBM.SYSTABLES catalog table entries for the NC and WCS schemas:

```
SELECT
        STRIP(NC312.CREATOR)||'.'||NC312.NAME,
        STRIP(WCS4.CREATOR)||'.'||WCS4.NAME
    FROM SYSIBM.SYSTABLES WCS4 ,
         SYSIBM.SYSTABLES NC312
    WHERE WCS4.NAME=NC312.NAME
      AND WCS4.CREATOR='CMNSRV'
      AND NC312.CREATOR='NETCDBU'
      AND NC312.TYPE='T'
      AND WCS4.NAME NOT LIKE 'IC%'
      AND WCS4.NAME NOT LIKE 'ET%'
      AND WCS4.NAME NOT LIKE 'NCET%'
      AND WCS4.NAME NOT LIKE 'NCPAY%'
    ORDER BY NC312.NAME
   ;
---------+---------+---------+---------+---------+---------+---------+---------+------

---------+---------+---------+---------+---------+---------+---------+---------+------
NETCDBU.ACC_CMDGRP           CMNSRV.ACC_CMDGRP
NETCDBU.ACC_GROUP            CMNSRV.ACC_GROUP
NETCDBU.ACC_MODE             CMNSRV.ACC_MODE
NETCDBU.ACC_USRGRP           CMNSRV.ACC_USRGRP
NETCDBU.ACCTRL               CMNSRV.ACCTRL
NETCDBU.APIS                 CMNSRV.APIS
NETCDBU.BROWSER              CMNSRV.BROWSER
NETCDBU.CATEGORY             CMNSRV.CATEGORY
NETCDBU.CATESGP              CMNSRV.CATESGP
NETCDBU.CGPRREL              CMNSRV.CGPRREL
NETCDBU.CGRYREL              CMNSRV.CGRYREL
NETCDBU.CMDS                 CMNSRV.CMDS
NETCDBU.CURRCCACHE           CMNSRV.CURRCCACHE
NETCDBU.CURRCONV             CMNSRV.CURRCONV
NETCDBU.CURRCVLIST           CMNSRV.CURRCVLIST
NETCDBU.CURRFCACHE           CMNSRV.CURRFCACHE
NETCDBU.CURRFORMAT           CMNSRV.CURRFORMAT
NETCDBU.CURRLIST             CMNSRV.CURRLIST
NETCDBU.DISCCALC             CMNSRV.DISCCALC
NETCDBU.DISCCODE             CMNSRV.DISCCODE
NETCDBU.KEYS                 CMNSRV.KEYS
NETCDBU.MACROS               CMNSRV.MACROS
NETCDBU.MALL                 CMNSRV.MALL
NETCDBU.MCUSTINFO            CMNSRV.MCUSTINFO
NETCDBU.MERCHANT             CMNSRV.MERCHANT
NETCDBU.MERCHANTTAX          CMNSRV.MERCHANTTAX
NETCDBU.MSHIPMODE            CMNSRV.MSHIPMODE
NETCDBU.NC_CACHE             CMNSRV.NC_CACHE
NETCDBU.OFS                  CMNSRV.OFS
NETCDBU.ORDERPAY             CMNSRV.ORDERPAY
NETCDBU.ORDERS               CMNSRV.ORDERS
NETCDBU.ORDPAYMTHD           CMNSRV.ORDPAYMTHD
NETCDBU.PERFLOG              CMNSRV.PERFLOG
NETCDBU.POOL_CMD             CMNSRV.POOL_CMD
NETCDBU.POOLS                CMNSRV.POOLS
NETCDBU.PRODATR              CMNSRV.PRODATR
NETCDBU.PRODDSTATR           CMNSRV.PRODDSTATR
NETCDBU.PRODPRCS             CMNSRV.PRODPRCS
NETCDBU.PRODSGP              CMNSRV.PRODSGP
NETCDBU.PRODUCT              CMNSRV.PRODUCT
NETCDBU.PROXY                CMNSRV.PROXY
NETCDBU.PRSPCODE             CMNSRV.PRSPCODE
NETCDBU.RATE                 CMNSRV.RATE
NETCDBU.RJRNVPAIR            CMNSRV.RJRNVPAIR
NETCDBU.RJRORDERS            CMNSRV.RJRORDERS
NETCDBU.RJRORDS              CMNSRV.RJRORDS
NETCDBU.SCALE                CMNSRV.SCALE
NETCDBU.SETCURR              CMNSRV.SETCURR
NETCDBU.SG_STORES            CMNSRV.SG_STORES
NETCDBU.SHADDR              CMNSRV.SHADDR
NETCDBU.SHIPMODE            CMNSRV.SHIPMODE
NETCDBU.SHIPPING            CMNSRV.SHIPPING
NETCDBU.SHIPTO              CMNSRV.SHIPTO
NETCDBU.SHOPDEM             CMNSRV.SHOPDEM
NETCDBU.SHOPGRP             CMNSRV.SHOPGRP
NETCDBU.SHOPPER             CMNSRV.SHOPPER
NETCDBU.SHOPPINGS           CMNSRV.SHOPPINGS
NETCDBU.STATCODE            CMNSRV.STATCODE
NETCDBU.STRCGRY             CMNSRV.STRCGRY
NETCDBU.TASK_MER_OF         CMNSRV.TASK_MER_OF
NETCDBU.TASKS               CMNSRV.TASKS
NETCDBU.TAXCGRY             CMNSRV.TAXCGRY
NETCDBU.TAXPRCODE           CMNSRV.TAXPRCODE
NETCDBU.TEMPADDR            CMNSRV.TEMPADDR
NETCDBU.TEMPSHOP            CMNSRV.TEMPSHOP
NETCDBU.TMPORDER            CMNSRV.TMPORDER
NETCDBU.USRTRAFFIC          CMNSRV.USRTRAFFIC
DSNE610I NUMBER OF ROWS DISPLAYED IS 67
```

4.1.4 Determine DB2 database parameters for WCS

Before moving forward, this is a good time to make some decisions that will affect the performance, monitoring capability, maintenance scenarios, ease-of-tuning, and manageability of your resulting WCS. The key choice is whether to use the same DB2 database settings that your NC system used, or to change some or all of your DB2 settings for the new WCS system.

Following are basic DB2 customization options that can have great impact on your new WCS system.

4.1.4.1 Stogroups

You must determine what set (or sets) of physical disk volumes you will store your WCS database objects in. Within DB2 for OS/390, there are two methods used to define the physical storage location of objects: the user-defined and stogroup methods. With the user-defined method, you must create and manage the VSAM linear data sets yourself. In earlier days, there were some benefits of user-defined data sets, but today, the vast majority of users use stogroup definitions that allow DB2 to control the management and placement of data sets.

It is recommended that you create at least one stogroup that will contain the volumes used exclusively to store your database objects. Beyond this minimum recommendation, you can also decide to define multiple stogroups with which you can implement schemes to assure the segregation of data sets across volumes. An example would be to create different stogroups for tables and indexes, whereby indexes are never resident on the same volume as the tablespace they reference. The benefit of such a scheme would be to reduce I/O contention during index access of a table.

4.1.4.2 Tablespaces

A tablespace in DB2 for OS/390 is a construct to map logical tables to physical VSAM data sets. A tablespace can contain one or more tables, while a table must be defined to be in a single tablespace. What you must decide is how you wish to spread your tables across physical tablespaces. The extremes possible are that you could have one tablespace within which all your tables reside, or you could have each table dedicated to its own private tablespace.

There are a fair number of factors that can be considered when deciding how to define tablespaces for tables. The most notable are table size and how the table is accessed by the application. Large tables should be provided a private tablespace, and very large tables should be provided a partitioned tablespace. Small tables can be defined to share a common tablespace, or each can have its own private tablespace. When multiple tables will share a tablespace, the tablespace should be defined as segmented.

For our particular migration, we chose to create each table in its own private tablespace. This allows for the maximum amount of flexibility in the future, although it also means you will have the largest number of data sets to define and manage.

Another factor to consider in your DB2 design is whether you will require tables having rows longer than 4 K. The default for tablespaces is that they are created in the BP0 buffer pool, which is a 4 K page size buffer pool. In DB2 for OS/390, a table row cannot span pages, and so, the maximum record length is bounded by page size. After allowing for page and record headers, the largest row that will fit into a 4 K page is 4056 bytes. If you require longer rows, you will need to create the corresponding tablespace(s) in 32 K buffer pools, where a maximum record length of 32714 bytes is possible. Note that in DB2 for OS/390 Version 6, in addition to 4 K and 32 K page sizes, 8 K and 16 K page sizes are also supported.

4.1.4.3 Buffer pools

A final consideration is determining your DB2 buffer pool usage. The DB2 accounting and statistics traces accumulate statistics for each buffer pool separately. Significant tuning can also be done for buffer pools for such things as memory usage, write thresholds, sequential pre-fetch, hiperpool usage, and so on. Therefore, from a performance monitoring and tuning standpoint, you may wish to segregate objects into different buffer pools. A very common methodology is to divide your database objects into four buffer pools: read-only tablespaces, read-only indexes, read-write tablespaces, and read-write indexes occupying different buffer pools. This method allows for significant tuning, diagnostic, and monitoring capabilities.

4.2 Methods to migrate data

There are several options for how to actually migrate the data from a Net.Commerce instance to a WebSphere Commerce Suite instance. This section describes these options and helps you to decide which one best suits your skills and environment.

4.2.1 Insert with subselect

The first method one may think of to move data from one table to another of identical, or almost identical, definition would be a simple SQL insert statement with a subselect. Upon reflection, though, this is not a recommended path for migration for several reasons:

- To insert data from the NC schema to the WCS schema, both must reside on the same DB2 system. There is no capability to have the Commerce instances on different OS/390 or DB2 systems.
- In cases where tables differ in definition, the SQL insert query must be customized.
- SQL insert is not the best performing option. This is due to the costs of insertion of individual rows, individual insertions for index entries, and

individual verification of referential constraints. For large amounts of data, this can be a large performance consideration

- And perhaps most importantly, due to the many referential constraints in the WCS schema, it is difficult to determine the order in which tables must be populated. With SQL, referential integrity is always maintained and so, a child table cannot be populated until its parent is.

4.2.2 DB2 for OS/390 DSNTIAUL Sample and Load utility

The DB2 for OS/390 DSNTIAUL sample application can be used to unload data and the DB2 for OS/390 Load utility used to populate tables using that unloaded data. There are a few good justifications for using this method for data migration:

- The DB2 for OS/390 Load utility is the most efficient way to populate tables where the number of rows is significant. This is primarily because the DB2 Load utility has a far more efficient interface with DB2 than SQL insert and because logging can be disabled when the Load utility is used.
- Data migration may be complicated when referential constraints exist on the tables being populated. This is because you must ensure that a parent record exists before attempting to insert a child record that is dependent on that parent. Determining the order of table population so as to avoid referential constraint problems can be difficult when you have many relations linking multiple tables. The DB2 for OS/390 Load utility, however, can optionally ignore referential constraint checking during the loading process. After loading the entire set of tables, referential constraint checking can then be performed all at once.
- The DSNTIAUL unload and the Load utility invocation to reload that data into a table can be performed on different OS/390 and/or DB2 systems.

The DB2 for OS/390 Load utility requires a rather specific format for its input data set. Fortunately, DB2 for OS/390 supplies a sample unload program called DSNTIAUL. DSNTIAUL is an assembler language sample program that unloads tables into sequential data sets that can later be reloaded into the same DB2 table or into another DB2 table using the DB2 for OS/390 Load utility. As well as unloading data, DSNTIAUL also creates the Load utility syntax required to invoke the DB2 for OS/390 Load utility with the unloaded data sets.

Upon investigation, the DSNTIAUL application can also be used to unload tables on remote DB2 systems via DRDA. For cases where data is to be migrated across different database platforms that are linked for remote access, this can be a very valuable tool.

To use DSNTIAUL to unload you NC data, you must next prepare the input and output data sets for the DSNTIAUL invocations. DSNTIAUL will be used to unload every table common between the user's NC and new WCS instance, including any new user tables that are to migrate. This list of tables can be generated by using the previously mentioned query (Figure 4.1.3 on page 118) after any user additions to the WCS schema have been defined. Note that the list must be in the form of 'creator_id'.'table_name' and identify the NC tables to be unloaded.

For each invocation of DSNTIAUL, you can unload up to 100 tables. Each table will be unloaded to its own unique data set, defined by the JCL DD statements SYSREC00, SYSREC01 to SYSREC99, where the numeric portion of SYSREC is determined by the position of the table being unloaded in the SYSIN input list.

The Load utility syntax for all tables unloaded will be placed into the data set defined by the SYSPUNCH DD statement. An example of the JCL that can be used to execute the DSNTIAUL unload follows, where four tables are being unloaded (SYSREC00-SYSREC03) and the list of tables being unloaded is contained in a separate file named HAUSER.SPUFI.IN(TABLIST1):

```
//UNLD1   JOB CLASS=A,MSGCLASS=X,REGION=6M,NOTIFY=&SYSUID
//UNLD    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=FF.LOAD1,UNIT=SYSDA,SPACE=(CYL,(1,10),RLSE),
//          DISP=(NEW,CATLG)
//SYSREC00 DD DSN=FF.UNLD00,SPACE=(CYL,(5,10),RLSE),DISP=(NEW,CATLG)
//SYSREC01 DD DSN=FF.UNLD01,SPACE=(CYL,(5,10),RLSE),DISP=(NEW,CATLG)
//SYSREC02 DD DSN=FF.UNLD02,SPACE=(CYL,(5,10),RLSE),DISP=(NEW,CATLG)
//SYSREC03 DD DSN=FF.UNLD03,SPACE=(CYL,(5,10),RLSE),DISP=(NEW,CATLG)
//SYSTSIN  DD  *
 DSN SYSTEM(DB2P)
 RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIAUD)                -
      LIB('DB2V610P.RUNLIB.LOAD')
 END
//SYSIN    DD DSN=HAUSER.SPUFI.IN(TABLIST1),DISP=SHR
//
```

At this point, you have not modified the contents of the existing WCS base, beyond defining any user schema additions or modifications. You should perform full image copies of all the tablespaces in the database at this point. This provides a point of consistency, back to which you can restore easily if you encounter problems during the loading of the migrated data into the tables. The DB2 Copy utility, with the FULL YES parameter, should be used to back up the newly created WCS instance.

After executing the DSNTIAUL unloads, you will need to edit the resulting Load utility syntax. There are several Load utility options that you may need to add or change in the DSNTIAUL-generated syntax for each individual Load invocation:

- Edit the table names referenced in the Load INTO TABLE parameter, if needed, so that the creator and table name refers to the OS/390 table object you wish to port the data into.
- The parameter ENFORCE NO should be added. This parameter specifies that no referential integrity checking should be undertaken during the load process. By using this parameter, you can load sets of tables without needing to order them such that parents are loaded before children. By using this parameter, the tablespace will be placed in Check Pending status. Later, you will need to use the Check utility to perform referential integrity checking and to turn off the Check Pending status.
- Provide the ERRDDN, MAPDDN, and DISCARDDN parameters and data sets. These become important in a situation where you will be loading data from the common Net.Commerce for Windows NT tables into their OS/390 siblings. Unique indexes will cause discarding of duplicate entries into these added output files. This is beneficial in the cases, such as Net.Commerce-provided commands, where the OS/390 versions are the correct ones. However, for cases where the user may have added commands that inadvertently duplicate one provided in OS/390, the duplicate will also be put into the discard data set. To discern between those entries that are rightly discarded and those that will need to be changed and inserted into the OS/390 table, you will need to perform analysis.

- Use the REPLACE and RESUME parameters appropriately. For new user tables, RESUME NO and REPLACE YES will allow you to run the Load utility (multiple times if need be for debugging) without accumulating rows due to reloading. For existing Net.Commerce tables, you will need to use RESUME YES and REPLACE NO to ensure that you do not eliminate the existing rows in the tables.
- The final requirement is to provide the required JOB, EXEC, and DD statements. The following sample job can be used as a general guideline, but you may need to change the data set names and DD names to match those required by your own scenario. Space allocations for the data sets may also need to be changed to fit the needs determined by the size of your particular situation.

```
//LOAD1   JOB CLASS=A,MSGCLASS=X,REGION=0M,NOTIFY=&SYSUID
//*
//UTIL EXEC DSNUPROC,SYSTEM=DB2P,UID='HAUSER.LOAD1'
//*
//SORTWK01  DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK02  DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK03  DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK04  DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SYSUT1    DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTOUT   DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SYSERR   DD DSN=MIGR.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND)
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSMAP   DD DSN=MIGR.SYSMAP,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSDISC DD DSN=MIGR.SYSDISC,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSREC00 DD DSN=MIGR.UNLD00,DISP=SHR
//DSNUPROC.SYSIN     DD *
LOAD DATA LOG NO INDDN SYSREC24 RESUME YES ENFORCE NO
      INTO TABLE CMNSRV.MERCHANT
   (
   MERFNBR       POSITION     (    1 )    INTEGER,
   MENAME        POSITION     (    5 )    CHAR( 80)       NULLIF(85)='?',
   MECLNAM       POSITION     (   86 )    CHAR( 30),
   MECFNAM       POSITION     (  116 )    CHAR( 30),
   MECMNAM       POSITION     (  146 )    CHAR( 30)       NULLIF(176)='?',
   MECTITLE      POSITION     (  177 )    CHAR( 30)       NULLIF(207)='?',
   MECPH1        POSITION     (  208 )    CHAR( 30),
   MECPH2        POSITION     (  238 )    CHAR( 30)       NULLIF( 268)='?',
   MECFAX        POSITION     (  269 )    CHAR( 30)       NULLIF( 299)='?',
   MECMAIL1      POSITION     (  300 )    CHAR( 254)      NULLIF( 554)='?',
   MECMAIL2      POSITION     (  555 )    CHAR( 254)      NULLIF( 809)='?',
   MEPHONE       POSITION     (  810 )    CHAR( 30),
   MEADDR1       POSITION     (  840 )    CHAR( 50)       NULLIF( 890)='?',
   MEADDR2       POSITION     (  891 )    CHAR( 50)       NULLIF( 941)='?',
   MEADDR3       POSITION     (  942 )    CHAR( 50)       NULLIF( 992)='?',
   MECITY        POSITION     (  993 )    CHAR( 30),
   MESTATE       POSITION     ( 1023 )    CHAR( 20),
   MECNTRY       POSITION     ( 1043 )    CHAR( 30),
   MEZIPC        POSITION     ( 1073 )    CHAR( 20)       NULLIF( 1093)='?',
   MESTNAME      POSITION     ( 1094 )    CHAR( 80),
   MESTDESC      POSITION     ( 1174 )    VARCHAR         NULLIF( 2176)='?',
   MESCNBR       POSITION     ( 2177 )    INTEGER         NULLIF( 2181)='?',
   METHMB        POSITION     ( 2182 )    CHAR( 254)      NULLIF( 2436)='?',
   METHEAD       POSITION     ( 2437 )    CHAR( 254)      NULLIF( 2691)='?',
   METFOOT       POSITION     ( 2692 )    CHAR( 254)      NULLIF( 2946)='?',
   METBASE       POSITION     ( 2947 )    CHAR( 254)      NULLIF( 3201)='?',
   MECUR         POSITION     ( 3202 )    CHAR( 10),
   MEFIELD1      POSITION     ( 3212 )    VARCHAR         NULLIF( 3468)='?',
   MEFIELD2      POSITION     ( 3469 )    VARCHAR         NULLIF( 3725)='?',
   MEROID        POSITION     ( 3726 )    CHAR( 36)       NULLIF( 3762)='?'
   )
```

After executing the Load utility jobs to migrate your data, you will need to check the output to verify that all has gone normally. If rows have been discarded during any of the loads, you will need to verify that these discarded rows were provided by Net.Commerce for Windows NT. Any discarded rows that are not provided by the NC base will need to be analyzed, changed, and inserted, since they represent user-created data. Return codes for the Load utilities should be no larger than four, due to the fact that tablespaces will be left in Copy and Check Pending states.

4.2.3 Procedure for problems with OFS, commands, and macro tables

After performing the Loads of all tables, carefully look at the results of each. You may notice significant discards of rows that represent user-added overridable functions, commands, or macros. We found in our sample migration that these were often due to duplicate entries being attempted in unique indexes. The cause was largely due to user additions in NC having used reference numbers that became part of the WCS product itself. With the great use of referential integrety in the Commerce products, fixing such problems is non-trivial. Merely changing the reference number for one particular entry is not possible, since all referentially related entries must also be changed. Reference numbers are also tracked within the KEYS table, which would also need to be changed.

The solution for such a scenario is to reinstall all of the user's overridable functions, commands, and macros. The load process may, however, have corrupted these tables by allowing some entries to be inserted improperly. An example would be if a parent table row were rejected due to a duplicate unique index entry while a child of that row in another table was accepted . To put things right, a recovery back to the initial WCS state is required for such tables. This can be accomplished either by dropping and redefining the WCS schema and then reloading only the required tables, or by recovering the suspect tables to the level of the full image copies you performed when installing the WCS schema. The tables that are of concern are:

- ACC_CMDGRP
- ACC_GROUP
- ACC_MODE
- ACC_USRGRP
- ACCTRL
- APIS
- BROWSER
- CMDS
- KEYS
- MACROS
- OFS
- PERFLOG
- POOL_CMD
- POOLS
- TASK_MER_O
- TASKS

4.3 Database housekeeping

Regardless of the method you use to migrate your data from NC to WCS, there are DB2 for OS/390 procedures you should perform to verify, organize, and back up the migrated objects as well as to prepare for their optimal use.

4.3.1 The Check Data utility

The Check Data utility verifies that the contents of a set of tablespaces abide by the referential constraints that are defined in the tables they contain. After migrating to WCS, if any of the tablespaces are left in Check Pending status, you must run the Check Data utility. Since referential constraints are defined in the WCS, it seems reasonable that even if tablespaces are not overtly in Check Pending status, executing the Check Data utility for all tablespaces provides a checkpoint for data assurance.

For our sample migration, a single Check Data utility job was executed. You can perform the checks using multiple utility executions, where each separate job represents one or more tablespace sets. To create the utility syntax, the tablespace list was generated by a simple SQL Select query of all tablespaces in the Net.Commerce database:

```
//CHECK   JOB CLASS=A,MSGCLASS=X,REGION=0M,NOTIFY=&SYSUID
//*
//UTIL EXEC DSNUPROC,SYSTEM=DB2P,UID='HAUSER.CHECK'
//*
//SORTWK01 DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK02 DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK03 DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTWK04 DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SORTOUT  DD UNIT=SYSALLDA,SPACE=(CYL,(50,50))
//SYSERR   DD DSN=MIGR.CHECK.ERRORS,DISP=(NEW,CATLG,DELETE),
//         UNIT=SYSDA,SPACE=(CYL,(5,1),RLSE)
//DSNUPROC.SYSIN     DD *
CHECK DATA
    TABLESPACE TEMPDB.ACCOUNTR
    TABLESPACE TEMPDB.ACCRCMDG
    TABLESPACE TEMPDB.ACCRGROU
    TABLESPACE TEMPDB.ACCRMODE
..........
    TABLESPACE TEMPDB.USRTRAFF
    TABLESPACE TEMPDB.WTAXMERI
      SCOPE ALL
```

4.3.2 Reorg/Copy or Copy utilities

After verifying the contents of the migrated database using the Check Data utility, you should back up the tablespaces using the Copy utility. You can also, optionally, reorganize the data in the tablespaces and indexes to provide an optimal starting point for performance and space utilization by using the Reorg utility. For our sample migration, we opted to perform both a reorganization and a copy of the Net.Commerce system. The Reorg and Load utilities both contain the option of generating image copy data sets in parallel to the Reorg or Load task. In this example, by using the COPYDDN option combined with the REORG utility, the elapsed time to perform the image copy and the reorganization is only slightly more than that of the reorganization task alone.

For our sample migration, the Reorg utility job and syntax was generated with REXX (see Appendix B.2, “REXX program to generate a single REORG job” on page 200) using the tablespace list as used for the Check Data utility execution:

```
//REORGTS JOB CLASS=A,MSGCLASS=X,REGION=5M,NOTIFY=&SYSUID
//R001 EXEC DSNUPROC,SYSTEM=DB2P,UID='REO001'
//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK02  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK03  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK04  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SYSPRINT DD   SYSOUT=*
//SYSCOPY DD DSN=MIGR.COPY001,DISP=(NEW,CATLG,DELETE),
//        UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE),
//        DCB=(RECFM=FB,LRECL=4096,BLKSIZE=24576,BUFNO=30)
//SYSREC  DD DSN=MIGR.RECS001,DISP=(NEW,DELETE,KEEP),
//        UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE),
//        DCB=(RECFM=FB,LRECL=4096,BLKSIZE=24576,BUFNO=30)
//SYSIN    DD   *
 REORG TABLESPACE TEMPDB.ACCOUNTR LOG NO
       SORTDATA SORTKEYS COPYDDN(SYSCOPY)
```

4.3.3 Runstats

The final recommended maintenance step for the newly migrated WCS database is to collect statistics on all the tablespace and index objects. This final step only applies if you did not do integrated statistics collection during the Reorg.

In DB2 for OS/390, the database statistics are collected using the Runstats utility. Statistics are collected and placed into the DB2 system catalog. These statistics are used during the query bind process to help determine the optimal access path for query execution. Therefore, after collecting statistics, you should always rebind static plans to assure that suboptimal access paths are not retained.

There are numerous invocation options for the Runstats utility. For our sample migration, we ran the Runstats utility set to collect all statistics on the tablespaces, tables, and indexes as shown in the following sample. We generated the Runstats utility JCL and syntax, again, using REXX (see Appendix B.4, “REXX program sample to use with REORG utility” on page 202) and an input list of all the tablespaces in the migration.

```
//STATS JOB CLASS=A,MSGCLASS=X,REGION=5M,NOTIFY=&SYSUID
//E001 EXEC DSNUPROC,SYSTEM=DB2P,UID='STAT001'
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  RUNSTATS TABLESPACE TEMPDB.ACC
```

For our sample migration, the Reorg utility job and syntax was generated with REXX using the tablespace list as used for the Check Data utility execution:

```
//REORGTS JOB CLASS=A,MSGCLASS=X,REGION=5M,NOTIFY=&SYSUID
//R001 EXEC DSNUPROC,SYSTEM=DB2P,UID='REO001'
//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK02  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK03  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTWK04  DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(15,15))
//SYSPRINT DD   SYSOUT=*
//SYSCOPY DD DSN=MIGR.COPY001,DISP=(NEW,CATLG,DELETE),
//        UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE),
//        DCB=(RECFM=FB,LRECL=4096,BLKSIZE=24576,BUFNO=30)
//SYSREC  DD DSN=MIGR.RECS001,DISP=(NEW,DELETE,KEEP),
//        UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE),
//        DCB=(RECFM=FB,LRECL=4096,BLKSIZE=24576,BUFNO=30)
//SYSIN    DD   *
 REORG TABLESPACE TEMPDB.ACCOUNTR LOG NO
       SORTDATA SORTKEYS COPYDDN(SYSCOPY)
```

Chapter 5. Using WebSphere Commerce Studio

This chapter provides a step-by-step example of creating a store, publishing to an OS/390 server, and creating a Java Server Page (JSP).

This chapter is intended to show the basics of performing these tasks. It is not a complete reference of all the features of the WCSuite functions we're illustrating. To provide that would mean recreating much of the material that's already in place. (See Appendix D, "Related publications and resources" on page 209 for information on these other sources of WebSphere Commerce Suite education and training.)

5.1 Creating a store

WCSuite provides the WebSphere Commerce Studio (referred to in this book as "Commerce Studio") application. This is a Windows application that publishes stores to a variety of WCSuite servers running on either Windows NT, UNIX, OS/400, or OS/390. Commerce Studio has a Store Creator Wizard, which allows you to quickly build a sample store that you can modify for your own business needs.

5.1.1 390 store model

There is a 390-specific store model that can be downloaded from the internet at:

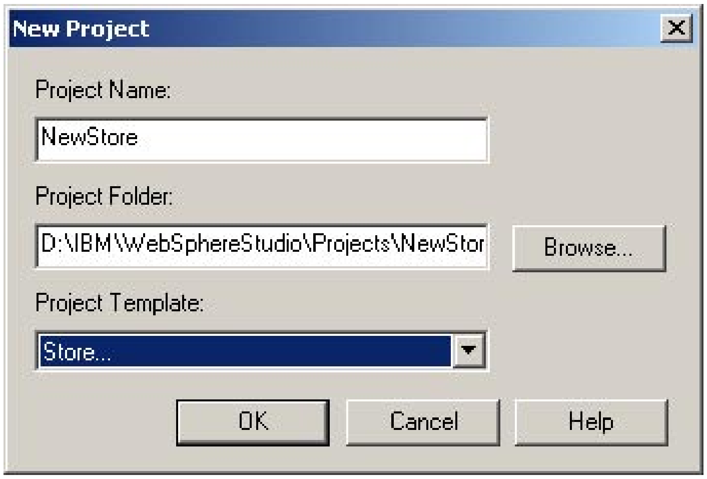
```
http://www.ibm.com/software/webservers/commerce/wcs_pro/downloadsn
c-os390.html
```

Download this model and follow the intructions to install it on your Windows PC. You will use this store model to create WCSuite stores that will run on the 390 only.

To create a store that will run on another platform as well as 390 (eg. Windows NT or AIX), do not use the 390 store model. In the case of a multiplatform store just use the "store" model as shown in the next section and then make the necessary changes (in Chapter 5.2.1, "Modify selected Net.Data macros" on page 137) on the 390 after you have published the store.

5.1.2 Use Store Creator in WebSphere Commerce Studio

Start the Commerce Studio by selecting **Start** -> **Programs** -> **IBM WebSphere Commerce Studio** -> **Studio 3.0** -> **IBM WebSphere Studio v3.0.2**. Select "new" and you see the screen shown in Figure 64 on page 128.



Project Name:

NewStore

Project Folder:

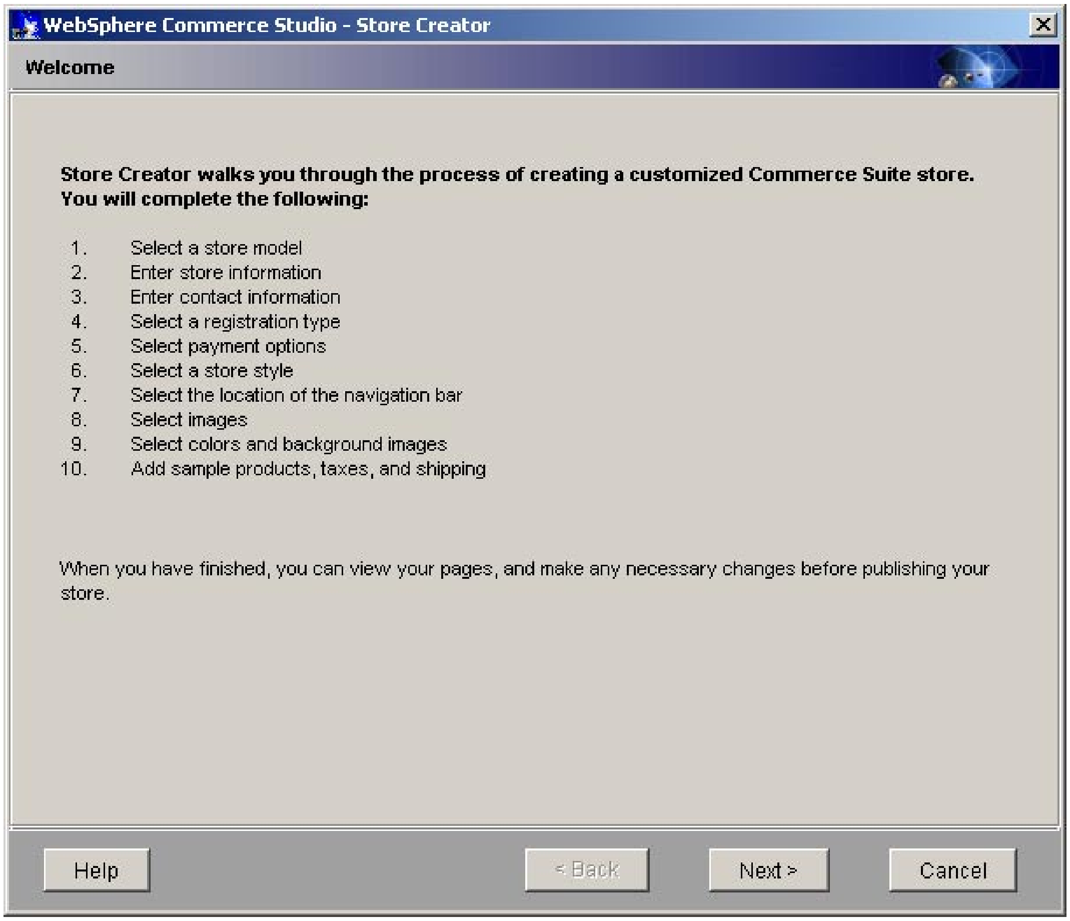
D:\IBM\WebSphereStudio\Projects\NewStor

Project Template:

Store...

Figure 64. Initial Store Creator screen

After you give your store a name (we used “NewStore” in our example) it is important to change the Project Template to “Store” before continuing, otherwise the Store Creator Wizard is not activated. The next screen you should see is shown in Figure 65.



Store Creator walks you through the process of creating a customized Commerce Suite store. You will complete the following:

1. Select a store model
2. Enter store information
3. Enter contact information
4. Select a registration type
5. Select payment options
6. Select a store style
7. Select the location of the navigation bar
8. Select images
9. Select colors and background images
10. Add sample products, taxes, and shipping

When you have finished, you can view your pages, and make any necessary changes before publishing your store.

Figure 65. Store Creator welcome screen

The Store Creator Wizard welcome screen explains the steps that you will go through to configure your store.

In the first step you select the store model, which is “Retail” by default. Commerce Studio is designed for the user to define their own store models. The task of defining your own Store Model is beyond the scope of this book.

Follow steps 1 through 10 to define the look and feel of your store. At step 10 you should add the sample products to your store to enable you to have some sample data to look at.

Note that the store sample XML document type definitions are held in three files on the OS/390 server, in the directory /usr/lpp/CommerceSuite/Tools/xml/dtd/tools/devtools/samples. The files are called sample_catalog.dtd, sample_shipping.dtd and sample_tax.dtd.

5.1.3 Store assets

After you have used the Store Creator Wizard you will notice a new directory structure in the left assets pane, containing your store assets (HTML, Net.Data macros, and images).

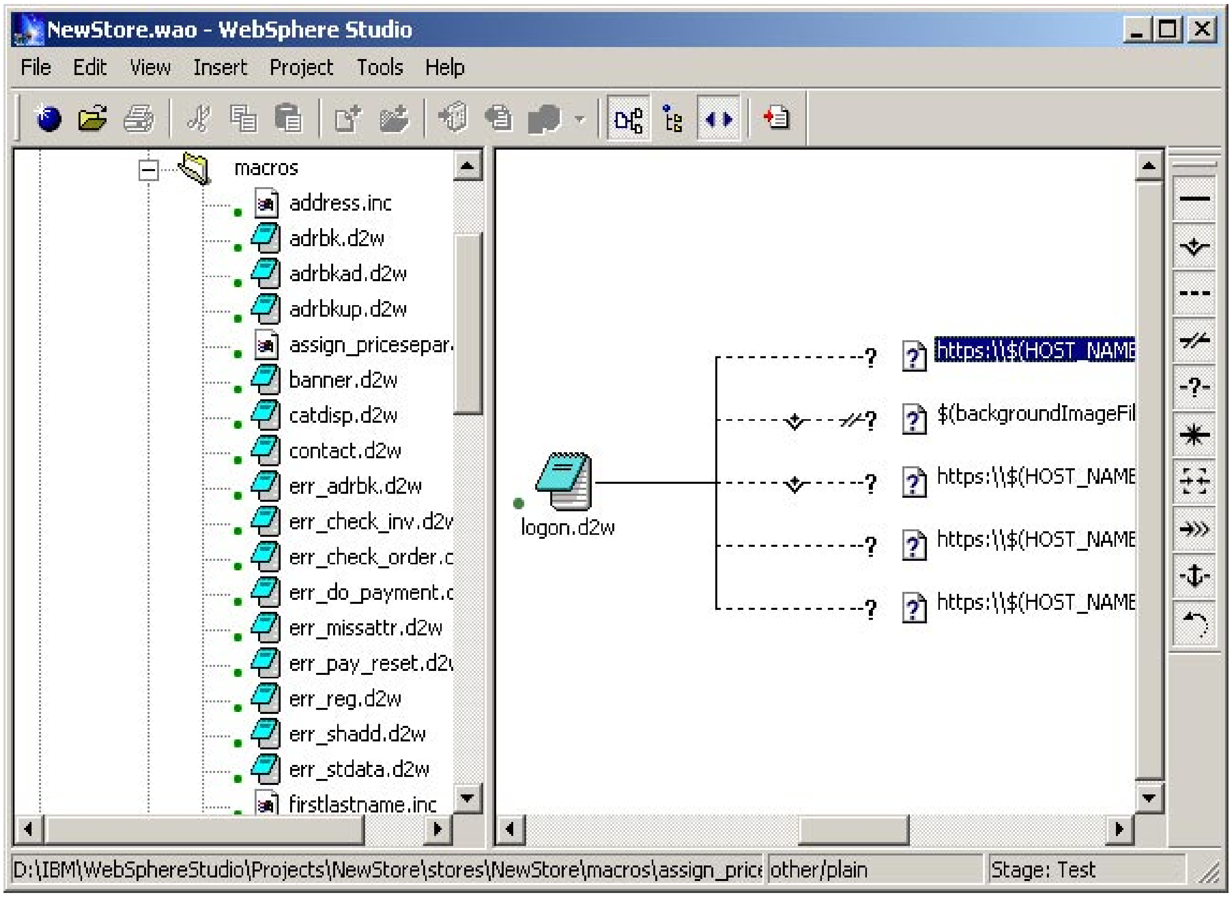


Figure 66. Assets and relatiionships view

When clicking on a file, if the relationships view is selected the file appears in the right-hand pane showing which assets it is related to.

Commerce Studio allows you to register your tool of choice as the editor for a given file extension. To determine the current editor for Net.Data macros (that is, files with the d2w extension) just double-click on the macro. You can use the Tools Registration dialog to change the default associations.

When you open an asset, a red check mark appears next to it. This indicates that the file is checked out for editting and is therefore unavailable to other store designers who may be updating the store assets in a team environment.

5.1.4 Commerce Studio file structure

Commerce Suite stores created using the Store Creator and rule projects created using Blaze Advisor Builder follow a specific file structure in Commerce Studio. All files used for store creation and rule creation are contained in the project for that store. The project is broken into the following folders:

- **rules** contains the projects folder, which contains all of the files associated with your Blaze Advisor rule project. Commerce Studio creates this folder when you use the Create Blaze Advisor Rule Project Wizard. There is only one rules folder per site.
- **stores** contains all files created by Store Creator. The stores folder includes the <storedir> folder (where <storedir> is defined using Store Creator.) The <storedir> folder contains all HTML and JavaServer Pages (JSP) files, the store profile (store.scpf), and the following subfolders:
  - **images** contains image files used in the store

    Note: If you are creating your store using Store Profile Editor, you can add your images to this folder, or create new folders of your choice, under the stores folder.
  - **macros** contains all Net.Data macros used in the store, and the store include files Storeinfo.inc and Storedir.inc. The Storeinfo.inc file controls the look and feel of the Net.Data files in your store.

    Note: If you are creating your store using Store Profile Editor, you can add your macros to this folder, or create new folders of your choice, under the stores folder.
  - **servlet** Commerce Studio automatically creates a servlet folder for all projects. If you create servlets for use in your store, store them in this folder. In all other cases, the servlet folder will be empty.
  - **theme** Contains the Master.css (cascading style sheet) file. Commerce Studio automatically creates a theme folder and Master.css file for all projects. Page Designer has a style sheet editor to edit this file.

If you wish to create your own store assets and publish your store through Commerce, you must save your files in the stores folder in order to publish properly.

5.2 Publishing the store

In the publishing stage, Commerce Studio sends all of the assets in the store (HTML pages, Net.Data macros, Java objects and images) to your OS/390 system. Also, Commerce Studio sends WCSuite database updates to enable your stores control information to be inserted into the various DB2 tables in the

WCSuite database. These database updates are applied by a Java servlet running in the WAS WebServer on OS/390.

After completing the previous 10 steps to create the store, you are presented with the “where to go next” screen. At this point you need to define your target servers of where you wish to publish to. Commerce Studio had two main window panes, the left-hand pane is the tree view of your stores assets, the right hand pane is either the relations or publishing view:

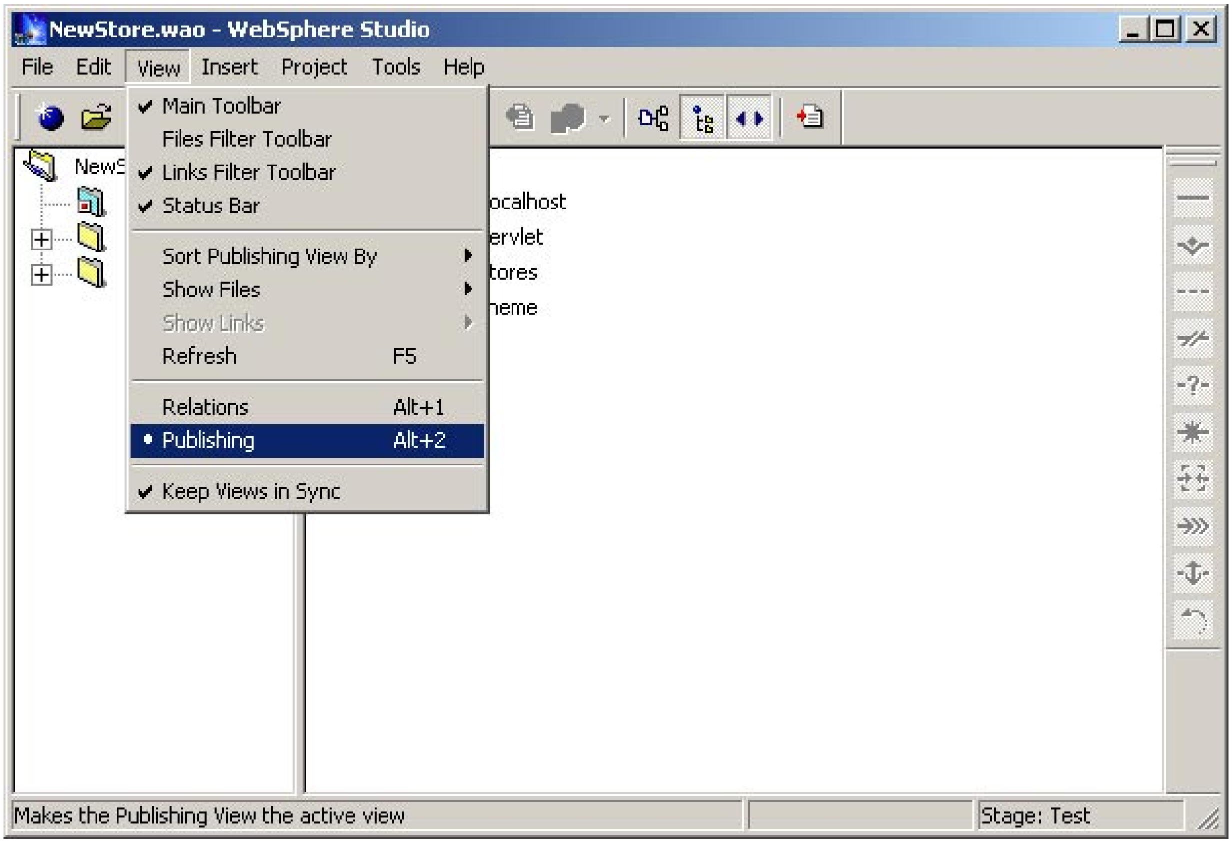


Figure 67. Relations and Publishing in Commerce Studio

Make sure that you select the Publishing view so that you can define the servers which you wish to publish to.

Commerce Studio has the concept of two different publishing stages, one for “test” and one for “production” server. To switch between these stages right-click anywhere within the right-hand pane.

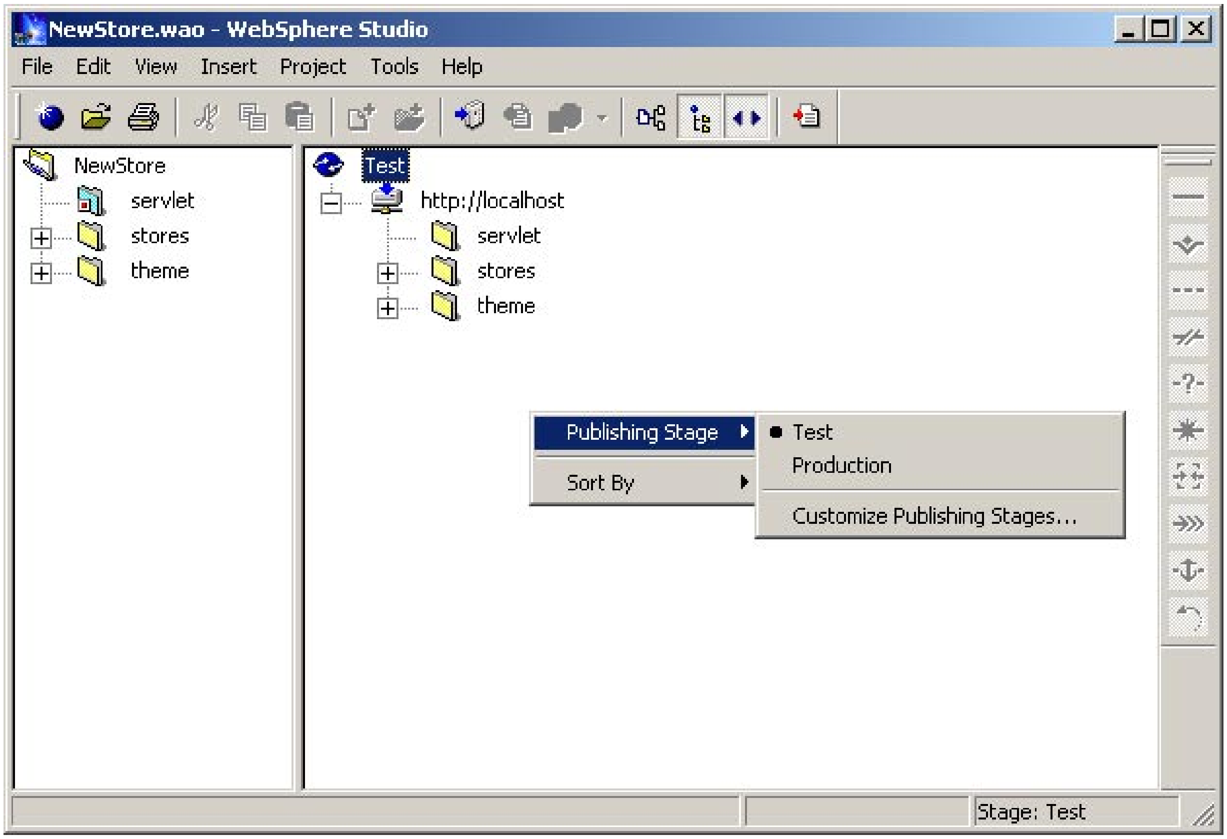


Figure 68. Test and Production publishing stages

When in the desired stage (test or publishing), right-click on the word “test” or the word “publishing” to insert your target OS/390 server. Use the following procedure to add as many test and production servers as are necessary.

5.2.0.1 Adding publishing targets

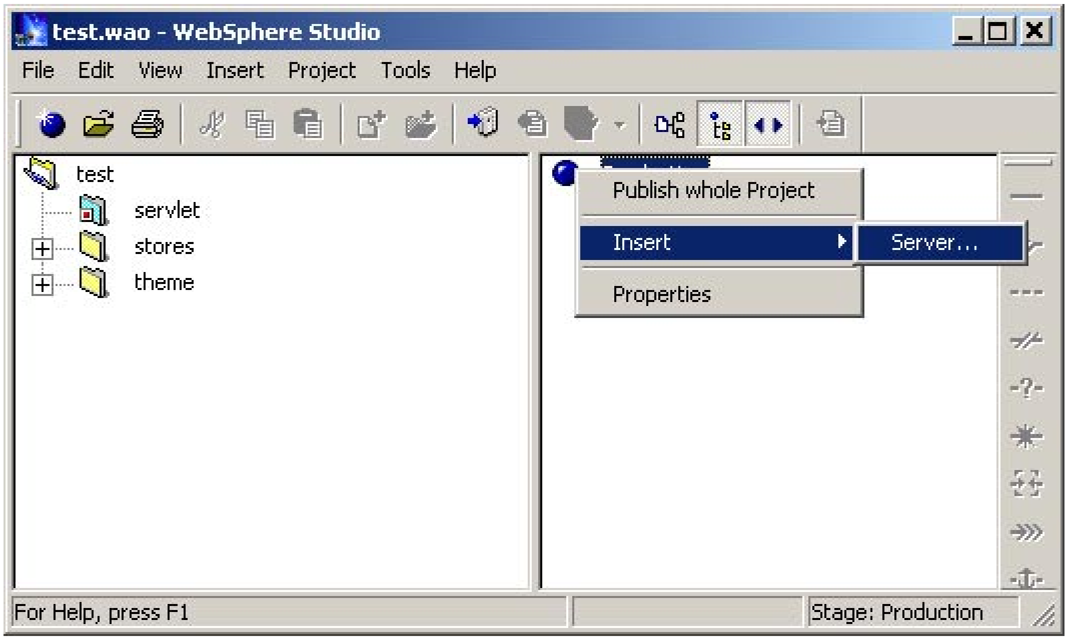
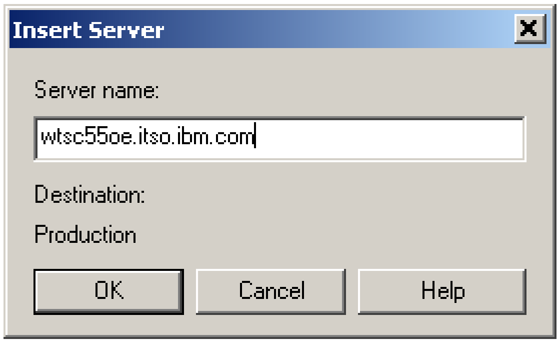


Figure 69. Inserting a server to publish to

This action brings up the dialog shown in Figure 70, where you insert the name of your OS/390 server.

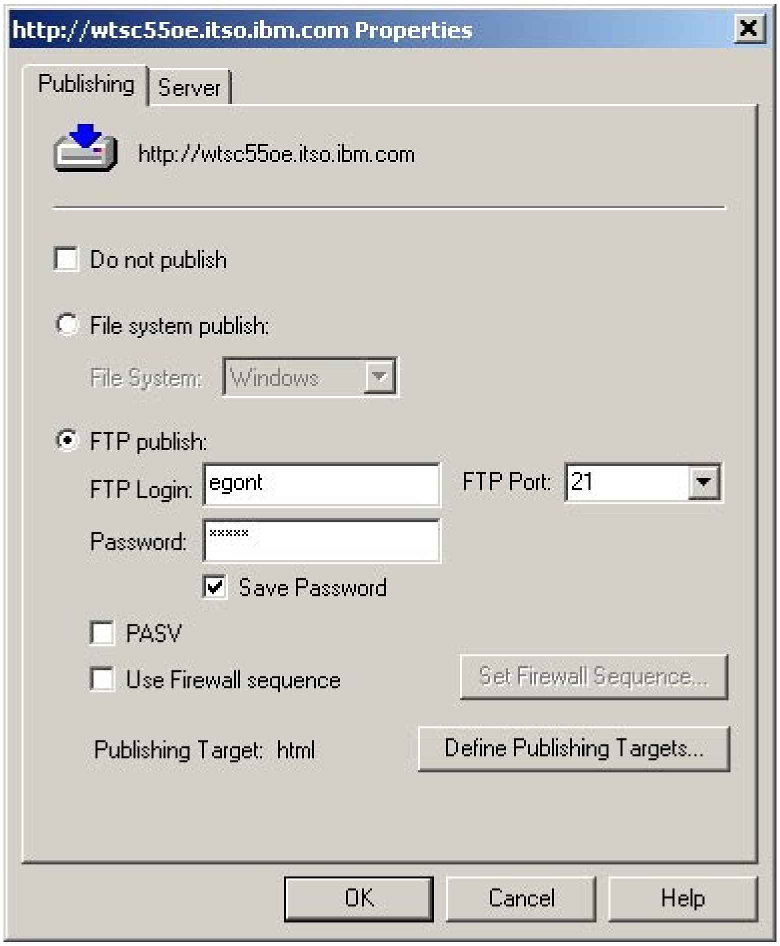


Server name:
wtsc55oe.itso.ibm.com

Destination:
Production

Figure 70. Type your server name

Once you have inserted your server, right-click on it and change the properties to use ftp as the transport mechanism when you send your files to your OS/390 server. We assume that you can also use the “Windows” method of file transfer if you have used an SMB product (such as DFSSMB or SAMBA) to mount your OS/390 HFS as a local drive to your Windows machine. Note that we did not test this, however.



http://wtsc55oe.itso.ibm.com Properties

http://wtsc55oe.itso.ibm.com

☐ Do not publish

○ File system publish:

File System: Windows

● FTP publish:

FTP Login: egont

FTP Port: 21

Password: *****

☑ Save Password

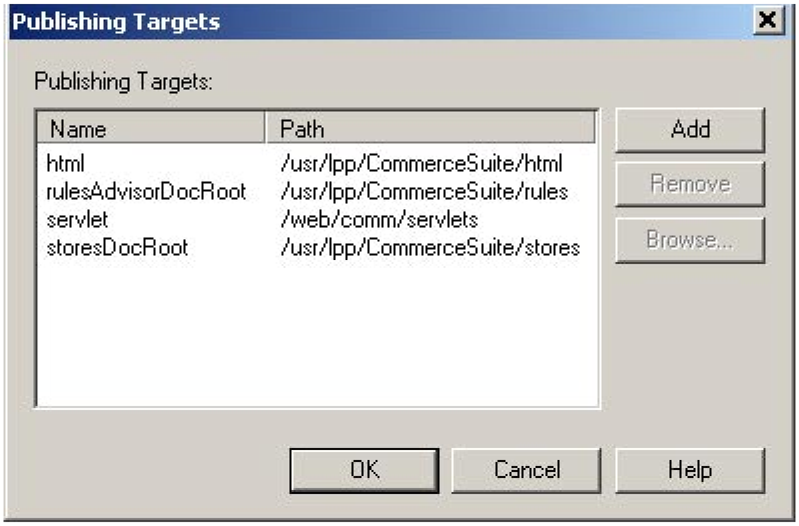
☐ PASV

☐ Use Firewall sequence

Publishing Target: html

Figure 71. Define server file transfer properties

After filling in the details for the ftp connection to your OS/390 system, click **Define Publishing Targets...** then as a minimum click the **storeDocRoot** path and change it to /usr/lpp/CommerceSuite/stores. We defined our publishing targets as shown in Figure 72.



Publishing Targets

Publishing Targets:

Name	Path
html	/usr/lpp/CommerceSuite/html
rulesAdvisorDocRoot	/usr/lpp/CommerceSuite/rules
servlet	/web/comm/servlets
storesDocRoot	/usr/lpp/CommerceSuite/stores

Figure 72. Defining Publishing Targets

Ensure that the file paths exist on your OS/390 system and that they have UNIX permission bits set to 755. Click OK, then click OK again on the hostname properties window.

Now expand the tree in the right-hand pane so that the “stores” and theme folders are visible.

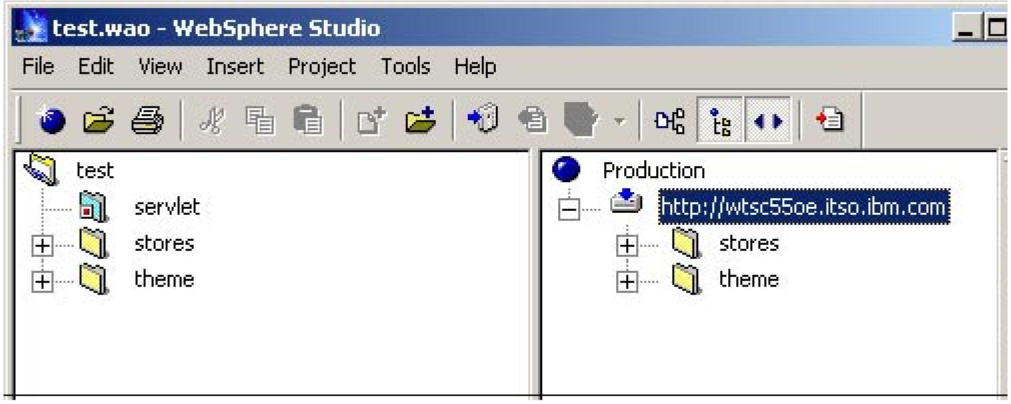
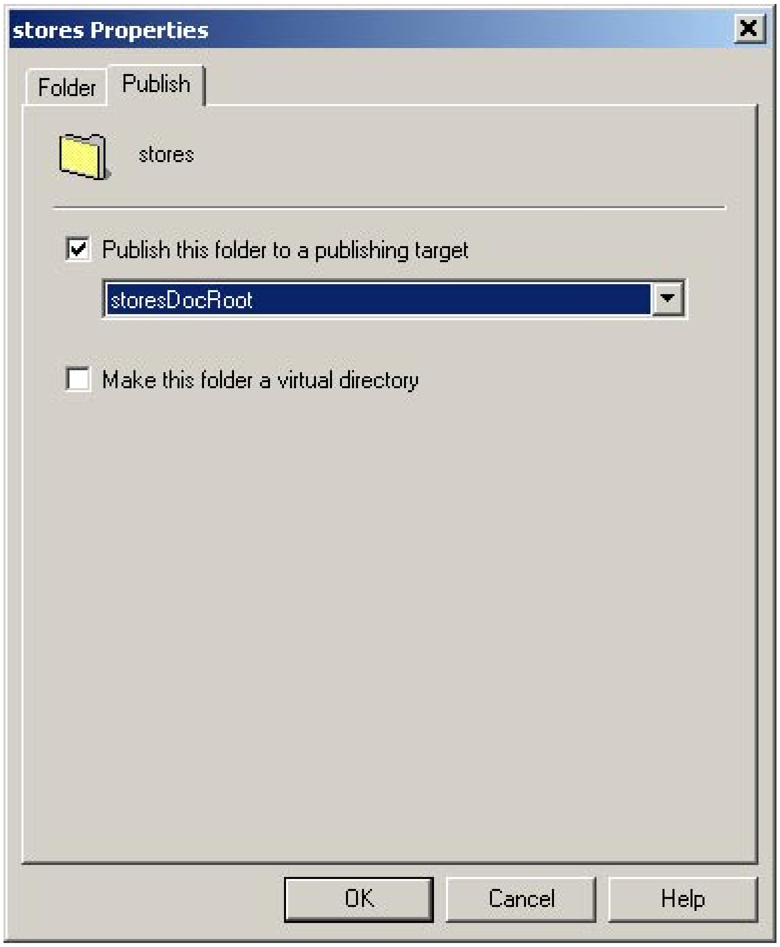

test.wao - WebSphere Studio
File Edit View Insert Project Tools Help
test
servlet
stores
theme
Production
http://wtsc55oe.itso.ibm.com
stores
theme

Figure 73. The stores and themes folders

Right-click on stores and change the properties as illustrated in Figure 74.



stores

☑ Publish this folder to a publishing target

storesDocRoot

□ Make this folder a virtual directory

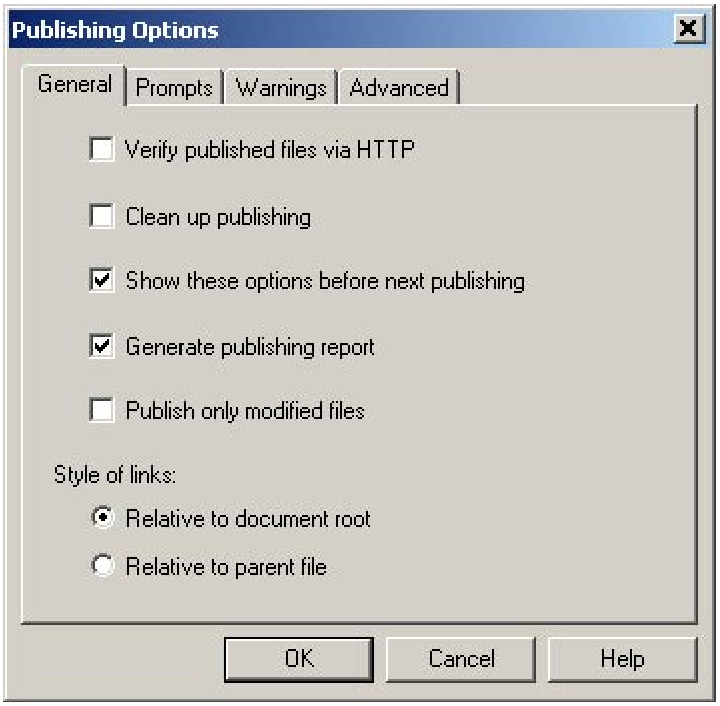
Figure 74. Stores (and theme) properties

Make sure that the virtual directory option is not selected. Do the same to the theme folder (that is, publish to storesDocRoot and ensure that virtual directory is not selected.)

You are now ready to publish your store. The GWAPI Webserver and the WAS Webserver server must both be running on OS/390 to accept the updates from Commerce Studio.

It is important that the WAS MerchantAdminServlet started when the WAS Webserver started; check the WAS ncf log to ensure that there were no error messages at startup time of the WAS Webserver.

To publish your store, right-click on the server in the right pane and select “Publish this server.” If necessary, uncheck the “Publish only modified files” option if you wish to re-publish all assets of your store.



☐ Verify published files via HTTP

☐ Clean up publishing

☑ Show these options before next publishing

☑ Generate publishing report

☐ Publish only modified files

Style of links:

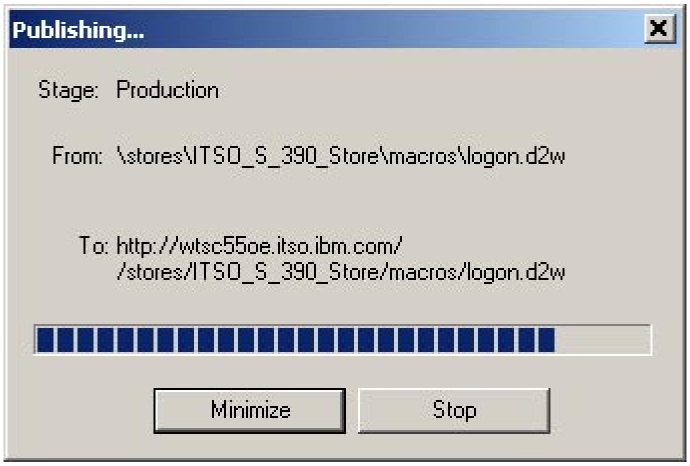
● Relative to document root

○ Relative to parent file

Figure 75. Publishing options

During publishing you will be asked to enter the password to the NCADMIN userid. This authority is required to make the necessary updates to the WCSuite database. (Note that on our systems the NCADMIN password had to be typed in lower case characters).

You will see various file transfer messages as the store is published.



Publishing...

Stage: Production

From: \stores\ITSO_S_390_Store\macros\logon.d2w

To: http://wtsc55oe.itso.ibm.com/
/stores/ITSO_S_390_Store/macros/logon.d2w

Figure 76. A 390 store being published

During the final phase of the publishing stage you will be shown a summary report of what took place during publishing and any notification of errors that occurred.

After publishing, log on to your OS/390 server before launching the store and type the following comands:

```
cd /usr/lpp/CommerceSuite/stores
chmod -R 755 <storename>
```

5.2.1 Modify selected Net.Data macros

Do not make the changes to the net.data macros if you used the “390 store model” discussed in 5.1.1, “390 store model” on page 127.

The following three store files generated by Commerce Studio (when using the “store” model rather than the “390 store” model) are not compatible with Commerce Suite on OS/390:

- orderlstc.d2w
- platform.inc
- shptolst.d2w

Therefore, you must copy the default versions of these files supplied with Commerce Suite for OS/390 and modify them manually.

For each store you have published:

1. Copy the three files listed above from /usr/lpp/CommerceSuite/stores/macros to the directory for each store you have published. For example:

   cp /usr/lpp/CommerceSuite/stores/macros/orderlstc.d2w <your_store_directory>/macros/orderlstc.d2w

   cp /usr/lpp/CommerceSuite/stores/macros/platform.inc <your_store_directory>/macros/platform.inc

   cp /usr/lpp/CommerceSuite/stores/macros/shptolst.d2w <your_store_directory>/macros/shptolst.d2w

2. Edit the two Net.Data macro files (orderlstc.d2w and shptolst.d2w), and change the name of the store from <storename> to the name of your store on

the first include statement located at the top of each file to reflect the name of your store.

For example change "<storename>/macros/Storedir.inc" to "ITSO_Store/macros/Storedir.inc"

3. Save and close the files.

Note: You only need to copy the platform.inc file. You do not need to modify it.

5.2.2 Verify the store exists on the OS/390 system

At the end of the file transfer and publishing stage you will be presented with a screen offering to launch your store. Select this option and your store homepage will be served to your browser. You should bookmark your store homepage so that you can easily return to it later. Alternatively, you can access the homepage at the following URL:

```
http://<hostname>/stores/<store name>
```

where <hostname> is the name of the Webserver running the Commerce Suite instance and <store name> is the name of the store (from when you ran the Store Creator Wizard).

5.3 Replace an existing Net.Data macro

As an exercise, we suggest you replace the logon.d2w macro. As a start, make a copy of the existing logon.d2w to a file called newlog.d2w.

Modify newlog.d2w to use new display text. You might try replacing any of the following Net.Data variables with your own text:

$(TXT_PAGEDESCRIPTION_LOGON) = "Type your logon ID and password to retrieve your registration information."

$(LBL_LOGONID) = "Logon ID"

$(LBL_PASSWORD) = "Password"

The change could be as simple as using upper case instead of mixed case text or shortening the descriptive text. If you are familiar with HTML, you may want to change the page layout. Keep in mind that your changes should be obvious enough that you can recognize that you are using the new macro once published.

Save your changes and check in the new file once it is complete.

5.3.0.1 Using the Store Profile editor

The Store Profile editor is available for changing basic store information, including the assignment of WCSuite tasks to Net.Data macros. The Store Profile editor uses a file called store.scpf in your store directory in Commerce Studio. This is an XML file that contains tags holding the configuration of your store.

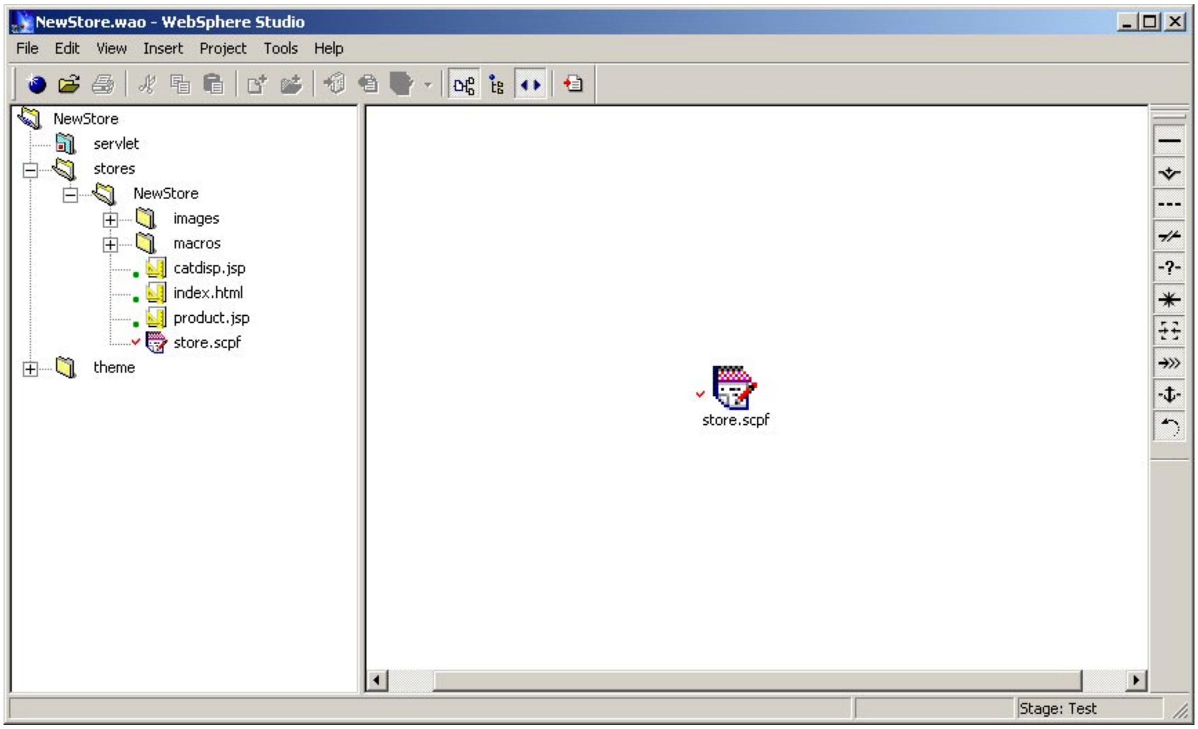
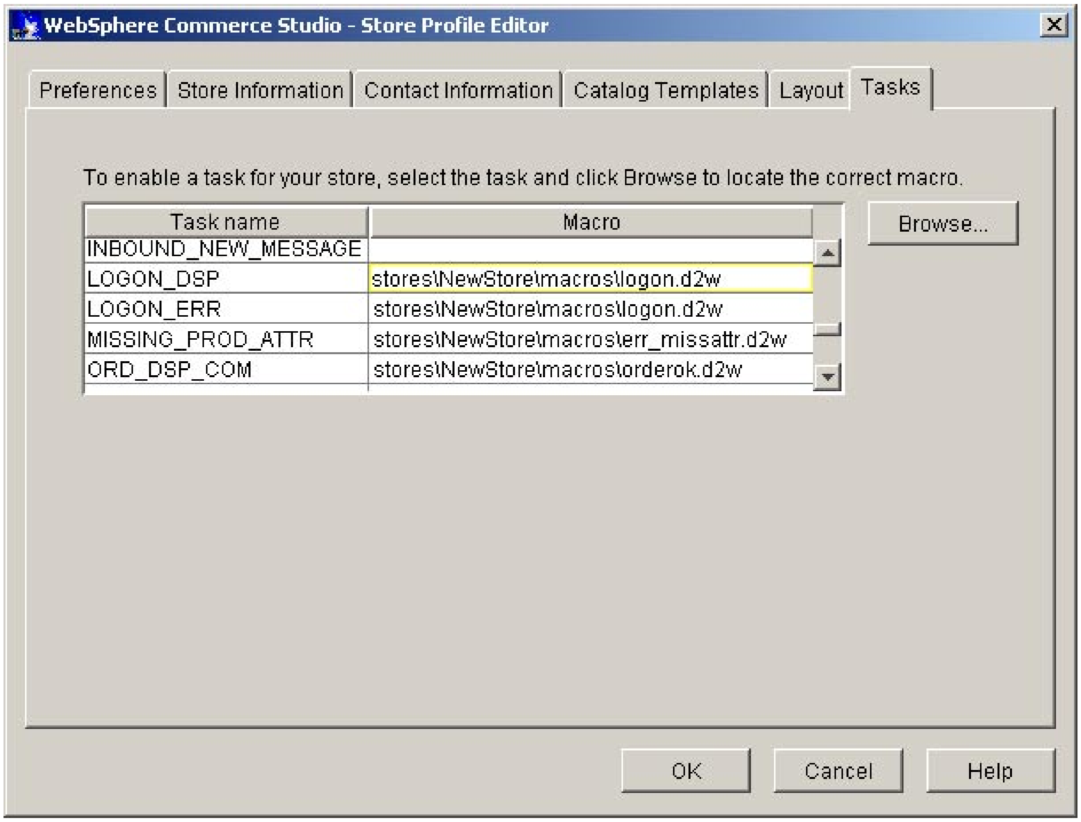

NewStore.wao - WebSphere Studio

Figure 77. Store Profile XML file

You can launch the Store Profile editor in a number of ways: you can select **Tools**->**Wizards**->**Store Profile editor**, or double-click on the store.scpf file. Use the Store Profile editor to assign the new macro to the Commerce Suite logon display task (LOGON_DSP).



To enable a task for your store, select the task and click Browse to locate the correct macro.

Task name	Macro
INBOUND_NEW_MESSAGE	
LOGON_DSP	stores\NewStore\macros\logon.d2w
LOGON_ERR	stores\NewStore\macros\logon.d2w
MISSING_PROD_ATTR	stores\NewStore\macros\err_missattr.d2w
ORD_DSP_COM	stores\NewStore\macros\orderok.d2w

Figure 78. Assigning a new macro to the LOGON_DSP task

After you have assigned the new macro, re-publish the store.

Test the store and ensure your new logon page appears.

5.4 Creating Java Server Pages

Java Server Pages (JSPs) are new with WCSuite 4.1. JSP is a dynamic scripting capability for Web pages that allows Java and a few special tags to be embedded into a Web file (HTML/XML, etc). The suffix of a JSP traditionally ends with .jsp to indicate to the Webserver that the file is a JSP file. JSPs are a server-side technology; they are compiled into Servlets and run on the server.

Product and Category display pages can be written using JSP (as well as the traditional Net.Data macros which can still be used also). Although you have either of these technologies available, you can only use one kind of template in your store. That is, if you create a category page template using JSP files, you must create product list and product page templates using JSP files.

In this example we will lead you through creating a JSP, testing it, and integrating the JSP into your sample store.

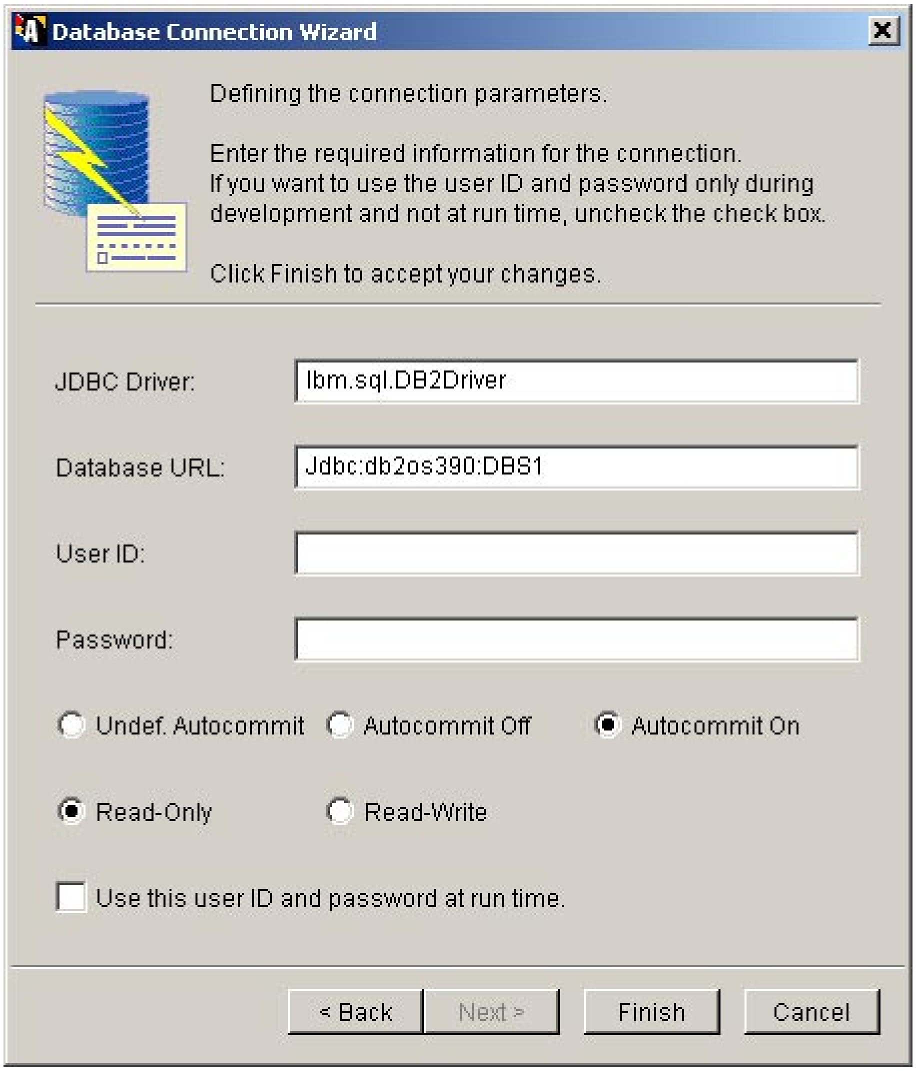
5.4.1 Import your OS/390 database object model

Before you begin creating any Java objects that access DB2, you need to update the database model to match your OS/390 system definitions. Select **Project** from the menu bar and select **Import Object Model - Database** from the drop-down list. The Database Class Provider Editor window will appear.

Click the **Connections** tab, click **Edit**, then click **Next** . Select "Ibm.sql.DB2Driver" in the JDBC Driver field and "Jdbc:db2os390:<your DB2 location name>" in the "Database URL" field. Leave the userid and password fields blank. Select the following radio buttons:

Autocommit On

Read-Only



Defining the connection parameters.

Enter the required information for the connection.
If you want to use the user ID and password only during development and not at run time, uncheck the check box.

Click Finish to accept your changes.

JDBC Driver: Ibm.sql.DB2Driver

Database URL: Jdbc:db2os390:DBS1

User ID:

Password:

○ Undef. Autocommit ○ Autocommit Off ● Autocommit On

● Read-Only ○ Read-Write

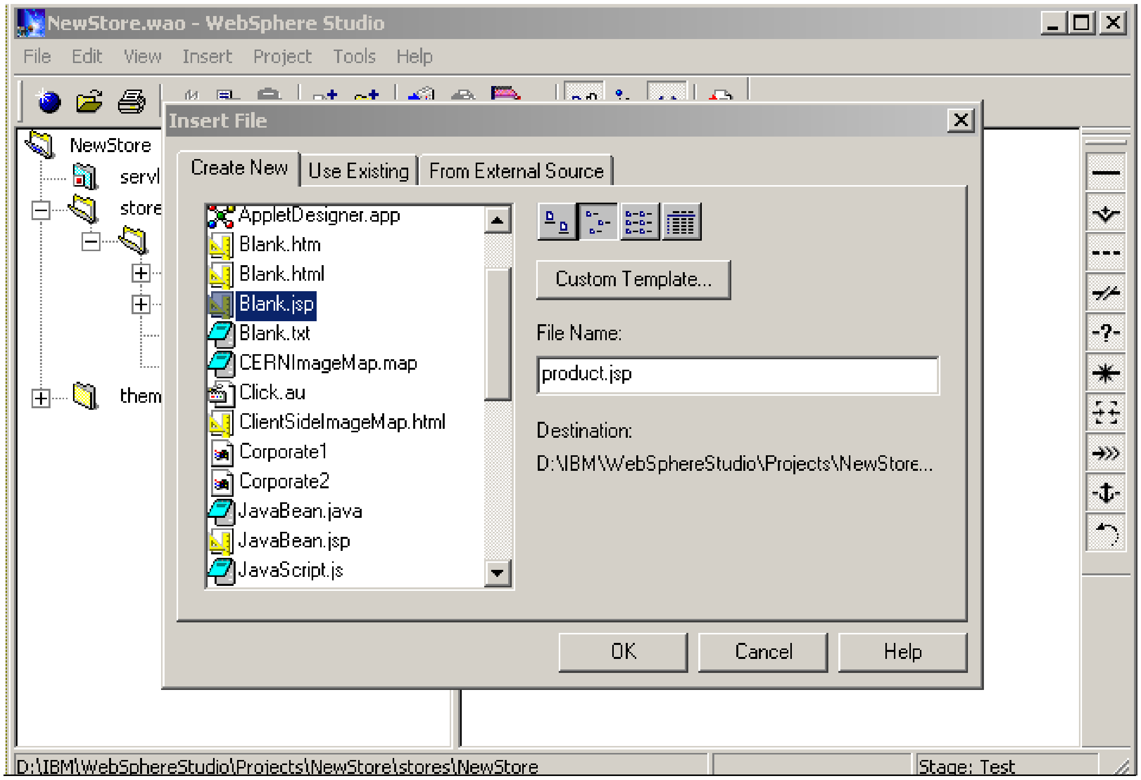
□ Use this user ID and password at run time.

Figure 79. Changing your database connection parameters

Click **Finish**. If you see a message saying a connection could not be established then take the option to save anyway. Click **OK** twice.

5.4.2 Creating a product JSP

You create the JSP in Commerce Studio. Start Commerce Studio, right-click the <storedir> folder, where <storedir> is the name of your store. From the **Insert** menu, select **File**. The Insert File dialog box displays. In the **Create New** tab, select Blank.jsp.



File Name: product.jsp

Destination: D:\IBM\WebSphereStudio\Projects\NewStore...

D:\IBM\WebSphereStudio\Projects\NewStore\stores\NewStore

Stage: Test

Figure 80. Creating a blank jsp with Commerce Studio

In the File Name field, rename the file. For example, rename the file to "product.jsp". Click **OK**. The file product.jsp (in a "checked-out" status, notice the red check mark) displays in the <storedir> folder and in the relationships view (the large pane on the right).

Note: To rename a file in Commerce Studio, select the file, then from the **Edit** menu, select **Rename**; the fastpath is the F2 key. Alternatively, select a file (it turns blue), then select it again and you will be able to rename it.

5.4.3 Add a ProductBean

The editor that Commerce Studio uses to edit JSP and HTML files is called “Page Designer.” Page Designer is very similar to NetObjects TopPage. We present these steps in some detail since you have to get this exactly right or it will not work.

To add a ProductBean to a JSP file, complete the following steps:

1. Open the appropriate JSP file, for example Product.jsp.
2. In Commerce Studio, select the <storedir> folder.
3. Double-click the appropriate JSP file. The file opens in Page Designer.
4. Of the three view tabs (Normal, HTML Source, and Preview) at the bottom of the right-hand pane, ensure that the **Normal** view is selected.
5. Add the ProductBean to the file. Adding the ProductBean allows you to access the product information in the Commerce Suite database.

6. In Page Designer, in the View drop-down menu from the action bar, ensure that Contents Manager is selected.
7. In the Contents Pane (the top left pane of the Page Designer window), navigate to <drive:>\IBM\CommerceStudio\databeans\lib (you need to know where Commerce Studio is installed on your Windows machine).
8. From the drop-down list of the lower left-hand pane (which says “Image Files ...”), select Java Bean Files (.class; .jar). The available jar files display in the pane below.

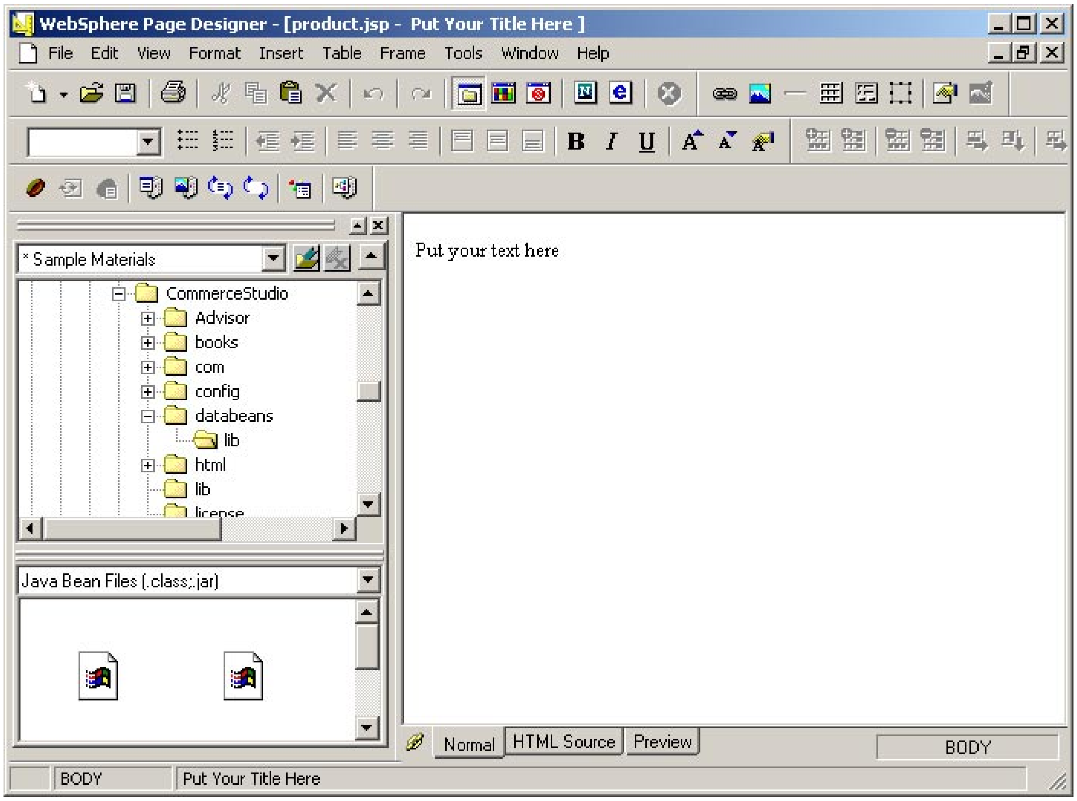

WebSphere Page Designer - [product.jsp - Put Your Title Here]
File Edit View Format Insert Table Frame Tools Window Help
* Sample Materials
CommerceStudio
Advisor
books
com
config
databeans
lib
html
lib
license
Java Bean Files (.class;.jar)
Put your text here
Normal
HTML Source
Preview
BODY
BODY
Put Your Title Here

Figure 81. Page Designer initial screen

9. Select wcsclient.jar. Drag and drop the file into the Normal view. The Beans Selection window is displayed.
10. Select com.ibm.commerce.beans.ProductBean. Click **OK**. The Attribute window is displayed.
11. In the Scope drop-down list, select **request** (this gives the bean a lifetime of the current request only).
12. In the Attribute window, select the **setProperty** tab.
13. In the Property field, type * then click **Add** (adding the * sets all properties in the bean that match the parameters in the page URL to the values specified in the URL. With * set, the server iterates through available properties and request parameters, matching up ones with identical names. In this instance, the values for the merchant reference number (merchant_rn) and the product reference number (prrfnbr) are set in the bean.
14. Click **OK**. The J icon is displayed in the Page Designer window (which indicates that there is some Java here on the page).

15. In Page Designer, select the **HTML Source** tab to view the bean tag generated by Page Designer. The bean tag should display as follows:

```
<jsp:useBean class="com.ibm.commerce.beans.ProductBean" id="productBean" scope="request">
<jsp:setProperty property="*" name="productBean" />
<% com.ibm.commerce.beans.DataBeanManager.activate(productBean, request); %></jsp:useBean>
```

5.4.4 Add dynamic product information

If you are creating a product page template, you will want to add product information to your page from the Commerce Suite database. Instead of adding static information, you can add information dynamically using the ProductBean, which will be updated when your JSP is requested by a browser.

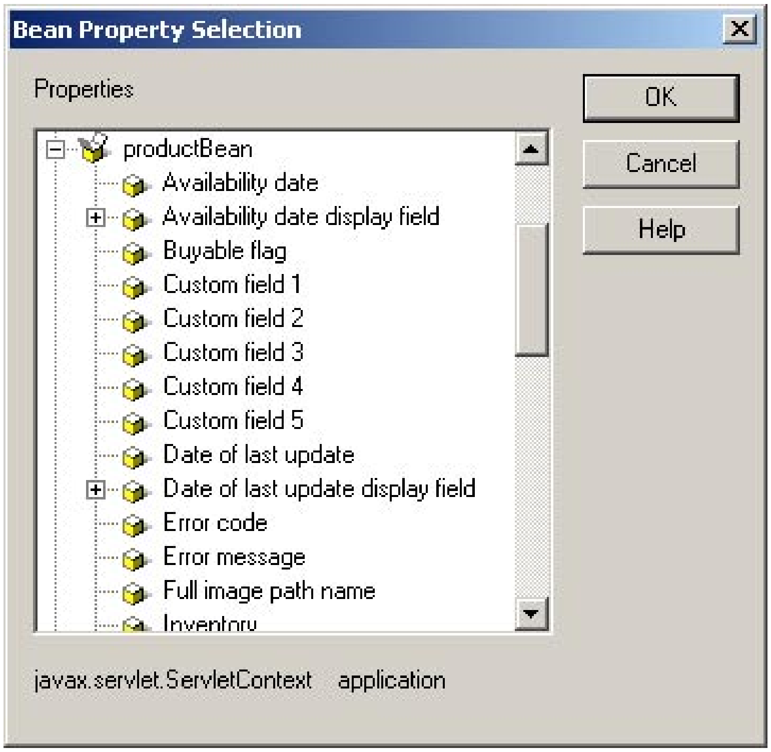
To add dynamic product information, complete the following steps:

1. In the normal view, position your cursor after the J icon.
2. From the **Insert** menu, select **Form and Input Fields** -> **Form**. The Form and Input Fields toolbar is displayed and a form (a pink box) is added to your page. This form will allow your shoppers to add products to an interest list (sometimes known as a shopping cart). You will complete the creation of this form in the “Complete the form for the InterestItemAdd command” task, later in the “Create a product page” process.
3. Position your cursor inside the form (if it is not there already).
4. Add dynamic product information to your product page. Include the following elements by following the instructions in the sections below.
   - Product short description
   - Product price and currency
   - Product long description
   - Thumbnail image

If you wish, you can further customize your product page using the features available in Page Designer.

5.4.4.1 Add product short description

1. From the **Insert** menu, select **Dynamic Elements** -> **Property Display**. The Attribute dialog box is displayed.
2. Click **Browse**. The Bean Property Selection dialog box is displayed.
3. Select **+** to display the properties for the productBean.



- productBean
  - Availability date
  - Availability date display field
  - Buyable flag
  - Custom field 1
  - Custom field 2
  - Custom field 3
  - Custom field 4
  - Custom field 5
  - Date of last update
  - Date of last update display field
  - Error code
  - Error message
  - Full image path name
  - Inventory

javax.servlet.ServletContext application

Figure 82. Product Bean

4. Select **Short description**, then click **OK**. The productBean.shortDescription is displayed in the Properties field in the Attribute dialog box.
5. In the Sample Text field, type `PRODUCT_NAME`, or sample text of your choice. Click **OK**. Press Enter.

The sample text is displayed in the product page template. When the product page is requested, the sample text will be replaced with information from the Commerce Suite database.

5.4.4.2 Add product price and currency

1. From the **Insert** menu, select **Dynamic Elements** -> **Property Display**. The Attribute dialog box is displayed.
2. Click **Browse**. The Bean Property Selection dialog box is displayed.
3. Select **+** to display the properties for the productBean.
4. Expand **Price**, expand **Currency display field**, then select **Currency display**. Click **OK**. The productBean.price.displayCurrency.presentationString is displayed in the Properties field in the Attribute dialog box.
5. In the Sample Text field, type `PRICE` or sample text of your choice. Click **OK**. Press Enter.

When the product page is requested, the sample text will be replaced with information from the Commerce Suite database.

5.4.4.3 Add product long description

Repeat the steps listed in product short description previously, selecting **Long description field 1** instead of short description. In the Sample Text field, type `PRODUCT_DESC1` or sample text of your choice. Press Enter.

Repeat the process described, selecting **Long description field 2** and **field 3**. Press Enter each time.

5.4.4.4 Add a thumbnail image

1. From the **Insert** menu, select **Dynamic Elements** -> **Dynamic Image**. The Attribute dialog box is displayed.
2. In the SRC Attribute field, click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the productBean.
3. Select **Thumbnail image path name**. Click **OK**. The productBean.thumbnailPathName is displayed in the SRC Attribute field in the Attribute dialog box. Click **OK**. Press Enter.

The sample image (with a red X through it) displays in the products table. When the product page is requested, the sample image will be replaced with an image from the Commerce Suite database.

5.4.4.5 Complete the form for the InterestItemAdd command

Now add the InterestItemAdd command (this is the WCSuite command that adds an item into your virtual shopping cart prior to checkout) to the same form.

Complete the following steps:

1. Right-click on your form. From the menu, select **Attributes**. The Attribute window is displayed.
2. In the Action field, type `/webapp/commerce/command/InterestItemAdd`. This is the URL for the InterestItemAdd command.
3. Select the method of “Post” by clicking the radio button.
4. Select the **Hidden Fields** tab.
5. In the Name field, type `merchant_rn`. This is the merchant reference number and represents the store in the mall which owns this shopping list and is a required parameter to the WCSuite InterestItemAdd command.
6. Under the Value field, select the “Specify by property” check box. Click **Browse**. The Bean Property Selection dialog box is displayed.
7. Select **+** to display the properties for the productBean.
8. Select **Merchant reference number**. Click **OK**. The productBean.merchantReferenceNumber is displayed in the Value field in the Attribute dialog box. Click **Add**.
9. In the Name field, type `product_rn`
10. Under the Value field, select the “Specify by property” check box. Click **Browse**. The Bean Property Selection dialog box is displayed.
11. Select **+** to display the properties for the productBean.
12. Select **Product reference number**. Click **OK**. The productBean.productReferenceNumber is displayed in the Value field in the Attribute dialog box. Click **Add**.

13.In the Name field, type `url`.

14.In the Value field, type:

```
/webapp/commerce/command/InterestItemDisplay?merchant_rn=<%=java.net.URLEncode
r.encode(productBean.getMerchantReferenceNumber())%>
```

15.Click **Add**, then **OK**.

5.4.4.6 Add an interest list button

To add a button to your product page that allows customers to add products to their interest list, complete the following steps:

1. If it is not already selected, select the form.
2. In the Form and Input Fields toolbar, click **SUB**. The Submit button is added to the form, and the Attribute dialog box is displayed.
3. In the Name field, type `InterestItemAdd`, or text of your choice.
4. In the Label field, type `Add to interest list`, or text of your choice.
5. Ensure that Submit is selected as the Button Type. Click **OK**. The new button is displayed in the form.
6. From the File menu, select **Save** (you've now saved product.jsp).

Your Page Designer screen should now look something like the one shown in Figure 83.

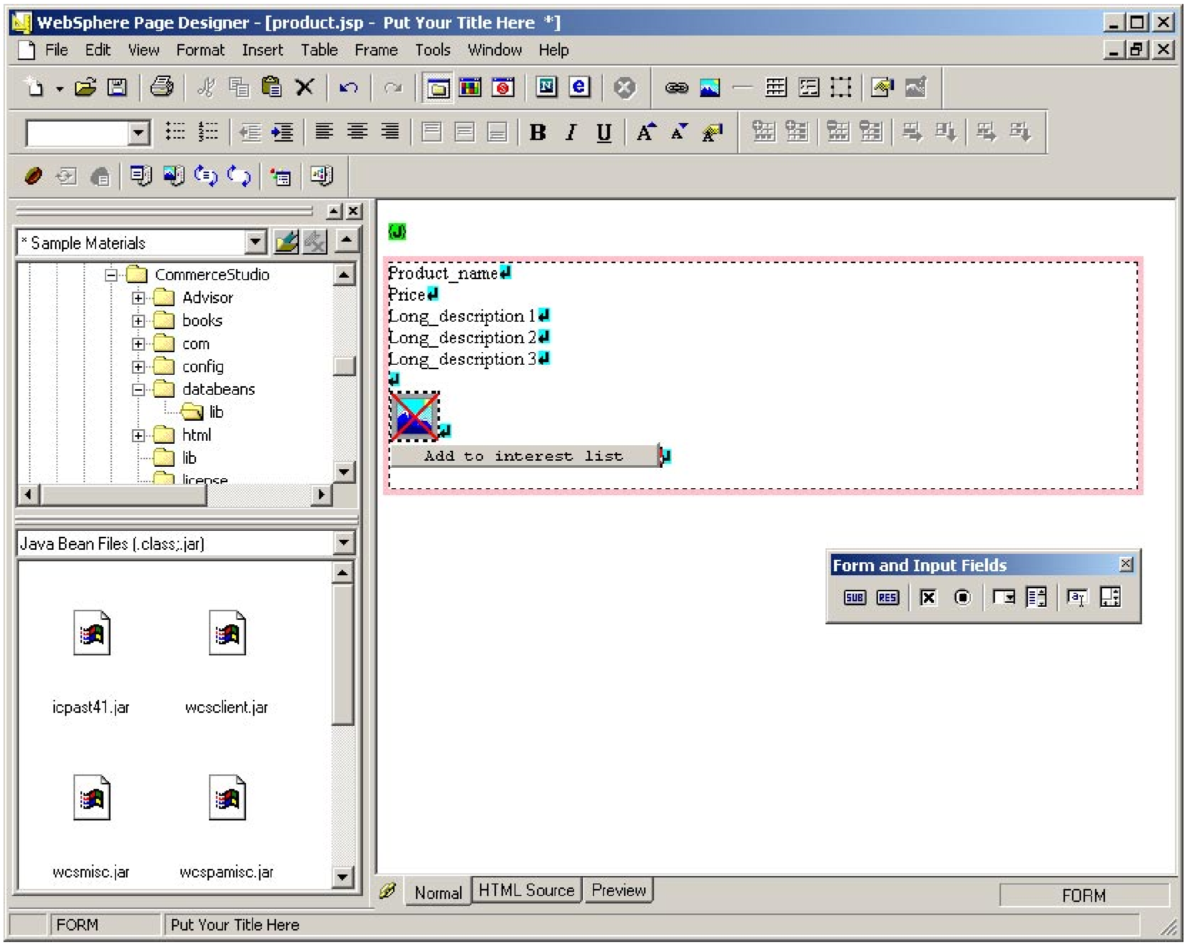


Figure 83. Page Designer Competed Form

Close the page designer, and from Commerce Studio make sure that you “check-in” product.jsp.

Publish your Commerce Studio project to your OS/390 system. Check the report produced for any failures.

5.4.4.7 Test the product JSP

Test that the product page template works with a sample product by following these steps.

Note: If you are familiar with the Commerce Suite schema, you can query the database using DB2 utilities (such as ISQL) to determine the merchant reference number and a suitable product reference number instead of following steps 1 through 3.

1. Open your store in a browser.
2. Browse through the catalog until you are viewing a list of products.
3. Right-click on a product and use the **copy link location** action from the pop-up menu. This puts the URL in the clipboard. Simply place the cursor in the “Go to:” field of the browser and then “paste” to view the complete link.

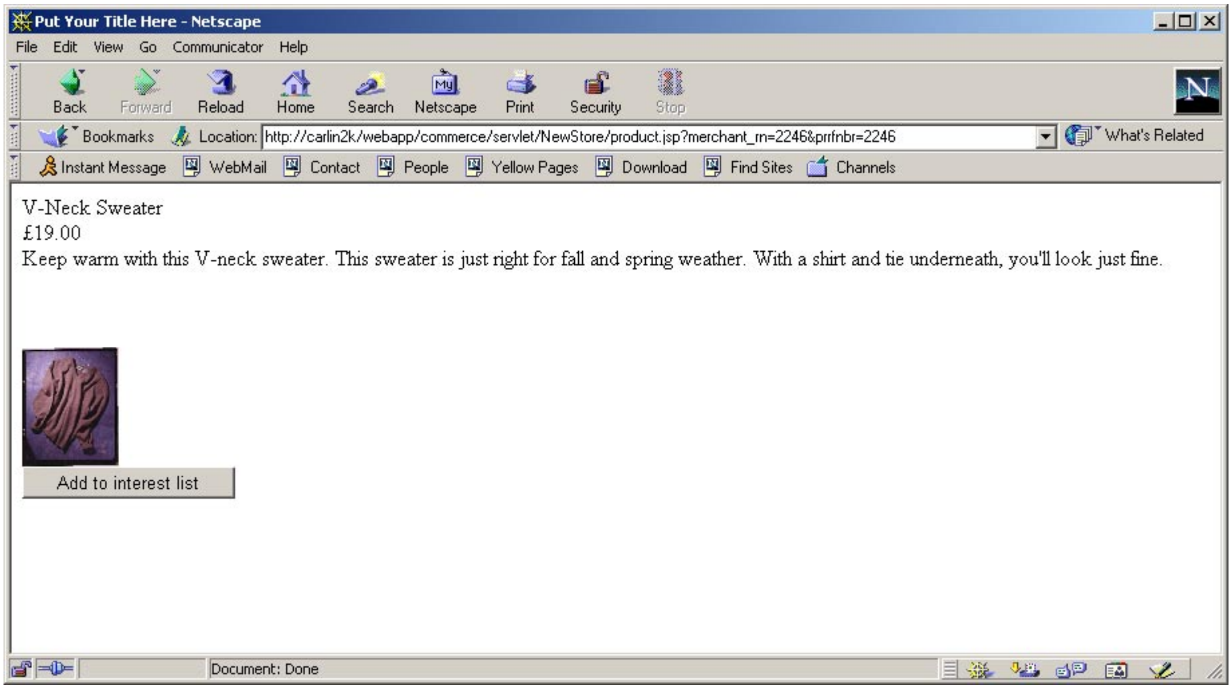
4. Make note of the merchant reference (prmenbr) and product reference number (prrfnbr). You can also get this information from the PRODUCT table in DB2.
5. Type the following URL in the browser to display the page:

   ```
   http://<your-host>/webapp/commerce/servlet/<storedir>/product.jsp?merchant_
   rn=xxxx&prrfnbr=yyyy
   ```

   where <storedir> is the store directory for your store, product.jsp is the name of the product display page JSP file and xxxx and yyyy are the appropriate reference numbers from step 4.

Make sure that WCSuite and WebSphere Application Server are active in the OS/390 system before you run your JSP.

The output from the jsp should look something like Figure 84.


Put Your Title Here - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://carlin2k/webapp/commerce/servlet/NewStore/product.jsp?merchant_rn=2246&prrfnbr=2246
What's Related
Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels

V-Neck Sweater
£19.00
Keep warm with this V-neck sweater. This sweater is just right for fall and spring weather. With a shirt and tie underneath, you'll look just fine.

Add to interest list
Document: Done

Figure 84. Output from product.jsp

There are three common errors you might see back at the browser when you try to run your jsp, instead of the success screen. The common problems and resolutions are as follows:

- java.lang.ClassFormatError: wrong name

  When you see this you should stop and re-start your WC Suite Container in the WebSphere Application Server.

- Server.javax.servlet.ServletException: xxxx

  This is due to a bad parameter in either the merchant_rn or prrfnbr.

- com.sun.jsp.JspException: Compilation failed

  This is a programming mistake in your JSP. You may have typed something incorrectly or maybe you used two FORMS instead of one.

5.4.5 Category pages

Category pages in the catalog help shoppers navigate through the various groups of products or services available in a store. The first category pages lead shoppers to the areas in which they want to shop. Subsequent category pages further narrow the selected product type that the shopper wishes to browse. The last category page in a shopping path contains links to product pages.

Typically, there are two types of category pages: category pages that link to subcategory pages, and category pages that link to product pages (these are called product list pages). To create either of these two types of category pages, you need to create at least two templates: one that displays a list of links to subcategories of a category, and one that displays a list of links to product pages that belong to a specific subcategory. If you want a category to link simultaneously to subcategories and to products, you can create a template that contains both types of lists.

Category templates usually contain a table (typically, an HTML table that includes dynamic elements), which displays the given number of categories. Graphics and descriptions can identify each category in the table, and hypertext links can connect shoppers to subcategories or products in that category.

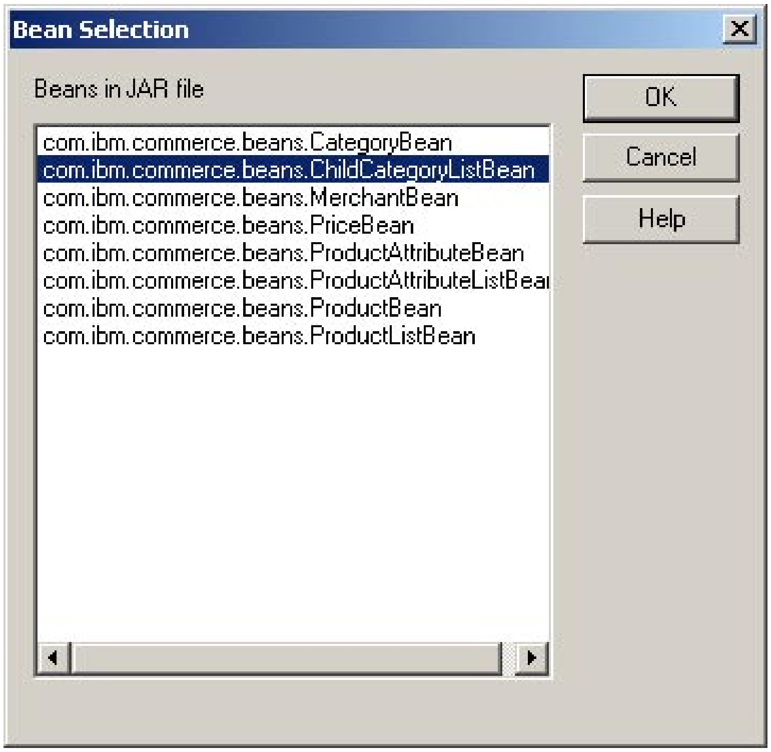
You can create a category template using one of the following methods: with JavaServer Pages (JSP) technology in Page Designer, with Net.Data macros in a text editor, or by modifying the supplied sample product template.

5.4.5.1 Create a category display JSP

Now we will create the category display JSP. To do this create a blank JSP following the procedure in 5.4.2, “Creating a product JSP” on page 141, but this time call it catdisp.jsp.

5.4.5.2 Add a ChildCategoryListBean

1. In Commerce Studio, select the <storedir> folder, where <storedir> is the name of your store.
2. Double-click the appropriate JSP file. The file opens in Page Designer.
3. Ensure that the **Normal** tab at the bottom of the Page Designer window is selected.
4. In Page Designer, in the **View** menu, ensure that **Contents Manager** is selected.
5. In the Contents Pane (the left pane of the Page Designer window), navigate to <drive>:\IBM\CommerceStudio\databeans\lib.
6. From the drop-down list, select **Java Bean Files** (.class; .jar). The available files display in the field below.
7. Select **wcsclient.jar**. Drag and drop the file into the Normal view. The Beans Selection window is displayed.



com.ibm.commerce.beans.CategoryBean
com.ibm.commerce.beans.ChildCategoryListBean
com.ibm.commerce.beans.MerchantBean
com.ibm.commerce.beans.PriceBean
com.ibm.commerce.beans.ProductAttributeBean
com.ibm.commerce.beans.ProductAttributeListBea
com.ibm.commerce.beans.ProductBean
com.ibm.commerce.beans.ProductListBean

Figure 85. ChildCategoryListBean selection

8. Select com.ibm.commerce.beans.ChildCategoryListBean. Click **OK**. The Attribute window is displayed.
9. In the **jsp:useBean** tab, in the Scope drop-down list, select ***request***.
10. Select the **setProperty** tab.
11. In the Property field, type *. Click **Add**.

    Adding the * sets all properties in the bean that match the parameters in the page URL to the values specified in the URL. In this instance, the values for the merchant reference number (merchant_rn) and the category reference number (cgrfnb) are set in the bean.
12. Click **OK**. The J icon is displayed in the Page Designer window.

The next step is to add dynamic category information.

13. From the Insert menu, select **Dynamic Elements** -> **Dynamic Loop**. The Attribute dialog box is displayed.
14. Click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the childCategoryListBean.

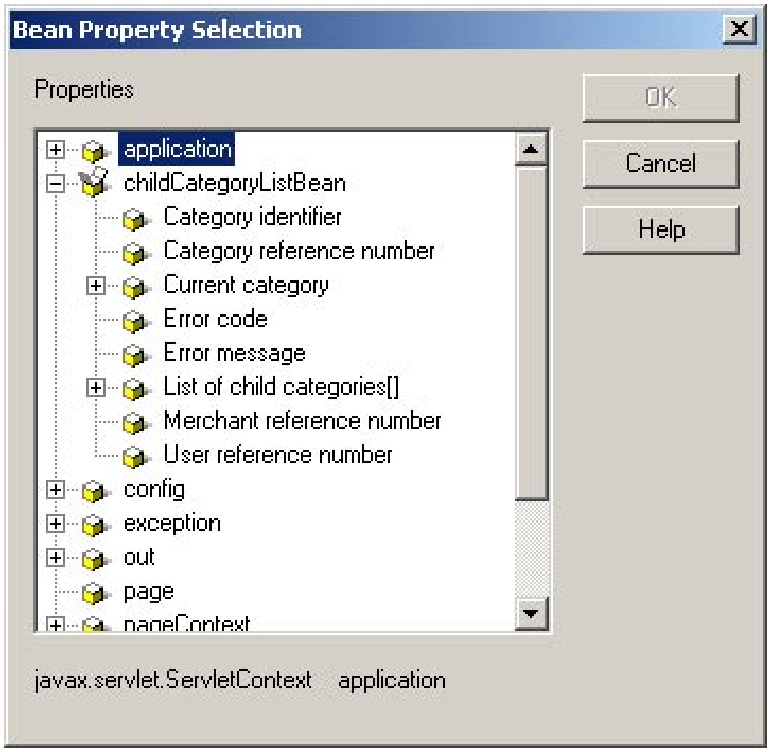


Figure 86. Child Category List Bean properties

15.Select **List of child categories[]** and click **OK**. The childCategoryListBean.childCategory[] is displayed in the Loop Property field in the Attribute dialog box. Click **OK** again and the loop property box is displayed in the main body of the JSP.

16.Place your cursor in the loop property box. As you complete the following steps, ensure your cursor displays within the marks that denote the loop property box.

17.Add dynamic category information to your category page. Most category pages include the category name and a thumbnail image.

18.From the **Insert** menu, select **Dynamic Elements** -> **Property Display**. The Attribute dialog box displays.

19.Click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the childCategoryListBean.

20.Expand **List of child categories[]**, then **Category name**, and then click **OK**. The childCategoryListBean.childCategory[].name is displayed in the Property field in the Attribute dialog box.

21.In the Sample Text field, type `CATEGORY_NAME` and click **OK**.

The sample text is displayed in the category page. When the category page is requested by the shopper, the sample text will be replaced with information from the Commerce Suite database.

22.From the **Insert** menu, select **Dynamic Elements** -> **Dynamic Image**. The Attribute dialog box is displayed.

23.In the SRC Attribute field, click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the childCategoryListBean.

Select **List of child categories[]**, then **Thumbnail image path name**. Click **OK**. The childCategoryListBean.childCategory[].thumbnailPathName is displayed in the SRC Attribute field in the Attribute dialog box. Click **OK**. The sample image is displayed in the category page. When the category page is requested, the sample image will be replaced with the image from the Commerce Suite database.

5.4.5.3 Create links to subcategory pages

Once you have created your category page, you must add links to any subcategory pages. To create links to subcategory pages, complete the following steps:

1. In catdisp.jsp, select the CATEGORY_NAME text. The text displays with a reversed background, and blinks. Press Shift while dragging and left-clicking with the mouse over the text. The text no longer blinks. From this Insert menu, select **Insert Link**. The Attribute dialog box is displayed.
2. Select the **Dynamic URL** tab.
3. In the URL field, type the URL for the category display command: `/webapp/commerce/servlet/CategoryDisplay`
4. In the Parameters box, click **Edit**. The URL Parameter Editor is displayed.
5. In the Name field, type `merchant_rn` (merchant reference number).
6. Select the **Specify by property** checkbox. Click **Browse**. The Bean Property Selection dialog box is displayed.
7. Select **+** to display the properties for the childCategoryListBean.
8. Expand **List of child categories[]**, then select **Merchant reference number**. Click **OK**. Click **Add**.
9. In the Name field, type `cgrfnbr` (category reference number).
10. Select the **Specify by property** checkbox. Click **Browse**. The Bean Property Selection dialog box is displayed.
11. Select **+** to display the properties for the childCategoryListBean.
12. Expand **List of child categories[]**, then select **Category reference number**. Click **OK**. Click **Add**.
13. Click **OK**, then **OK** again.
14. From the **File** menu, select **Save**.
15. Publish catdisp.jsp using Commerce Studio.

5.4.5.4 Test the catalog JSP

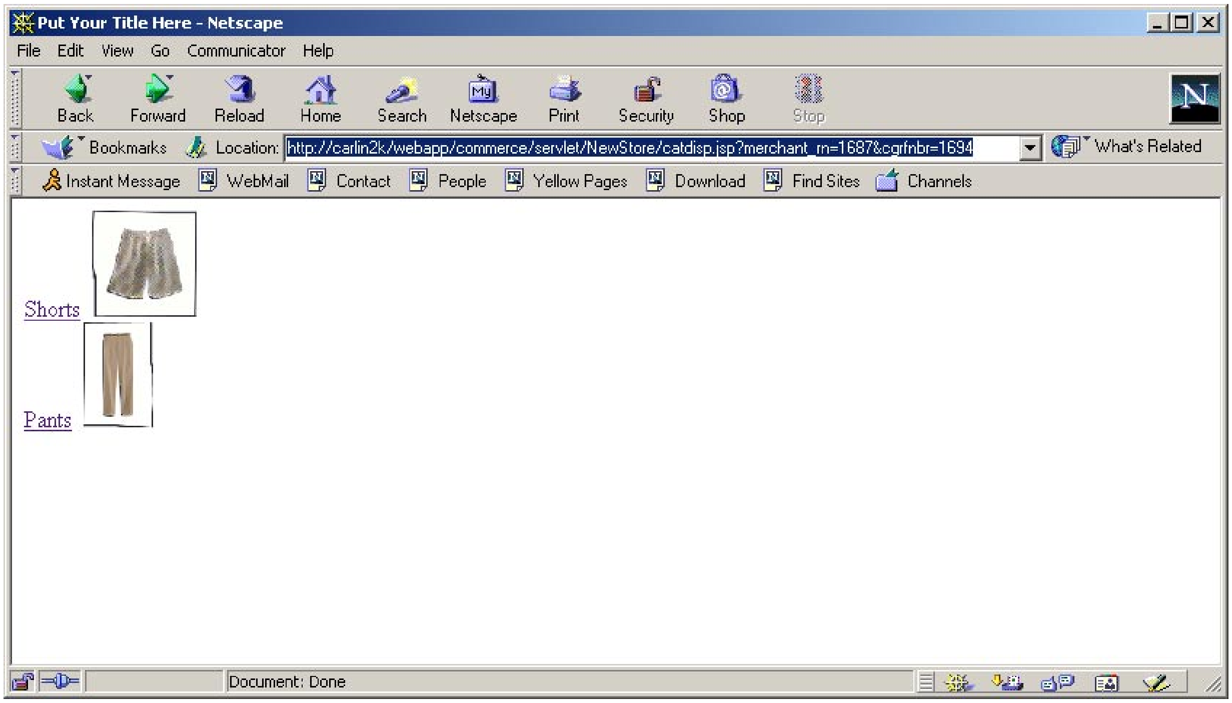
After you have created catalog display JSP you can test that it produces the correct dynamic pages and that the pages display properly.

Use the following URL:

```
<host>/webapp/commerce/servlet/<storeDir>/catdisp.jsp?merchant_rn=xxxx&cgrfnbr=yyyy
```

where <host> is the hostname, <storeDir> is the store directory, xxxx is the merchant reference number, and yyyy is the category reference number (again, both of these numbers can be found by querying the WCSuite PRODUCT table in DB2).

The appropriate catalog page is displayed. It may look something like Figure 87.



Location: http://carlin2k/webapp/commerce/servlet/NewStore/catdisp.jsp?merchant_rn=1687&cgrfnbr=1694

Figure 87. Output from catdisp.jsp

5.4.6 Create a product list JSP

A product list page JSP is a type of category JSP that lists the products within a category. As described in 5.4.6, “Create a product list JSP” on page 154, create another blank jsp and call this one productlist.jsp. This section assumes that you have read that section of this book.

1. After dragging wcsclient.jar to the right-hand pane, select the **productListBean**, again change the scope of the bean to ***request***, and add * as a property on the setProperty tag. Click **OK**.

The ProductListBean includes the property product[], which returns the list of products associated with the current category. We will now add dynamic infomation to the bean.

2. Click just after the J symbol. From the **Insert** menu, select **Dynamic Elements** -> **Dynamic Loop**. The Attribute dialog box is displayed.
3. Click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the productListBean.
4. Select **List of products[]**. Click **OK**. The productListBean.product[] displays in the Loop Property field in the Attribute dialog box. Click **OK**. The loop property box is displayed.
5. Place your cursor in the loop property box. As you complete the following steps, ensure your cursor displays with the hash marks that denote the loop property box.

6. From the **Insert** menu, select **Dynamic Elements** -> **Property Display**. The Attribute dialog box is displayed.
7. Click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the productListBean.
8. Select **List of products[]**, then **Short description**. Click **OK**. The productListBean.product[].shortDescription is displayed in the Property field in the Attribute dialog box.
9. In the Sample Text field, type `PRODUCT_NAME`. The sample text is displayed in the dialog box. When the product list page is requested, the sample text will be replaced with information from the Commerce Suite database.
10. From the **Insert** menu, select **Dynamic Elements** -> **Dynamic Image**. The Attribute dialog box is displayed.
11. In the SRC Attribute field, click **Browse**. The Bean Property Selection dialog box is displayed. Select **+** to display the properties for the productListBean.
12. Select **List of products[]**, then **Thumbnail image path name**. Click **OK**. The productListBean.product[].thumbnailPathName is displayed in the SRC Attribute field in the Attribute dialog box. Click **OK**. The sample image is displayed in the products table. When the product list page is requested, the sample image will be replaced with the image from the Commerce Suite database.

5.4.6.1 Create links to product pages

Once you have created your product list page, you must add links to any product pages.

1. Select the text PRODUCT_NAME. The text is displayed with a black background and blinks. Press and hold Shift while dragging and left-clicking with the mouse over the text. The text no longer blinks.
2. From the **Insert** menu, select **Insert Link**. The Attribute dialog box is displayed.
3. Select the **Dynamic URL** tab.
4. In the URL field, type the URL for the product display command: `/webapp/commerce/servlet/ProductDisplay`
5. In the Parameters box, click **Edit**. The URL Parameter Editor is displayed.
6. In the Name field, type `merchant_rn` (merchant reference number).
7. Select the **Specify by property** checkbox. Click **Browse**. The Bean Property Selection dialog box is displayed.
8. Select **+** to display the properties for the productListBean.
9. Select **List of products[]**, then **Merchant reference number**. Click **OK**, then **Add**.
10. In the Name field, type `prrfnbr` (product reference number).
11. Select the **Specify by property** checkbox. Click **Browse**. The Bean Property Selection dialog box is displayed.
12. Select **+** to display the properties for the productListBean.
13. Select **List of products[]**, then **Product reference number**. Click **OK**, then **Add**.

14. Click **OK**, then **OK** again.

15. From the **File** menu, select **Save**.

16. Publish productlist.jsp to your server using Commerce Studio.

5.4.7 Integrate JSPs into a store

After you have created catalog and product pages using JSP technology, you must integrate them into your store. Stores created with Store Creator, or stores that include the default Net.Data macros, use the C++ CatalogDisplay and ProductDisplay commands to display the existing catalog pages created with Net.Data macros. To integrate your JSP catalog pages into the store, you must replace the C++ CatalogDisplay and ProductDisplay commands with the corresponding Java commands.

The instructions for integrating the catalog make the following assumptions:

- You created a store using the Store Creator, and included the sample products in that store.
- You created the following JSP catalog pages: catdisp, productlist, and product. For more information on how to create these see 5.4.2, “Creating a product JSP” on page 141.
- The JSP catalog pages are in the root of the <storedir> folder in the project directory for your store in Commerce Studio. For more information, see Commerce Suite file structure.

To integrate the JSP catalog pages into a store, complete the following steps:

Optionally delete the following files in the project directory for your store:

```
stores\<storedir>\macros\catdisp.d2w
stores\<storedir>\macros\proddisp.d2w
```

These are the original Net.Data category and product page templates for the store, created by the Store Creator, and will be replaced by the JSP catalog pages.

The following macros, located in stores\<storedir>\macros in the project directory for your store in Commerce Studio, contain C++ CatalogDisplay and ProductDisplay commands:

```
nav.d2w and navbottom.d2w
searchrslt.d2w
regNew.d2w
regUpdate.d2w
shopcart.d2w
```

In the above macros and any others you may have added, replace all instances of the C++ CatalogDisplay and ProductDisplay commands with the following Java commands:

```
/webapp/commerce/servlet/CatalogDisplay?merchant_rn=xxxx&cgrfnbr=yyyy
/webapp/commerce/servlet/ProductDisplay?merchant_rn=xxxx&prrfnbr=zzzz
```

where `xxxx` is the merchant reference number for the store, `yyyy` is the category reference number for the desired category and `zzzz` is the product reference number of the desired product.

Note: If you choose, instead of replacing the entire C++ command, you can opt to replace command with servlet and ensure that the command passes the correct parameters. Typically, you will need to remove either the prmenbr=nnnn parameter in a ProductDisplay command, or the cgmenbr=cccc parameter in a CategoryDisplay command, and add the merchant_rn=xxxx parameter when necessary.

Double-click the store profile (stores\<storedir>\stores.scpf). The Store Profile Editor is displayed.

Select the **Catalog Templates** tab.

Enter the file name in the Category page template file name field by clicking **Browse** and navigating to the category page template created with JSP technology. Repeat for the Product list page template file name and the Product page template file name fields.

Note: Ensure that you enter the fully qualified Commerce Studio path names for the JSP templates, for example, stores\<storeDir>\product.jsp. The path name will be changed to the correct Web format during publishing.

Publish the store. During publishing, the Commerce Suite database is updated with the new templates.

Test the store. The sample products and categories should display in the pages produced by the new templates, and you should be able to browse the catalog and order items successfully.

While testing your store, you may notice that the style of the catalog pages does not match the style of the rest of your store pages. You can change the look and feel of the catalog pages.

5.4.8 Change look and feel of JSP catalog page templates

Style information for the store pages created by the Store Creator is stored in the Storeinfo.inc file. The catalog pages use the style information in the body frame, which is identified as frame 3. To make your pages created by the JSP catalog templates match these styles, copy the style information for frame 3 in this file into a JSP file.

In the project for your store in Commerce Studio, open the Storeinfo.inc file named stores\<storedir>\macros\Storeinfo.inc.

Browse through the file to find the style definitions for your store pages. See the following example:

```
@DTW_ASSIGN(backgroundImageFile3,"")
@DTW_ASSIGN(textColor3,"FFFFFF")
@DTW_ASSIGN(backgroundColor3,"660066")
@DTW_ASSIGN(unvisitedLinkColor3,"CCFFFF")
@DTW_ASSIGN(visitedLinkColor3,"FFFF00")
@DTW_ASSIGN(activeLinkColor3,"FF0033")
```

Select the <storedir> folder in your store.

From the **Insert** menu, select **File**. The Insert File dialog box displays.

In the **Create New** tab, select **Blank.txt**.

Note: Since you are creating a JSP fragment, not a full JSP document, you cannot select Blank.jsp or edit the file with Page Designer, as the current version does not support editing HTML or JSP fragments.

In the File Name field, rename the file to storeinc.txt, or text of your choice. Click **OK**. The file (storeinc.txt) is displayed in Commerce Studio.

Double-click storeinc.txt. The file opens in Notepad or the associated text editor.

Add style information that corresponds to the style information in the Storeinfo.inc file. You must add this information following Java and JSP standards.

All color values are specified by color names, for example Black, Green, olive, red; or hexadecimal numbers, for example, 6699CC. The hexadecimal number identification, #, is required in Java.

See the following example:

```
<%! String backgroundImageFile3 = ""; %>
<%! String textColor3 = "#FFFFFF"; %>
<%! String backgroundColor3 = "#660066"; %>
<%! String unvisitedLinksColor3 = "#CCFFFF"; %>
<%! String visitedLinksColor3 = "#FFFF00"; %>
<%! String activeLinkColor3 = "#FF0033"; %>
```

Rename storeinc.txt to storeinc.jsp.

In Page Designer, add a JSP include directive to your JSP catalog page templates. To do this:

Open your catalog page template.

Position your cursor at the top of the page.

From the **Edit** menu, select **Attributes**. The Attribute window is displayed. Ensure that Document Properties displays in the Tag field.

Select the **JSP Tags** tab.

From the Tag Type drop-down list, select **JSP Directive** -> **include**. Click **Add**. The Edit JSP Element window is displayed.

In the File Name field, click **Browse** and navigate to storeinc.jsp.

Click **OK** twice to close the Edit JSP Element and Attribute windows.

In Page Designer, select the **HTML Source** tab.

Modify the contents of the <BODY> tag to reference the desired Java variables, for example:

```
<BODY bgcolor=<%= backgroundColor3 %> text=<%= textColor3 %>

link=<%= unvisitedLinksColor3 %> vlink=<%= visitedLinksColor3 %> alink=<%=
activeLinkColor3 %>>
```

Note: Once you modify the <BODY> tag, you can no longer use the Normal view in Page Designer to displays the colors. The color variables can only be resolved at run time (not design time). As a result, do not modify the <BODY> tag until you have decided on the desired colors.

If you make further changes to the style of your Net.Data pages using the Storeinfo.inc file, ensure that you also modify your storeinc.jsp file.

Chapter 6. Auction and personalization functions

In this chapter we provide examples of creating an auction and designing a personalized page using the Blaze Rules Server.

This chapter is intended to illustrate the basics of performing these tasks. It is not intended to be a place where we'll elaborate on all the features of the various WCSuite functions we're illustrating. To do so would mean recreating much of the material that's already in place. (See Appendix D, "Related publications and resources" on page 209 for information on these other sources of WebSphere Commerce Suite education and training.)

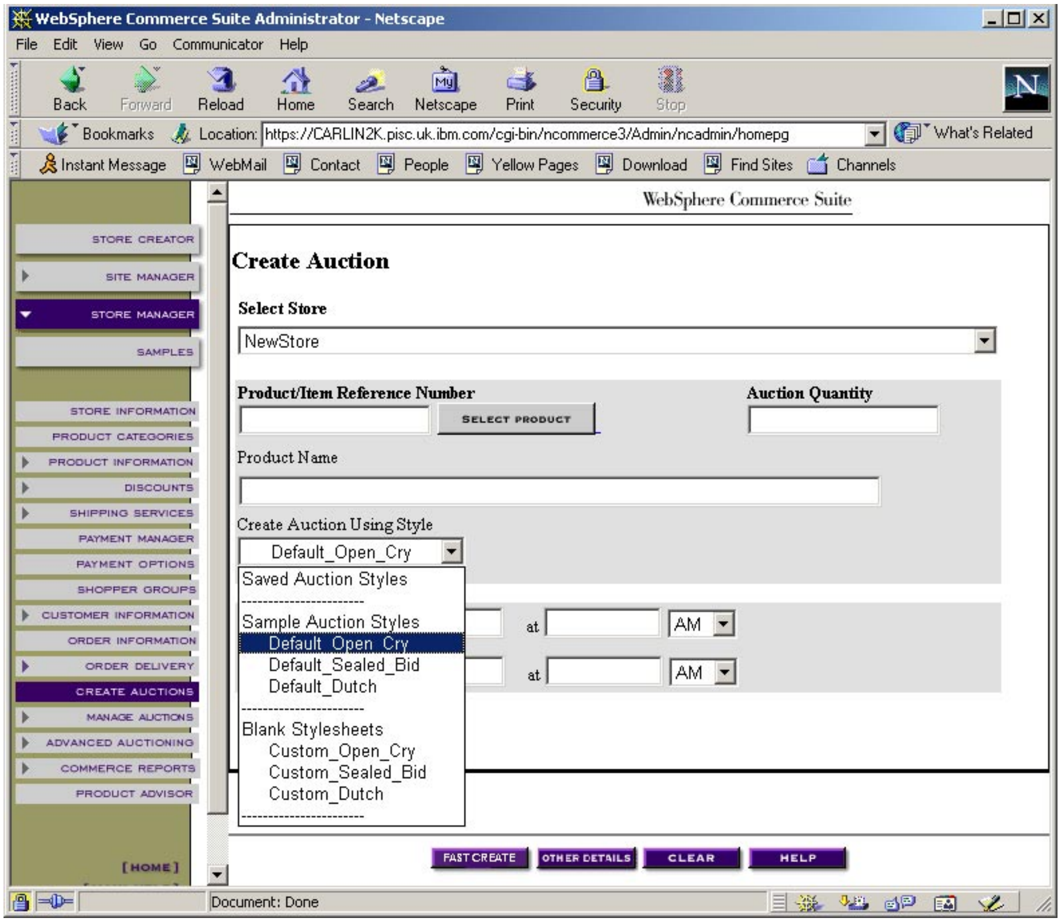
6.1 Creating an auction in your store

One of the highlights of this release of WCSuite is the ability to create and run auctions on your site. In this section we show an example of auctioning a product from your store. Auctions are not created in Commerce Studio. Auction management is achieved through the NCADMIN interface, so start your browser and enter the site URL followed by ncadmin:

```
http://<your store>/ncadmin
```

6.1.1 Create the auction

Once in the NCADMIN browser panel, go to Store Manager and select the **Create Auction** function for the store you created using the Store Creator Wizard in 5.1.2, "Use Store Creator in WebSphere Commerce Studio" on page 127.



WebSphere Commerce Suite Administrator - Netscape

Location: https://CARLIN2K.pisc.uk.ibm.com/cgi-bin/ncommerce3/Admin/ncadmin/homepg

WebSphere Commerce Suite

Create Auction

Select Store

NewStore

Product/Item Reference Number

Auction Quantity

Product Name

Create Auction Using Style

Default_Open_Cry

Saved Auction Styles

Sample Auction Styles
Default_Open_Cry
Default_Sealed_Bid
Default_Dutch

Blank Stylesheets
Custom_Open_Cry
Custom_Sealed_Bid
Custom_Dutch
----

at AM

at AM

Document: Done

Figure 88. Creating an auction with NCADMIN

Make the new auction a default open cry auction and select one of the existing products in the database to auction using the **Select Product** button.

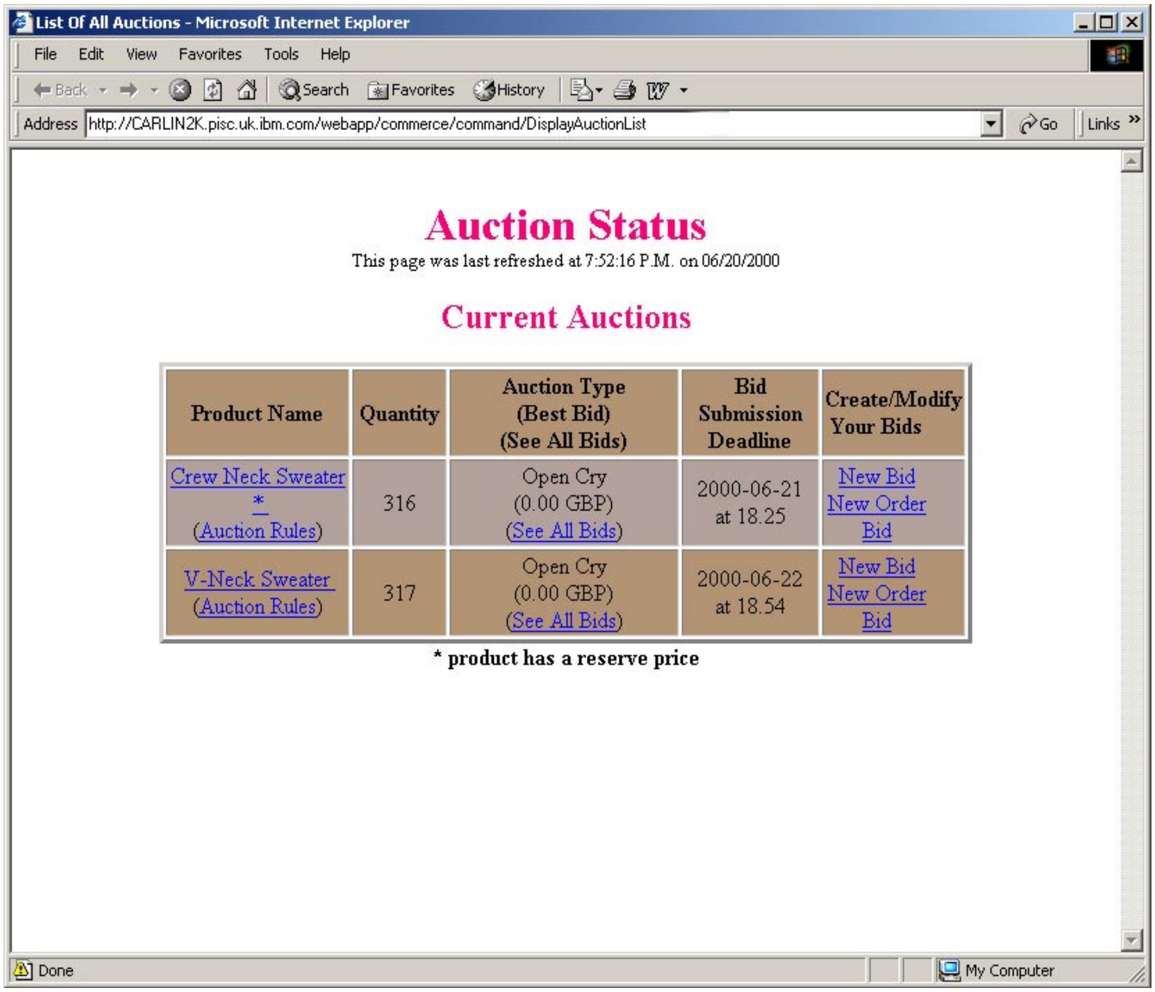
Enter the starting and ending dates of the auction in yyyy-mm-dd format. Enter the starting and ending times of the auction in hh:mm:ss format. For example, for an auction starting on 20th June 2000 at 18:00 you would enter 2000-06-20 and 06:00:00 and make sure the PM drop-down is selected. Note that the Javascripts validating your input are quite strict, so be sure you use exactly the format shown or you will receive “invalid input” messages.

Click the **Other Details** button to see what options you can change for your auction, including adding a description.

Click the **Save** button to save your auction details in the WCSuite database. A reference number for your new auction will be shown.

Verify that you have set up your auctions correctly and display auction status with the command:

```
http://<your server>/webapp/commerce/command/DisplayAuctionList
```



List Of All Auctions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://CARLIN2K.pisc.uk.ibm.com/webapp/commerce/command/DisplayAuctionList
Go Links
Done
My Computer

Auction Status

This page was last refreshed at 7:52:16 P.M. on 06/20/2000

Current Auctions

Product Name	Quantity	Auction Type (Best Bid) (See All Bids)	Bid Submission Deadline	Create/Modify Your Bids
Crew Neck Sweater * (Auction Rules)	316	Open Cry (0.00 GBP) (See All Bids)	2000-06-21 at 18.25	New Bid New Order Bid
V-Neck Sweater (Auction Rules)	317	Open Cry (0.00 GBP) (See All Bids)	2000-06-22 at 18.54	New Bid New Order Bid

* product has a reserve price

Figure 89. Display Auction Status

6.1.2 Verify the auction’s operation

To verify that your auction is working you may wish to place a dummy bid for the product.

Note that in the example shown above you can place a bid because the auction has already started. If you set your auction to start sometime in the future you will not have the option to place a bid until the auction starts.

When you try to place a bid you will be asked to register; at that point you can choose to receive email notification when you are outbid for a product.

6.1.3 Futher exploring auctions

The Metropolitan Mall comes with a demonstration of some of the auction capabilities and functions. For more information, see:

```
http://<your 390 WCSuite server>/demoauct/
```

6.2 Providing a personalized product recommendation

WCSuite provides a rules-based solution for generating personalized contents. Personalization could be used to do product recommendations designed to promote items in a variety of ways, for example, up-sell and cross-sell. Commerce Studio has site development tools as well as asset management tools. The site development tools include the Blaze Advisor Builder as a rule creation environment. This is a Java-based graphical interface that enables the site developer to create, and the business manager to edit the rules that the site uses to make personalized product recommendations.

In this section, using the Demomall's “6ixth Avenue” store, we illustrate how a store can be easily personalized using WCSuite. This tutorial personalizes only the shopcart page for the 6ixth Avenue store. This personalization function introduces a set of products that are recommended as Specials to the shopper when the shopping cart is viewed. The techniques illustrated here could be used to personalize any page of a WCSuite site. We explore only a small set of personalization functions that are possible using WCSuite. For an in-depth understanding of the personalization capabilities, refer to the Commerce Studio online documentation. To personalize your site you will perform the following steps. This tutorial uses these steps to personalize the shopcart page of the 6ixth avenue store.

6.2.1 Create a rulebase using Commerce Studio

Create a new Commerce Studio project by selecting **File** -> **New Project** from the menu bar.

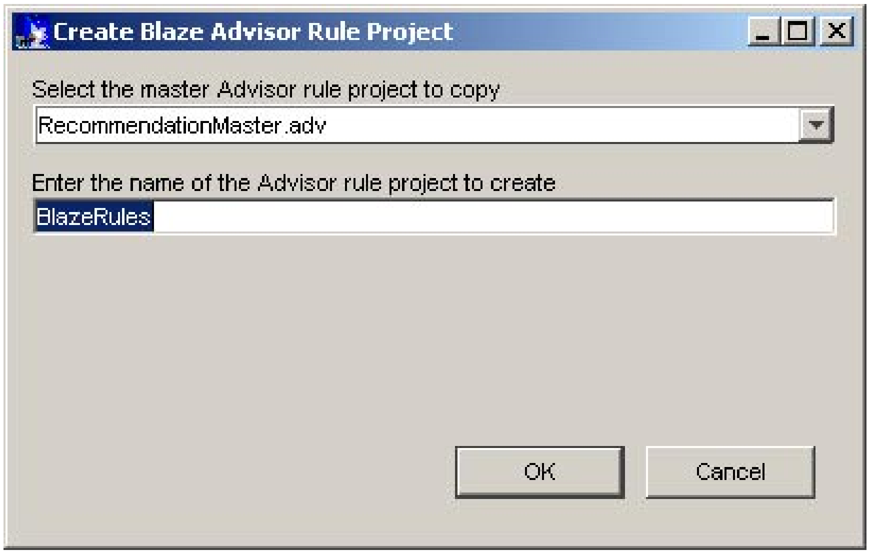
In the Project Name text box, enter: `WCSBlaze` (or text of your choice).

Select Project Template of <none> and click **OK**.

To create an Advisor project inside your Studio project select **Tools** -> **Wizards** -> **Create Blaze Advisor Rule Project** from the menu bar.

From the “Select the master Advisor rule project to copy” drop-down menu select **RecommendationMaster.adv**

In the “Enter the name of the Advisor rule project to create” text field enter `BlazeRules`.



Select the master Advisor rule project to copy

RecommendationMaster.adv

Enter the name of the Advisor rule project to create

BlazeRules

Figure 90. Create Blaze Advisor Rule Project

Click **OK**.

This will create a folder called *rules* in the BlazeRules Commerce Studio project.

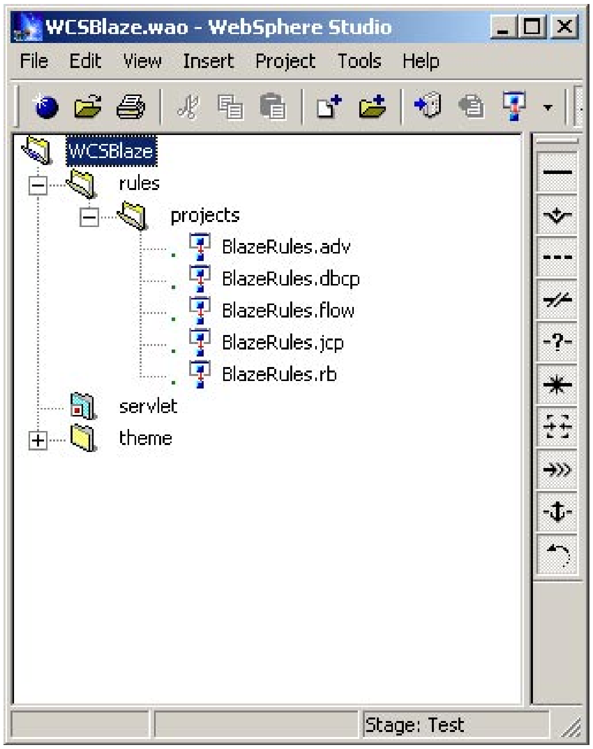


Figure 91. Blaze Advisor assets

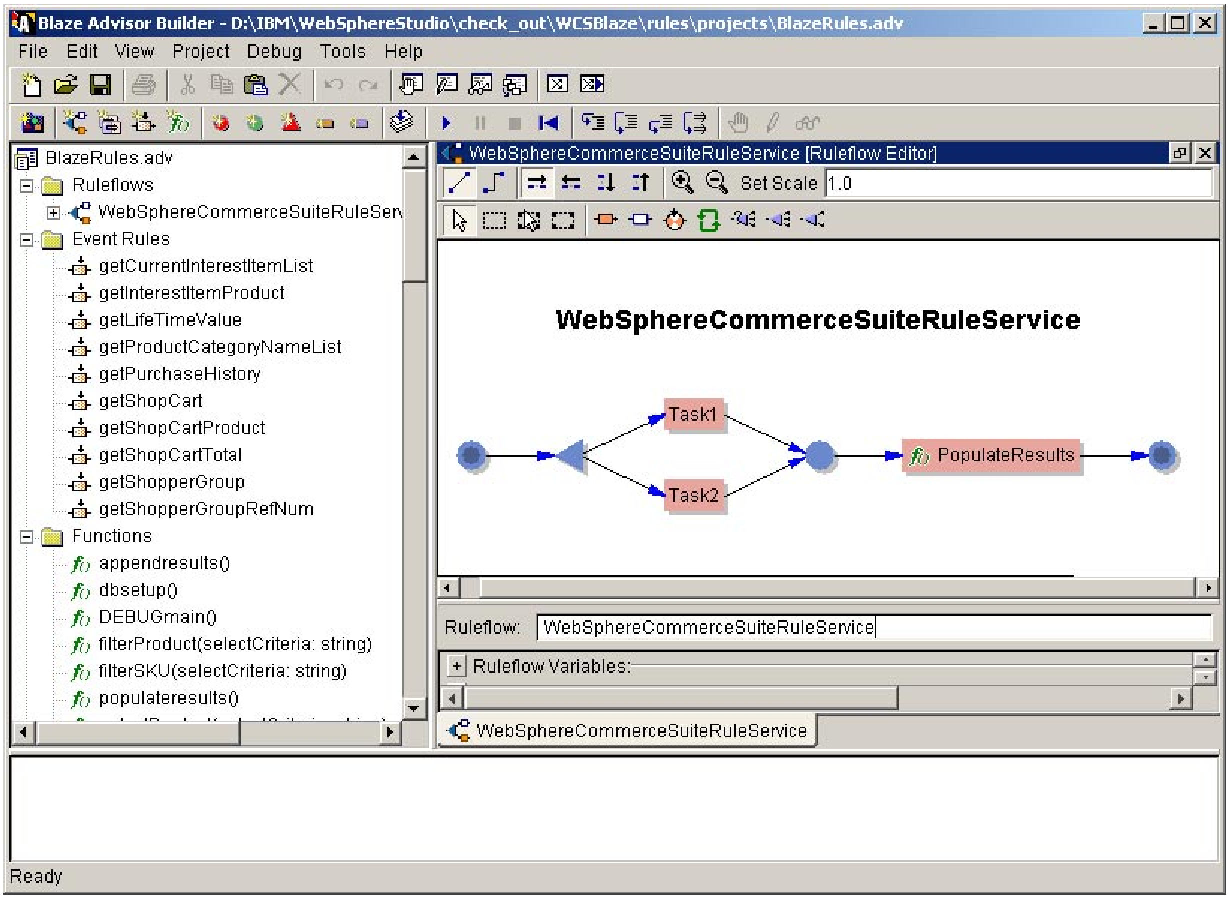
6.2.2 Launch the Blaze Advisor Builder

Rule development is done using the Blaze Advisor Builder. To launch the Advisor Builder for the new project double-click on the **BlazeRules.adv** file. Once you have launched Builder, you can begin creating rulesets and rules.

6.2.3 Modify the Rulebase using Blaze Advisor Builder

Now add a ruleset and some rules to the new Advisor project. We will be creating a new ruleset and rules for the shopcart display page. The implementation of the rules project utilizes the ruleflow element to structure the execution flow in the project. The ruleflow contains a decision point where the decision is made as to which branch is followed.

Add a branch to the ruleflow. To do this, double-click on the **WebSphereCommerceSuiteRuleService** rule flow (on the left) to open the ruleflow editor (on the right).


Blaze Advisor Builder - D:\IBM\WebSphereStudio\check_out\WCSBlaze\rules\projects\BlazeRules.adv
File
Edit
View
Project
Debug
Tools
Help
BlazeRules.adv
Ruleflows
WebSphereCommerceSuiteRuleServ
Event Rules
getCurrentInterestItemList
getInterestItemProduct
getLifeTimeValue
getProductCategoryNameList
getPurchaseHistory
getShopCart
getShopCartProduct
getShopCartTotal
getShopperGroup
getShopperGroupRefNum
Functions
appendresults()
dbsetup()
DEBUGmain()
filterProduct(selectCriteria: string)
filterSKU(selectCriteria: string)
populateresults()
WebSphereCommerceSuiteRuleService [Ruleflow Editor]
WebSphereCommerceSuiteRuleService
Task1
Task2
PopulateResults
Ruleflow Variables:
WebSphereCommerceSuiteRuleService
Ready

Set Scale 1.0

Ruleflow: WebSphereCommerceSuiteRuleService

Figure 92. Graphical view of the Blaze Rules flow

Double-click on the decision block (small triangle) or one of the branches (emerging from this triangle). This opens the decision editor for the *Request_Service_Branch_With_Conditions* decision block

Click the **New Branch** button on the tool bar. This will create a new entry for a branch. Scroll down the editor and in the newly created branch enter the branch condition, replacing *true* with

```
ncRequest.service = "myrecommendation"
```

The value specified (myrecommendation) is the name of the requested ruleset.

Note that the names and string values specified are case sensitive.

Close the decision editor. The ruleflow editor should look like Figure 93.

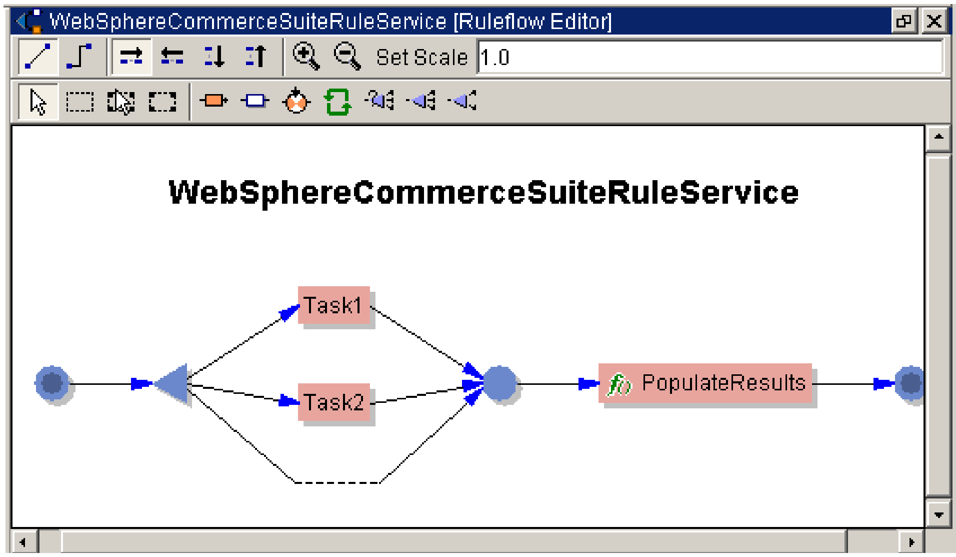

WebSphereCommerceSuiteRuleService [Ruleflow Editor]
Set Scale
1.0
WebSphereCommerceSuiteRuleService
Task1
Task2
PopulateResults

Figure 93. Modified rule flow

You can identify the newly added branch in the Ruleflow editor as the branch with the dotted line in the figure above.

Now we will add a task to the new branch. Click on the **Insert Task** tool bar item (shaped as an orange rectangle). Then click on the dotted line of the new branch. This will add a task called Task1. Double-click on Task1 to open the task editor. Change the Task name to `MyRecommendationTask`, select the Ruleset radio box and from the Choice drop-down menu choose **New Ruleset**. This launches the ruleset editor.

Create the ruleset and its rules. As shown below, in the Ruleset text box enter the rule set name `myrecommendation`.

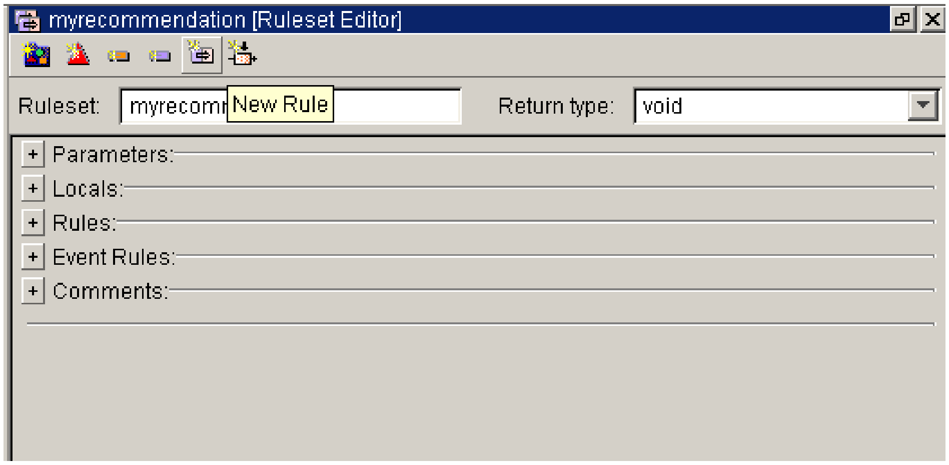


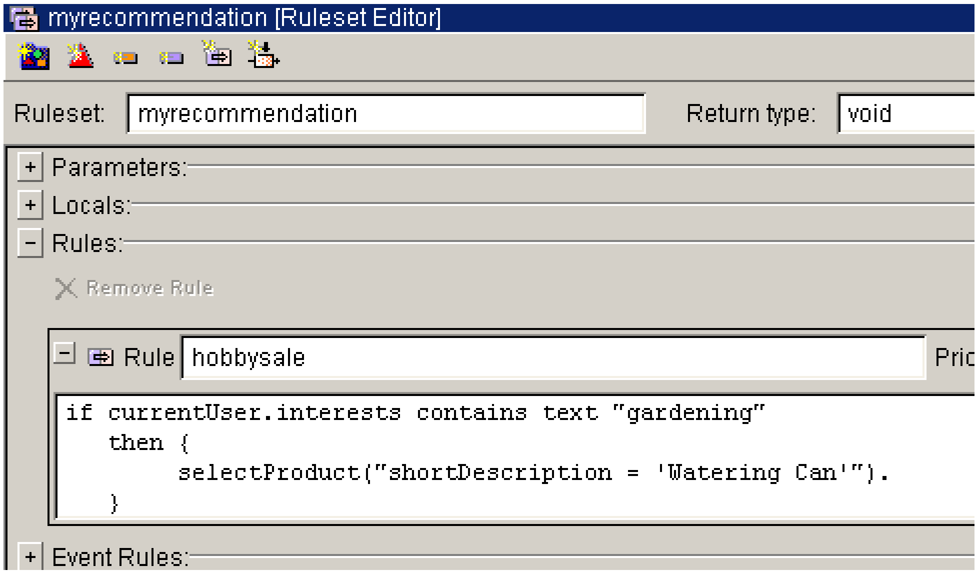
Figure 94. New rule set

Click **New Rule** on the tool bar to create a new rule. This will open a new entry for the rule you just created. We will personalize a page for the 6ixth Avenue Demomall store.

In the Rule text box enter the rule name `hobbysale`.

In the rule definition area enter the following rule code:

```
if currentUser.interests contains text "gardening"
then {
selectProduct("shortDescription = 'Watering Can'").
}
```



```
if currentUser.interests contains text "gardening"
   then {
        selectProduct("shortDescription = 'Watering Can'").
   }
```

Figure 95. Inserting a rule definition

From the menu bar select **Project** -> **Compile** to compile the project. Make sure there are no errors (you should see Compiling..., then Compilation succeeded messages).

Note: We recommend that you consider adding a default rule to the ruleset. This is to handle the case where none of the other rules fire and thus, no products are recommended. An example of a default rule is as follows (use the previous instructions to create a new rule for the current ruleset):

Rule name: `default_rule`

Rule:

```
if true then selectProduct ("shortDescription = 'T-Shirt'")
```

Close the ruleset editor.

Your rules project is now complete. From the menu bar select **Project** -> **Compile** to compile the project. In the message pane you should see the messages “Compiling,” and then “Compilation Successful.”

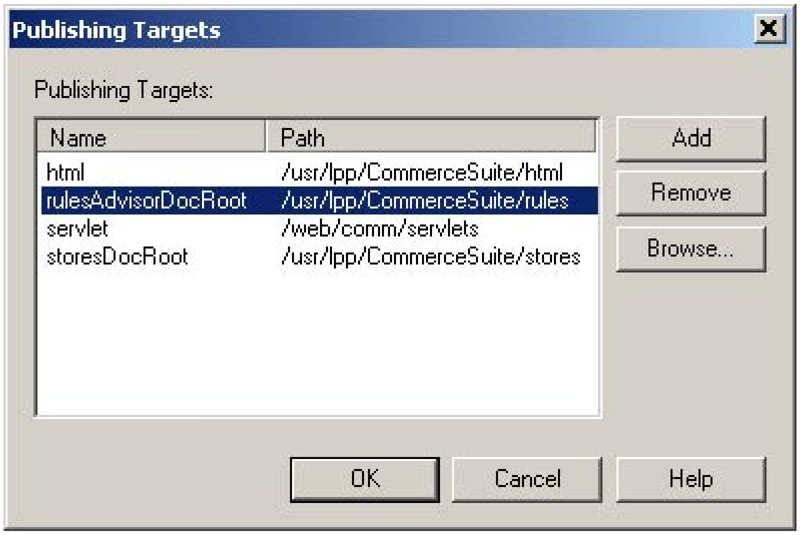
Save the project by selecting **File** -> **Save Project** from the menu bar. Then exit the Advisor Builder.

6.2.4 Publish the rulebase to the WCS server

Now that the rulebase is ready, it needs to be made available to the WCSuite server. This is done by publishing the project. All the associated files for the rules Advisor project need to be published. If any files are missing the project will not run.

From the project in Commerce Studio, right-click on the **rules** folder and from the pop-up menu select **Publish this folder**. We recommend that the “Publish modified files” is unchecked to ensure that all the files in the folder are published.

The files are sent to the WCSuite server and placed in the rulesAdvisorDocRoot, which by default is in /usr/lpp/CommerceSuite/rules. This location can be changed by right-clicking on your target server, selecting properties and then clicking the **Define Publishing Targets** button.



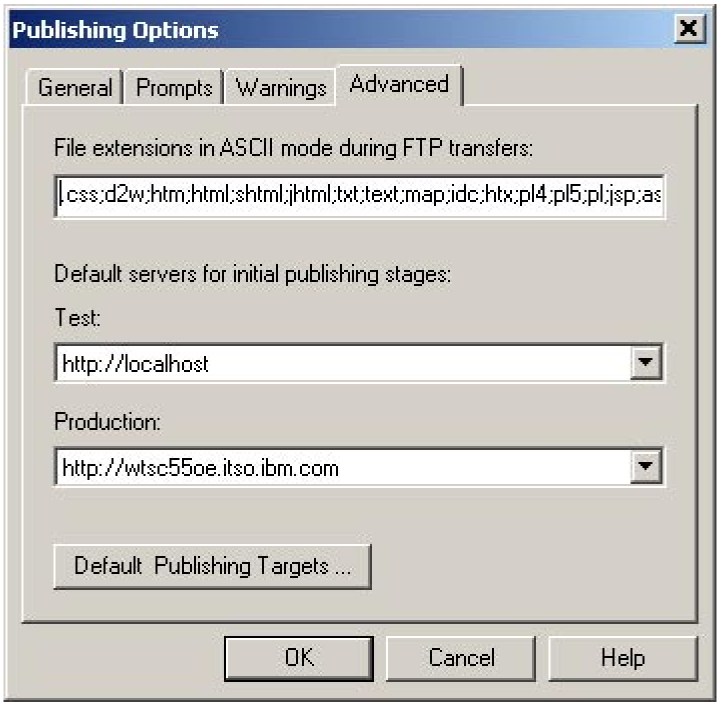
Publishing Targets

Publishing Targets:

Name	Path
html	/usr/lpp/CommerceSuite/html
rulesAdvisorDocRoot	/usr/lpp/CommerceSuite/rules
servlet	/web/comm/servlets
storesDocRoot	/usr/lpp/CommerceSuite/stores

Figure 96. Defining default publishing directory for rules

From the **Advanced** tab of the Publishing Options screen ensure that filetypes .adv, .jcp, .dbcp, .ccp, .flow, and .rb are included in the field File extensions in ASCII mode during FTP transfers (by default they are included).



File extensions in ASCII mode during FTP transfers:

.css;d2w;htm;html;shtml;jhtml;txt;text;map;idc;htx;pl4;pl5;pl;jsp;as

Default servers for initial publishing stages:

Test:

http://localhost

Production:

http://wtsc55oe.itso.ibm.com

Figure 97. Publishing options for personalization

6.2.5 Configure rule service in WCS server using Configuration Manager

If personalization is enabled (in ncommerce.conf) a Java Virtual Machine is created to run the personalization server when the WCSuite instance starts.

To enable personalization add the following (replacing with your own directories) to ncommerce.conf:

```
RULE_SERVER_ENABLED YES
PERS_ENABLED YES
MAX_RECOMMENDATIONS 100
RULE_SERVICE_START_TIMEOUT 60
NC_JRE_PATH /usr/lpp/java18/J1.1/bin
JVM_INITIAL_HEAP_SZ 2097152
JVM_MAX_HEAP_SZ 16777216
JDK_PATH /usr/lpp/java18/J1.1/bin
ADVISOR_SERVER_PATH /usr/lpp/CommerceSuite/AdvSrv/lib
CONFIG_PATH /u/cmnsrv/configs

NC_CLASSPATH
/usr/lpp/java18/J1.1/lib/classes.zip:/usr/lpp/db2/db2610/classes/db2sqljclasse
s.zip:/usr/lpp/db2/db2610/classes/db2sqljruntime.zip:/usr/lpp/CommerceSuite/li
b/pznadvserver.jar:/usr/lpp/CommerceSuite/lib/pznadvclient.jar:/u/cmnsrv/confi
gs:/usr/lpp/CommerceSuite/AdvSrv/lib:/usr/lpp/CommerceSuite/AdvSvr/lib

PZN_USER_CLASSPATH
/usr/lpp/java18/J1.1/lib/classes.zip:/usr/lpp/CommerceSuite/AdvSvr/lib/Advisor
Svr.jar:/usr/lpp/CommerceSuite/AdvSvr/lib/Advisor.jar:/usr/lpp/CommerceSuite/A
dvSvr/lib/xml4j.jar:/usr/lpp/CommerceSuite/AdvSvr/lib/NdJdbc.jar:/usr/lpp/db2/
db2610/classes/db2sqljclasses.zip:/usr/lpp/db2/db2610/classes/db2sqljruntime.z
ip:/usr/lpp/CommerceSuite/lib/pznadvclient.jar:/usr/lpp/Commerce
uite/AdvSvr/lib:/u/cmnsrv/configs
```

```
ADVISOR_SERVER_JARS
AdvisorSvr.jar;Advisor.jar;NdJdbc.jar;xml4j.jar;AdvisorCOM.jar
```

Add the following to your ncommerce.envvars file:

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/CommerceSuite/lib:/usr/lpp/CommerceSuit
e/AdvSvr/lib:/usr/lpp/internet/sbin:/usr/lpp/ldap/lib:/usr/lpp/WebSphere/AppSe
rver/bin:/usr/lpp/java18/J1.1/lib/mvs/native_threads:/usr/lpp/db2/db2610/lib

LD_LIBRARY_PATH=/usr/lpp/java18/J1.1/lib/mvs:/usr/lpp/java18/J1.1/lib/mvs/nati
ve_threads:/usr/lpp/CommerceSuite/lib:/usr/lpp/db2/db2610/lib

CLASSPATH=/usr/lpp/CommerceSuite/lib/pznadvclient.jar:/u/cmnsrv/configs:/usr/l
pp/db2/db2610/classes/db2sqljclasses.zip:/usr/lpp/CommerceSuite/html/en_US/nca
dmin/db2java.zip

DB2SQLJPROPERTIES=/u/cmnsrv/configs/db2sqljjdbc_pzn.properties
```

Note that if you do not include /usr/lpp/CommerceSuite/AdvSvr/lib in either the NC_CLASSPATH (ncommerce.conf) or LIBPATH (ncommerce.envvars) or both, you will see, in the WCSuite SRVOUT, a message from the Personalization Server indicating that the software licence keys cannot be found.

The contents of the file db2sqljjdbc_pzn.properties are:

```
DB2SQLJSSID=<your DB2 subsystem id> (not location name)
DB2SQLJATTACHTYPE=RRSAF
DB2SQLJMULTICONTEXT=NO
DB2CURSORHOLD=NO
```

The Personalization Server looks for the file content-types.properties in the /usr/local/java/lib/ directory. If you do not have this path on your system, you should create the directory and copy the file to it, for example:

```
cp /usr/lpp/java/J1.1/lib/content-types.properties
/usr/local/java/lib/content-types.properties
```

Alternatively, you can create a link.

During testing, set MS_LOGLEVEL 3 in ncommerce.conf and look in the SRVOUT to see the following messages. First look for the JVM starting succesfully:

```
CNM0401D: JVM started successfully
```

Then look for the following message:

```
CMN3201S: Rule server is available.
```

The name of the rules project to use is in an XML file called wcs_advisor.server in the directory pointed to by CONFIG_PATH variable in ncommerce.conf. However, this file has to be in unicode format for the Java personalization server to read.

There is a text version of this file in /usr/lpp/CommerceSuite/install. Copy the file “wcs_advisor.server” from /usr/lpp/CommerceSuite/install/wcs_advisor.server to /usr/lpp/CommerceSuite/html/en_US/instance/wcs_advisor.server.bak.

Change the project name at the tag <Project>. For example: <Project>TutorialRules.adv</Project>.

Convert the file wcs_advisor.server.bak to wcs_advisor.server unicode.

```
iconv -f  IBM-1047 -t ISO8859-1 wcs_advisor.server.bak > wcs_advisor.server
```

When the personalization server starts, it will read this file and load the rules specified on the Project xml tag.

6.2.6 Defining the page containing personalized contents

The rule service is administered using the WCSuite 4.1 administration dialog (or ncadmin).

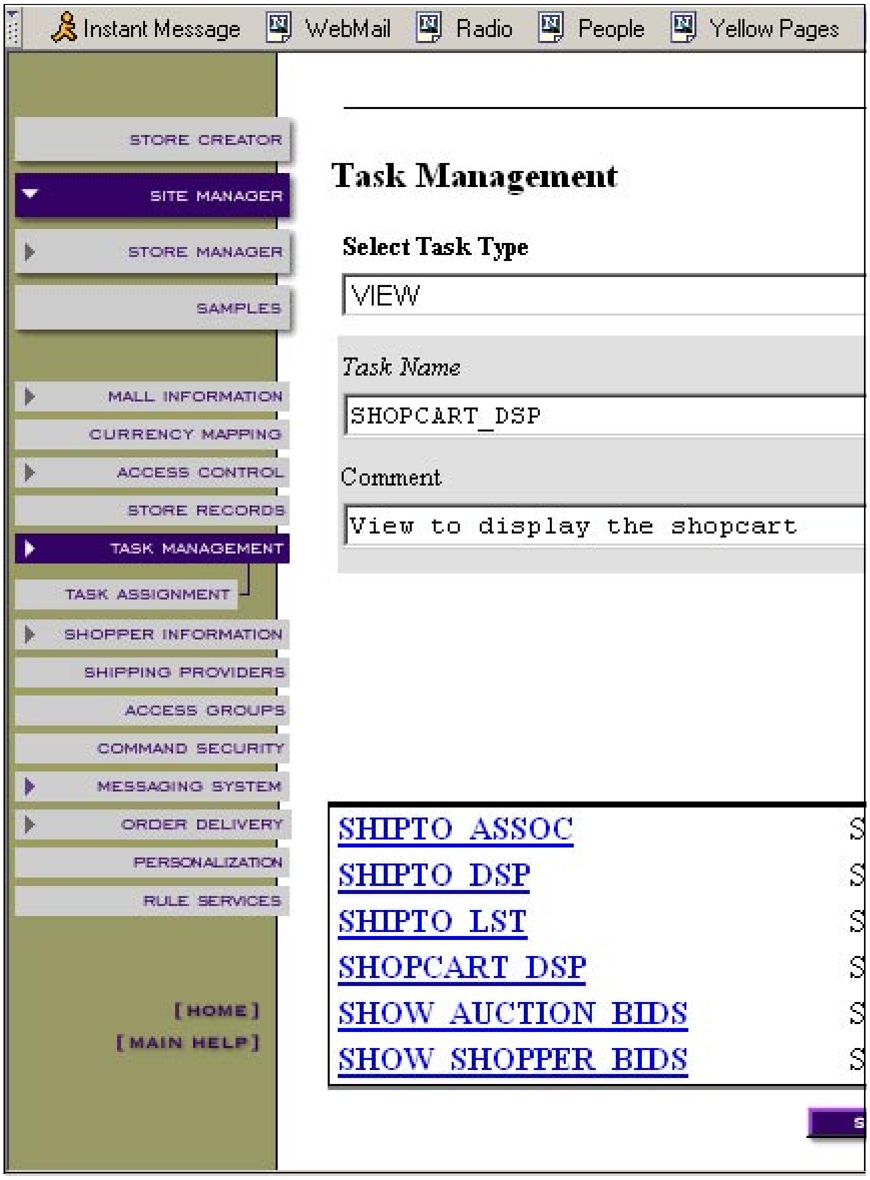
WCSuite “view tasks” are the most logical place to implement your product recommendations since they are the tasks responsible for generating and displaying the dynamic pages in your site. The most common view tasks where personalization can be inserted are: Category display (CAT_DSP View Task), Product display (PROD_DSP View Task), order confirmation (ORD_DSP_PEN View Task) and the shopping cart (SHOPCART_DSP View Task).

We will be personalizing the shopcart display page for the 6ixth Avenue Demomall sample store. To properly personalize this page, we will define a store-level WCSuite Overiddable Function (OF) for shopcart display. To do this follow these steps:

Launch your browser and log on to ncadmin (`http://<your_server>/ncadmin`). To assign the TaskDisplay to SHOPCART_DSP, select **Site Manager** -> **Task Management** and select View for the task type.

If the lower frame listing the available tasks is not visible, drag its borders to a size big enough that the list of tasks becomes visible.

Scroll the lower frame list and select **SHOPCART_DSP**. This will populate the fields in the upper frame.



Task Management

Select Task Type

VIEW

Task Name

SHOPCART_DSP

Comment

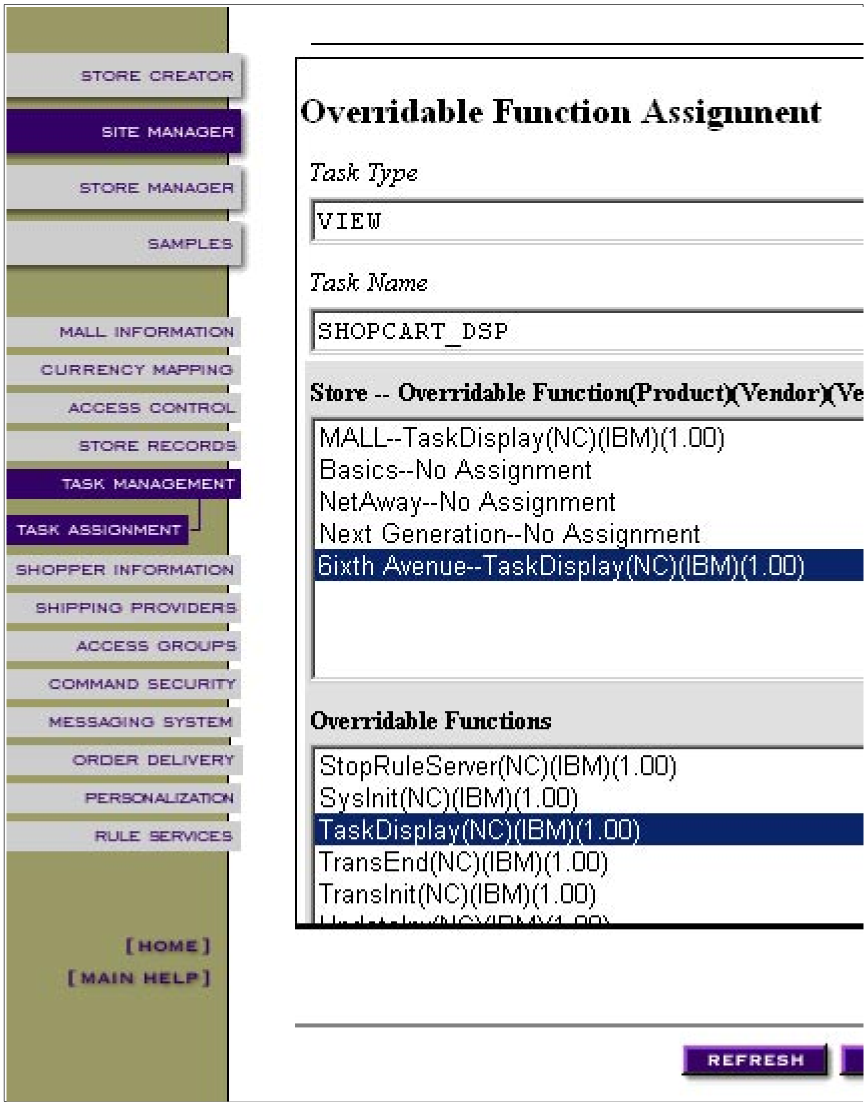
View to display the shopcart

SHIPTO_ASSOC
SHIPTO_DSP
SHIPTO_LST
SHOPCART_DSP
SHOW_AUCTION_BIDS
SHOW_SHOPPER_BIDS

Figure 98. Selecting the SHOPCART_DSP function

Click **Task Assignment** from the menu on the left and then select the **Overiddable Function** button in the right frame.

The 6ixth Avenue Store will probably not have an OF assigned to it. Assign the TaskDisplay OF to the 6ixth Avenue store and click **Update**. Note that to find the TaskDisplay OF you may have to adjust the pane divider down on the right.



Overridable Function Assignment

Task Type

VIEW

Task Name

SHOPCART_DSP

Store -- Overridable Function(Product)(Vendor)(Ve

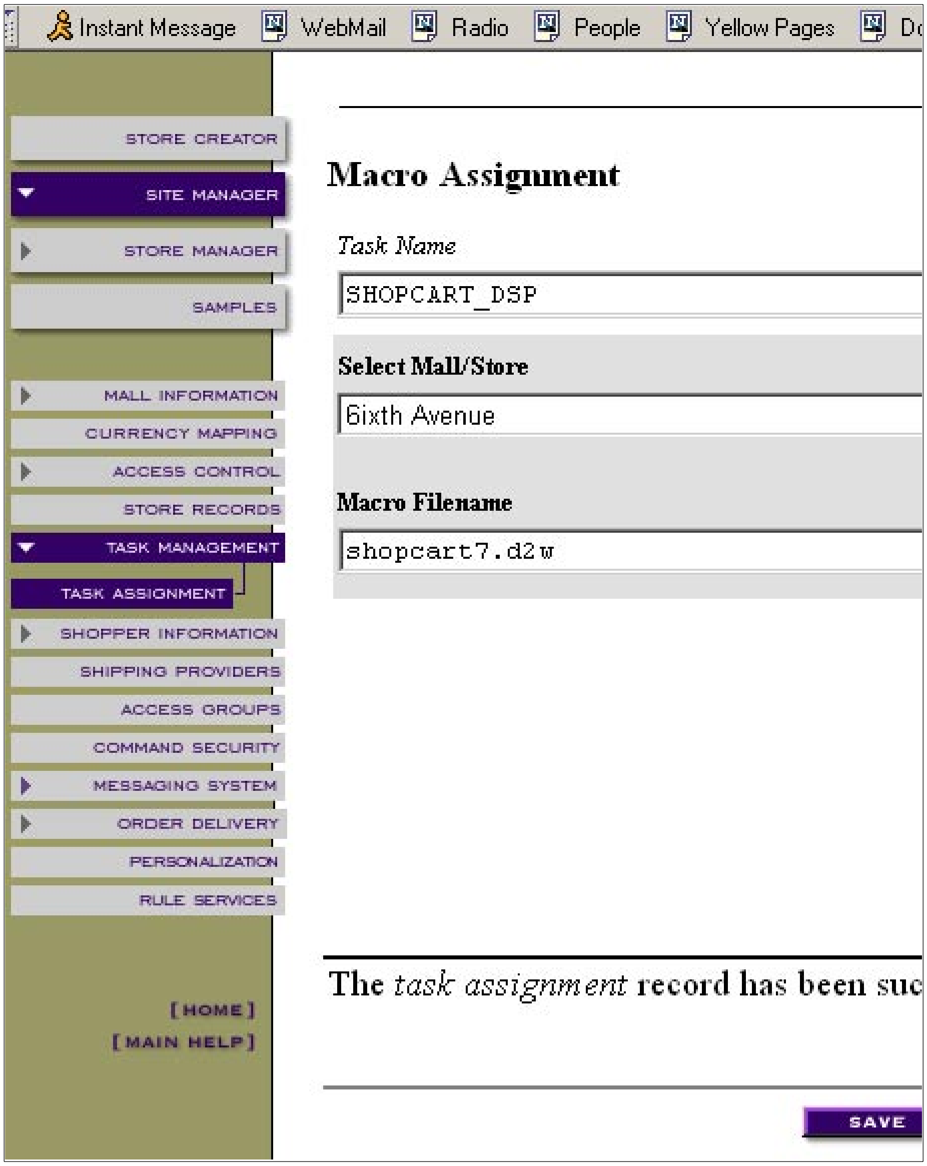
MALL--TaskDisplay(NC)(IBM)(1.00)
Basics--No Assignment
NetAway--No Assignment
Next Generation--No Assignment
6ixth Avenue--TaskDisplay(NC)(IBM)(1.00)

Overridable Functions

StopRuleServer(NC)(IBM)(1.00)
SysInit(NC)(IBM)(1.00)
TaskDisplay(NC)(IBM)(1.00)
TransEnd(NC)(IBM)(1.00)
TransInit(NC)(IBM)(1.00)

Figure 99. Assigning the TaskDisplay Overiddable function

To assign our own macro to the task, click **task management** again. This time select the **macro** button. Change the store to **6ixth Avenue** and assign a macro called shopcart7.d2w to the task. Click **save** to save the changes. You should see a message “The task assignment record has been successfully updated in the database.”



Macro Assignment

Task Name
SHOPCART_DSP

Select Mall/Store
Sixth Avenue

Macro Filename
shopcart7.d2w

The *task assignment* record has been suc

Figure 100. Assigning the shopping cart display macro

Shopcart7.d2w does not exist in the WCSuite product; it is a copy of /usr/lpp/CommerceSuite/models/demomall/macro/en_US/shopcart6.d2w. So copy the shopcart6.d2w macro to shopcart7.d2w in the same dirextory.

Look in shopcart7.d2w. There are three new sections for personalization. The first section looks like the following:

```
PR1 = ProdRec1 ? "$(ProdRec1)" : "-1"
PR2 = ProdRec2 ? "$(ProdRec2)" : "-1"
PR3 = ProdRec3 ? "$(ProdRec3)" : "-1"
PR4 = ProdRec4 ? "$(ProdRec4)" : "-1"
PR5 = ProdRec5 ? "$(ProdRec5)" : "-1"
PR6 = ProdRec6 ? "$(ProdRec6)" : "-1"
PR7 = ProdRec7 ? "$(ProdRec7)" : "-1"
PR8 = ProdRec8 ? "$(ProdRec8)" : "-1"
PR9 = ProdRec9 ? "$(ProdRec9)" : "-1"
PR10= ProdRec10? "$(ProdRec10)": "-1"
```

This assigns the recommended product ID values to PR1 to PR10. If no recommendations are returned, a value of “-1” is set for the variable.

The second section defines a function that will query the database to get the product information that will later be used to display the recommendations for the shopper:

```
%FUNCTION(DTW_ODBC) DisplayRecommendation(IN productRefNum) {
    SELECT prrfnbr, prsdesc,prmenbr
    FROM product
    WHERE prrfnbr=$(productRefNum)

    %REPORT {
        %ROW {
            <TD><A 
HREF="/cgi-bin/ncommerce3/ProductDisplay?prrfnbr=$(V_PRRFNBR)&prmenbr=$(V_PRME
NBR)">$(V_prsdesc)</a></TD>
        %}
    %}

    %MESSAGE {
        default: { %} : continue
    %}
```

The third insert (in the HTML_REPORT section of the macro) actually calls the DisplayRecommendation function defined above to display the recommended product data to the shopper.

```
%if("$(PR1)" == "-1")
<br>Sorry, no suggestions at this time!
%else
<TABLE WIDTH=530 CELLPADDING=0 CELLSPACING=0 BORDER=0>
    <tr>@DisplayRecommendation(PR1)</tr>
    <tr>@DisplayRecommendation(PR2)</tr>
    <tr>@DisplayRecommendation(PR3)</tr>
    <tr>@DisplayRecommendation(PR4)</tr>
    <tr>@DisplayRecommendation(PR5)</tr>
    <tr>@DisplayRecommendation(PR6)</tr>
    <tr>@DisplayRecommendation(PR7)</tr>
    <tr>@DisplayRecommendation(PR8)</tr>
    <tr>@DisplayRecommendation(PR9)</tr>
    <tr>@DisplayRecommendation(PR10)</tr>
</TABLE>
%endif
```

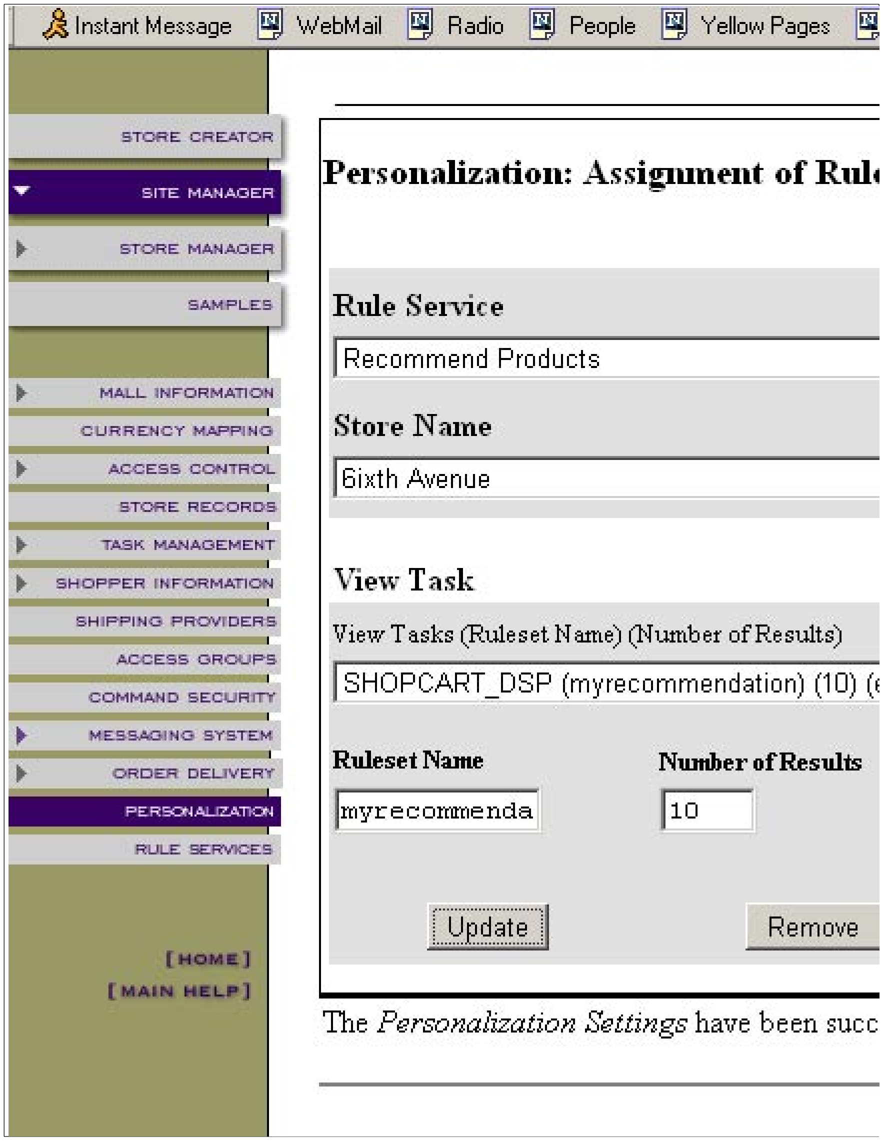
Note that the DisplayRecommendation function is only called if the value of the variable is not “-1”, which in insert one above was set for the condition when zero products are recommended.

Now that the task is assigned at the store level, we can begin administering the Product Recommendation rule service. To do this we will define the ruleset to the associated task.

Click **Site Manager**, then **Personalization**. At this point, if your Rule server is not started you will see a message telling you that it is not available and suggesting that you start the server and try again. If this is the case, check that you followed the configuration steps for enabling personalization.

For the rule service choose **Recommend Products**, for store name choose **6ixth Avenue**, and for view task choose **SHOPCART_DSP**. In ruleset name enter `myrecommendation` (note that this is not really the ruleset name but the value you entered in the rule branch condition (ncRequest.service = “myrecommendation”). For our case we used the same ruleset name as that value. Enter `10` for the number of results and ensure that the enable check box is selected.

Finally, click **Update**. You should see a message that the Personalization Settings have been successfully updated in the database.



Personalization: Assignment of Rul

Rule Service

Recommend Products

Store Name

6ixth Avenue

View Task

View Tasks (Ruleset Name) (Number of Results)

SHOPCART_DSP (myrecommendation) (10) (e

Ruleset Name	Number of Results
myrecommenda	10

The *Personalization Settings* have been succ

Figure 101. Assigning and enabling your personalization rules

This last update associates the ND_RecommendProducts OF with the shopcart display task. Now if you do a DB2 query of the task_mer_of table you should see something like the following:

```
select * from cmnsrv.task_mer_of where TASK_RN=6;
---------+---------+---------+---------+---------+---------+--
    TASK_RN  MERCHANT_RN        OF_RN     SEQUENCE
---------+---------+---------+---------+---------+---------+--
          6     ------              2         ----
          6         2066         9065            1
          6         2066            2            2
DSNE610I NUMBER OF ROWS DISPLAYED IS 3
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The first OF called (sequence number 1) has an OF_RN of 9065; this is the ND_RecommendProducts OF. The second OF (sequence number 2) is TaskDisplay.

Stop and restart the WCSuite server for all changes to take effect.

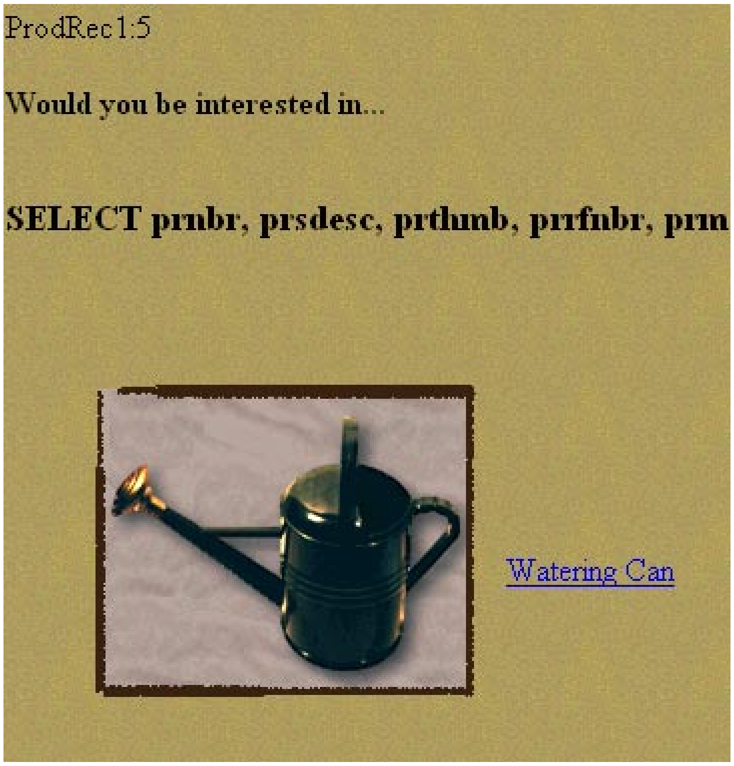
6.2.7 Verify personalization occurs when page is accessed

Always ensure that the caching function is off for the personalized page. In this case we have personalized the output to Shopcart Display, which is not cached by default (and never should be cached).

Using your browser, go to the demomall (`http://<your_server>/demomall/basemall.htm`) and click on **Registered User**. Enter a userid and password (for example, ncadmin) and then click on **6ixth Avenue store**. Click on **shopping cart** from the menu bar near the bottom of the page.

At this stage the recommendation rule will not yet suggest any products because the “gardening” interest or hobby has not be specified for the registered user ncadmin.

To add gardening as a hobby for the user, follow these steps. Click **Register** from the menu bar on the shopcart page. On the registration page enter values for Last name, Address, City, Zip/Postal Code and Country. Scroll further down and in the Interest/Hobby field enter `gardening` (Remember to use lower case characters for gardening since that is how we wrote the rule). Click the **Change Registration** button and then click **Proceed to Mall Directory**. Finally, click the **6ixth Avenue Store** again and select the shopping cart. You should see the recommended product.



ProdRec1:5

Would you be interested in...

SELECT prnbr, prsdesc, prthmb, prrfnbr, prm

Figure 102. Personalized product recommendation

6.2.8 Further personalization information

There is futher information available in a PDF file in the WCSuite HFS directory structure at /usr/lpp/CommerceSuite/AdvSvr/doc/installOS390.pdf. Another very good source of information is the WCSuite online help.

6.3 Using Mass Import to load additional products into your store

You can use either the WCSuite Administrator or the Mass Import utility to append or update database records in a store. You can use the Mass Import utility to populate an empty database with products, or to append and update data in an existing database. Mass Import is an efficient method to initially populate a store or to update numerous products at once.

By default, the Mass Import utility imports data to product- and category-related tables; however, you can import data to any WCSuite table.

A description and use of the Mass Import utility is documented in /usr/lpp/CommerceSuite/books/en_US/csutil.pdf.

In the previous version of WCSuite (called Net.Commerce V3.1.2) the input file to the Mass Import utility was a flat delimited file. Each line in the file represented a “command” that loaded a specific table. Not all tables could be loaded by the utility and Mass Import was generally most useful for small numbers of input records (hundreds to several thousand), but tended to fall short in the area of performance for large and very large input files.

To address this, the Mass Import utility has been enhanced in WCSuite 4.1.

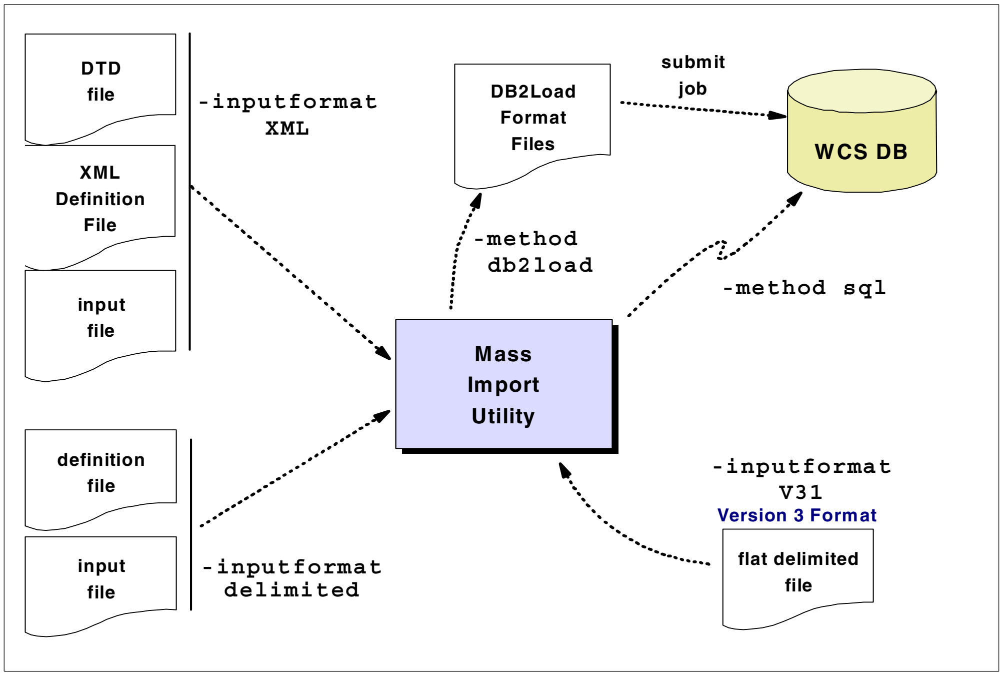


Figure 103. Mass Import Utility

Figure 103 illustrates the various options of the Mass Import utility in WCSuite V4.1. There are now three different input formats:

- XML
- An extended delimited format
- The same format as used with V3.1.2 (backwards compatibility), not discussed in this book

In addition, two output formats are now provided:

- Direct load of the tables using SQL commands. This is very similar to what was done in V3.1.2.
- Using the DB2LOAD utility. This option produces the JCL job streams used by the DB2LOAD utility, which must then be submitted separately from Mass Import. This option provides a much faster load.

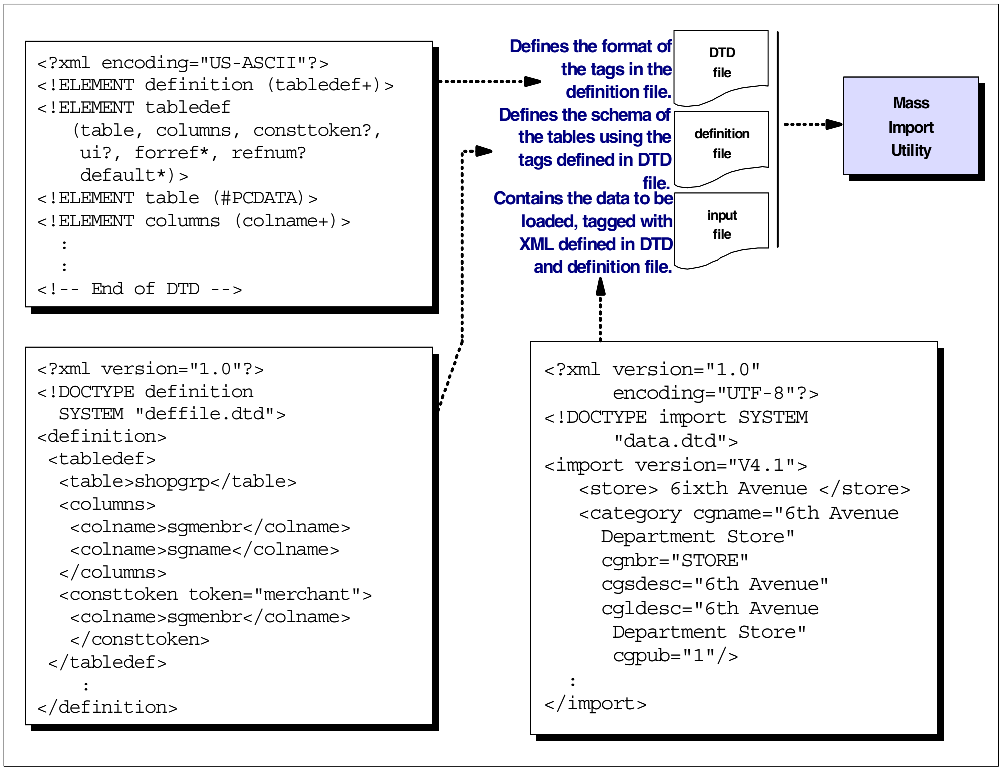
Which input format you use is really a matter of which you feel most comfortable with. XML and delimited format are equally flexible, but since XML is fast becoming an industry standard it is the method we recommend you use, and is the only input method discussed in detail in this book. The output method you employ does have some implications, which we discuss later.

The Mass Import utility registers all imported tables into the KEYS table automatically (if it has not been registered previously) and the information that is stored in the KEYS table will be used to set primary keys for new rows. The Mass Import command will automatically update the KEYMAXID column in the KEYS table, so do not use any other method to import data other than the Mass Import command or the next time the Mass Import utility is used, it will use the obsolete information from the KEYS table and this will result in an import error.

6.3.1 XML input format

There are four files needed for the XML input:

- The schema.dtd file. This is an XML dtd (data tag definition file) that is shipped with WCSuite. *Do not* modify this file. It provides a set of definitions for the tags that are used in the definition files.
- The next file is the definition file. This is also an XML file, which defines the schema of the database. In this sense it is very similar to the definition file for the delimited format input. Modify this file if you add columns or tables to the default schema of the WCSuite database, or if you want to import data to tables other than category- and product-related tables. A default file is supplied with WCSuite (/usr/lpp/CommerceSuite/xm/massimpt/schema.xml).
- A syntax or synfile. This is another XML dtd file, which you create using massimpt with the -synfile option. It provides a set of definitions for the tags that will be used in the next file (which is known as the definition file). An example is in (/usr/lpp/CommerceSuite/xml/massimpt/data.dtd).
- The last file is the input file, which defines the actual data that will be loaded into the tables. A sample is shipped with WCSuite (/usr/lpp/CommerceSuite/models/demomall/db/SixthAvenuedata.xml.in).



```
<?xml encoding="US-ASCII"?>
<!ELEMENT definition (tabledef+)>
<!ELEMENT tabledef
   (table, columns, consttoken?,
    ui?, forref*, refnum?
    default*)>
<!ELEMENT table (#PCDATA)>
<!ELEMENT columns (colname+)>
  :
  :
<!-- End of DTD -->
```

```
<?xml version="1.0"?>
<!DOCTYPE definition
  SYSTEM "deffile.dtd">
<definition>
 <tabledef>
  <table>shopgrp</table>
  <columns>
   <colname>sgmenbr</colname>
   <colname>sgname</colname>
  </columns>
  <consttoken token="merchant">
   <colname>sgmenbr</colname>
   </consttoken>
 </tabledef>
    :
</definition>
```

```
<?xml version="1.0"
      encoding="UTF-8"?>
<!DOCTYPE import SYSTEM
      "data.dtd">
<import version="V4.1">
   <store> 6ixth Avenue </store>
   <category cgname="6th Avenue
     Department Store"
     cgnbr="STORE"
     cgsdesc="6th Avenue"
     cgldesc="6th Avenue
      Department Store"
      cgpub="1"/>
  :
</import>
```

Defines the format of the tags in the definition file.
DTD file
Defines the schema of the tables using the tags defined in DTD file.
definition file
Contains the data to be loaded, tagged with XML defined in DTD and definition file.
input file
Mass Import Utility

Figure 104. Files needed for Mass Import XML option

6.3.2 Run Mass Import with -synfile parameter to generate syntax file

If you have changed schema.xml, run the `massimpt` command with the -synfile option to create the dtd syntax file. The format of the command is:

```
massimpt -infile <your_data>.xml.in -o <DB OWNER> -p <PLAN NAME> -s <DBS
SUBSYSTEM NAME> -log <LOGFILE NAME> -deffile <your_schema>.xml -synfile
<SYNFILE NAME> -method SQL -mode update -inputformat XML
```

An example of which is:

```
massimpt -infile my.data.xml.in -o CMNSRV -p MSERVER -s DBS2 -log mylog
-deffile /usr/lpp/CommerceSuite/xml/massimpt/schema.xml -synfile data.dtd
-method SQL -mode update -inputformat XML
```

This will create a file called data.dtd. Always check your logfile for any messages.

Note that if you miss off the -p <PLAN NAME> (which is not used on the distributed platforms) you may see an SQLSTATE 58006 error message. This means that there wasa problem in connecting to the database.

6.3.3 Create a definition file

The default definition file is schema.xml. The sample definition file imports data to the category, product, and related tables. If you have extended the Commerce Suite database (by adding columns or tables), or if you want to import data to tables other than category- and product-related tables, you will need to create a new definition file that describes the database schema.

Note: In the definition file all column and table names must be in lowercase.

To create a definition file, refer to the CommerceSuite utilities book /usr/lpp/CommerceSuite/books/en_US/csutil.pdf.

6.3.4 Run mass inport to load the database

Once youhave the required definition and input files, run the mass import command to load the product information into the WCSuite database. A working example of a mass import command is:

```
massimpt -infile my.data.xml -o CMNSRV -p MSERVER -s DBS2 -log mylog -err mylog
-f /usr/lpp/CommerceSuite/html/en_US/test/ncommerce.conf -method sql -mode
update -deffile new_schema.xml -synfile data.dtd -inputformat XML
```

Appendix A. Common problems and their causes

This appendix provides help with some common problems that we saw during the writing of this redbook and additional problems we anticipate some users might encounter.

A.1 Problems invoking SSL on Webserver

If your Webserver is not configured for SSL, you will need to do that before you can effectively use WebSphere Commerce Suite. This section gives you a quick guide to setting up SSL on the IBM HTTP Server for OS/390.

Note: This set of instructions assumes you'll use a "self-signed" certificate. That works well for initial testing purposes, but will not suffice when it comes time to go into production. You will eventually need to request and receive a signed certificate from a well-known and respected "Certificate Authority" such as Verisign.

We will use the IKEYMAN utility that comes with the HTTP Server.

A.1.1 Overview of IKEYMAN process to create self-signed certificate

If you're unfamiliar with the concept of public-private key pairs, certificates and Certificate Authorities, the process of using IKEYMAN to generate a self-signed certificate may seem confusing. This section provides an overview of the concept; we present the step-by-step details in the following sections.

A.1.1.1 If you were using a real Certificate Authority

Figure 105 illustrates what you would do with IKEYMAN when requesting a certificate from a real Certificate Authority.

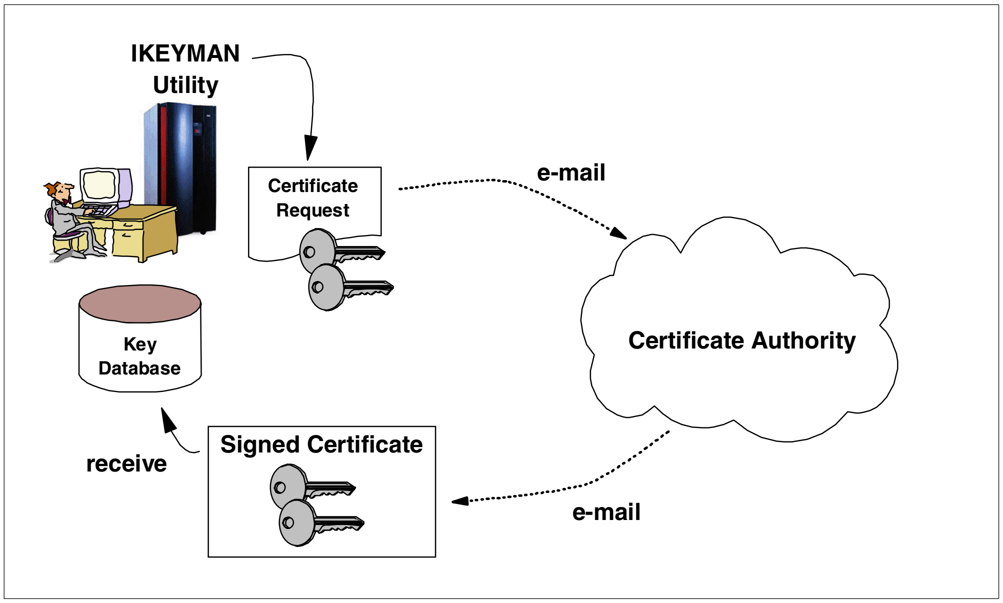


Figure 105. Overview of process where real CA is involved

The process is pretty straightforward: use IKEYMAN to generate a certificate request and e-mail that request (which is a file) to the CA. The CA will sign the request (which means they'll include in the certificate their digital signature) and e-mail it back to you. You then receive the request into your key database.

A.1.1.2 Acting as your own Certificate Authority

For testing purposes, acting as your own CA saves time. However, it makes the process a bit more complex because you have to do for yourself what the CA does as shown in Figure 105 on page 183.

Figure 106 illustrates the steps in the process when you are acting as your own CA, which is the process we cover in this redbook.

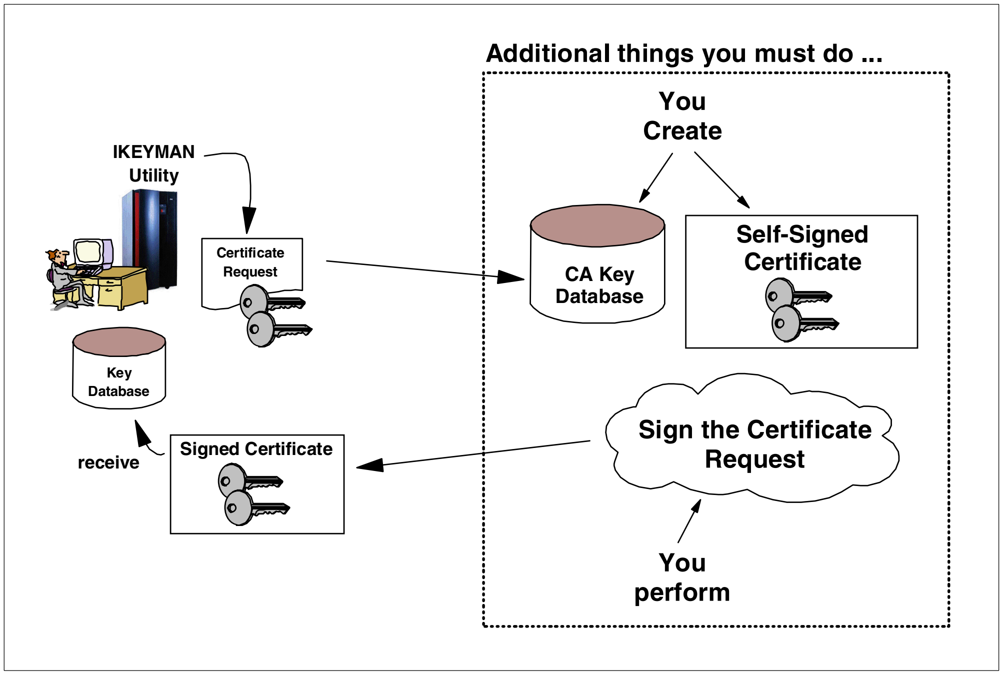


Figure 106. Activities when you act as your own CA and use a self-signed certificate

The best approach is to follow the instructions carefully and create a self-signed certificate you can use to establish SSL for testing purposes. After that you can request a certificate from a real CA.

A.1.2 Establish environment to run IKEYMAN utility

Make sure the following environment variables are set for the OMVS session from which you will run IKEYMAN:

- NLSPATH updated to include /usr/lpp/internet/%L/%N
- PATH updated to include /usr/lpp/internet/bin
- LIBPATH updated to include /usr/lpp/internet/bin

  Any reference to /usr/bin should be at the *end* of this environment variable.

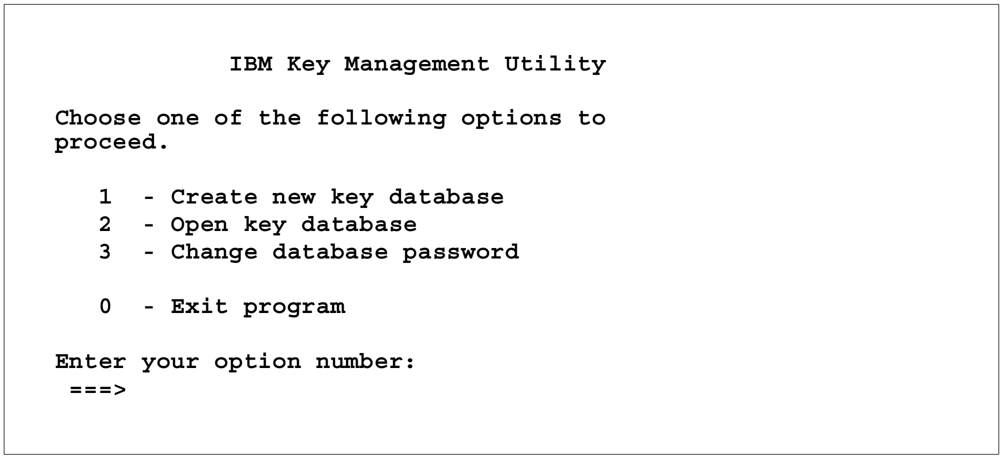
When you enter the OMVS shell, issue the `SET` command and then scroll through the output to make certain your updates are reflected in the environment variables displayed.

A.1.3 Start IKEYMAN utility

Do the following:

- Enter the OMVS shell
- Change directories to /usr/lpp/internet/bin
- Enter `IKEYMAN`

You should get a block of text on your screen (IKEYMAN is *not* a full screen panel application) that looks like Figure 107.



```
             IBM Key Management Utility

Choose one of the following options to
proceed.

   1  - Create new key database
   2  - Open key database
   3  - Change database password

   0  - Exit program

Enter your option number:
 ===>
```

Figure 107. IKEYMAN initial option panel

A.1.4 Create the Certificate Authority key database and key pair

Since you will be creating a “self-signed” certificate, you first need to create a key pair that represents the Certicate Authority. This key pair will *not* be used by your server during SSL session handshakes; it is being created just so we can “self-sign” a certificate and avoid the two or three week delay it normally takes to get a signed certificate from a *real* CA. Eventually you will need to get your key pair signed by a real CA. But don’t worry about that right now.

Do the following:

- Enter a 1 as your option on the initial option panel (shown in Figure 107).
- The utility will reply with:

  ```
  Enter key database name or press ENTER for "key.kdb".
  ```

  This is where you may name your own key ring file, or accept the default. It’s just a name, so use whatever you wish. We responded with “cakey.kdb”
- The utility will reply with:

  ```
  Enter password for the key database.......>
  ```

  Enter a password. Be sure to do this very carefully because you’ll be asked to enter it again:

  ```
  Enter password again for verification.....>
  ```
- After you enter the password for the second time, the utility will reply with:

  ```
  Should the password expire? (1 = yes, 0 = no) [1]:
  ```

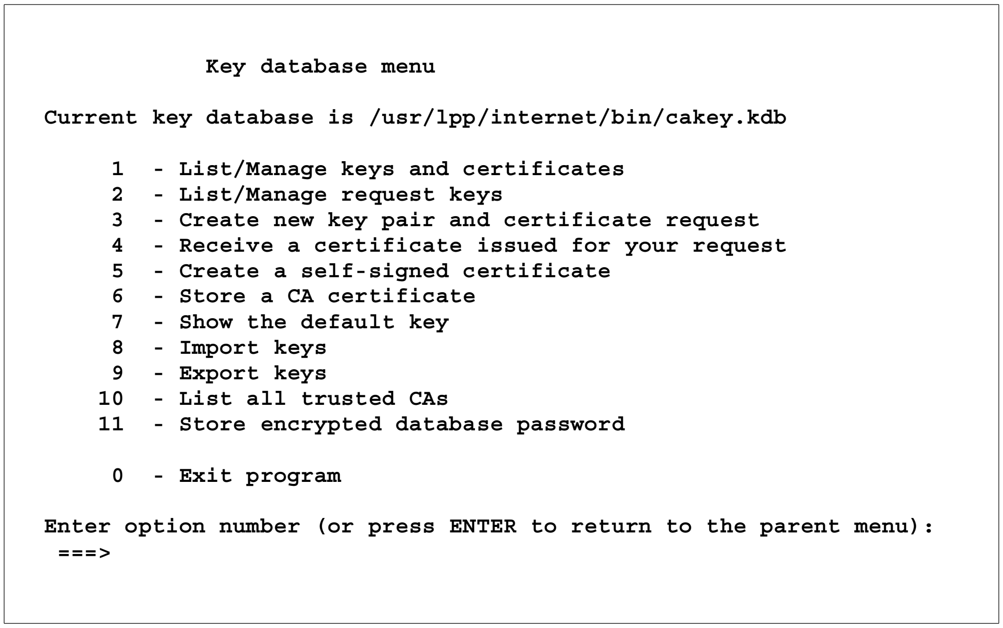
This is your choice. “Yes” (1) is the default. For a self-signed certificate you should enter 0 for “no.”

- The utility will reply with:

```
The database has been successfully created, do you want to continue to work
with the database now? (1 = yes, 0 = no) [1]:
```

The default is “Yes” (1) and you should choose that.

- The utility will reply with the next option panel, shown in Figure 108.



```
            Key database menu

Current key database is /usr/lpp/internet/bin/cakey.kdb

     1  - List/Manage keys and certificates
     2  - List/Manage request keys
     3  - Create new key pair and certificate request
     4  - Receive a certificate issued for your request
     5  - Create a self-signed certificate
     6  - Store a CA certificate
     7  - Show the default key
     8  - Import keys
     9  - Export keys
    10  - List all trusted CAs
    11  - Store encrypted database password

     0  - Exit program

Enter option number (or press ENTER to return to the parent menu):
 ===>
```

Figure 108. IKEYMAN key database menu

A.1.5 Create a self-signed certificate

We’ve created the key pair for the fictitious Certificate Authority we’ll use to sign our own certificate. Now we need to create a certificate and “sign” it. From the panel shown in Figure 108, do the following:

- Select option 5 (“Create a self-signed certificate”)
- The utility will reply with:

```
Enter version number of the certificate to be created (1, 2, or 3) [3]:
```

The default is “3”. Take the default.

- The utility will reply with:

```
Enter a label for this key................>
```

This may be any label of your choosing. For example, “Redbook CA”.

- The utility will reply with:

```
Select desired key size from the following options (512):
    1:    512
    2:    1024
Enter the number corresponding to the key size you want:
```

This defines the number of bits used in the key. The default for this is “1” for a key size of 512 bits. Again, take the default for your self-signed certificate.

- The utility will reply with:

  ```
  Enter certificate subject name fields in the following.
      Common Name (required)...............>
  ```

  This is where you will begin to supply information about the certificate authority. Since this is a fictitious CA, you are free to enter most any information you wish. For this question, we entered “Redbook CA”

- The utility will reply with:

  ```
  Organization (required)..............>
  ```

  This is a required field, but you may enter any organization name you wish. We entered “IBM”.

- The utility will reply with:

  ```
  Organization Unit (optional).........>
  ```

  This is optional, but we entered “ITSO”

- The utility will reply with:

  ```
  City/Locality (optional).............>
  ```

  We provided “Poughkeepsie”

- The utility will reply with:

  ```
  State/Province (optional)............>
  ```

  We provided “NY”

- The utility will reply with:

  ```
  Country Name (required 2 characters)..>
  ```

  Now this you have to be careful with. It is required, and for the United States the response is “us” (in *lower case*).

- The utility will reply with:

  ```
  Enter number of valid days for the certificate [365]:
  ```

  Taking the default means this certificate will be good for one year. For the purposes of testing the SSL capabilities of your system, this will suffice. Take the default.

- The utility will reply with:

  ```
  Do you want to set the key as the default in your
  key database? (1 = yes, 0 = no) [1]:
  ```

  This is asking you if you want to make this key the default key in the key database you created back in A.1.4, “Create the Certificate Authority key database and key pair” on page 185. Since it will be the only key in the database, answer “1” for “Yes.”

- The utility will reply with:

  ```
  Do you want to save the certificate to a file? (1 = yes, 0 = no) [1]:
  ```

  For a self-signed certificate you do, so answer “1” for “Yes.”

- The utility will reply with:

  ```
  Should the certificate binary data or Base64 encoded ASCII data be saved? (1
  = ASCII, 2 = binary) [1]:
  ```

  It’ll work either way. We provided “1” for “ASCII”.

- The utility will reply with:

  ```
  Enter certificate file name or press ENTER for "cert.arm":
  ```

  We responded with “cakey.arm.”
- The utility will reply with:

  ```
  Please wait while self-signed certificate is created...

  Your request has completed successfully,
  exit ikeyman? (1 = yes, 0 = no) [0]:
  ```

  Answer “1” for “Yes.” This will take you back to the OMVS prompt.

A.1.6 Create an operational key database

This will be the key database used by the Webserver for SSL. Do the following:

- Re-enter IKEYMAN
- Select option 1 to create a new key database. The utility will reply with:

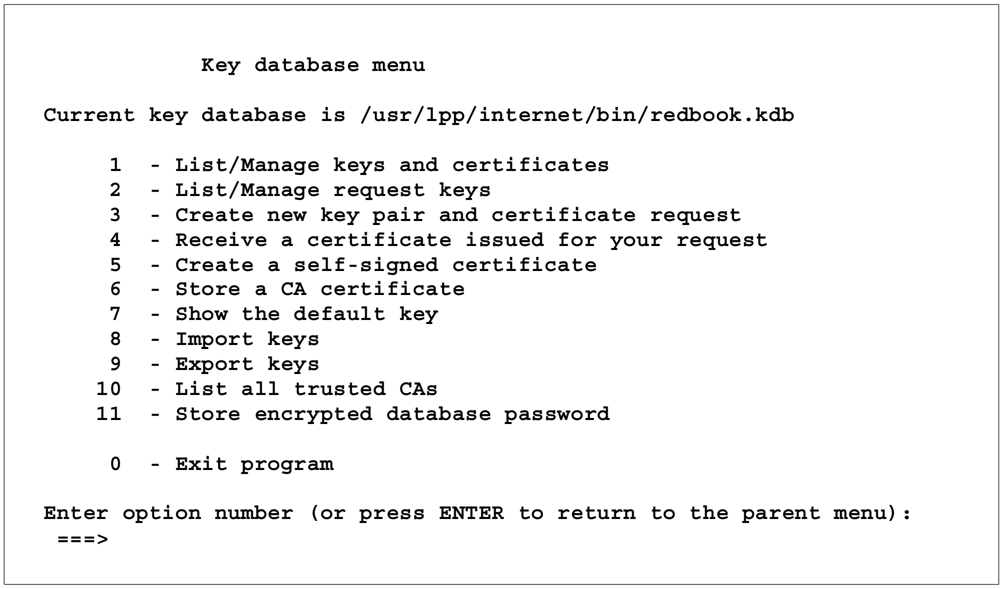
  ```
  Enter key database name or press ENTER for "key.kdb":
  ```

  For this redbook, we replied with “redbook.kdb”.
- The next several steps will be identical to what we showed you in A.1.4, “Create the Certificate Authority key database and key pair” on page 185. So we’ll be a bit more brief here.

  ```
  Enter password for the key database.......> <provide password>
  Enter password again for verification.....> <enter again>
  Should the password expire? (1 = yes, 0 = no) [1]: <0 for no>

  The database has been successfully created, do you want to continue to work
  with the database now? (1 = yes, 0 = no) [1]: <1 for yes>
  ```

A.1.7 Create a new key pair and certificate request

We’re still working with our “redbook.kdb” key database, so now we’re going to create a certificate request. The menu options should look like what’s shown in Figure 109 on page 189.



```
            Key database menu

Current key database is /usr/lpp/internet/bin/redbook.kdb

     1  - List/Manage keys and certificates
     2  - List/Manage request keys
     3  - Create new key pair and certificate request
     4  - Receive a certificate issued for your request
     5  - Create a self-signed certificate
     6  - Store a CA certificate
     7  - Show the default key
     8  - Import keys
     9  - Export keys
    10  - List all trusted CAs
    11  - Store encrypted database password

     0  - Exit program

Enter option number (or press ENTER to return to the parent menu):
 ===>
```

Figure 109. IKEYMAN options at point just before creating certificate request

Do the following:

- Select option 3, “Create new key pair and certificate request.” The utility will reply with:

  ```
  Enter certificate request file name or press ENTER for "certreq.arm":
  ```

  This is simply the name of an OMVS file that will contain the request. If you were creating a request for a real Certificate Authority (CA), you’d e-mail this request. But since we’re going to “self-sign” it, we’ll just point to it in a step that’s coming up.

  For this redbook, we responded with “redbook.arm”.

- The utility will respond with:

  ```
  Enter a label for this key................>
  ```

  This should be the host name of your system. For example, “www.yourhost.com.”

- From here, the questions asked by IKEYMAN get familiar:

  ```
  Select desired key size from the following options (512): 1
      1:    512
      2:    1024
  Enter the number corresponding to the key size you want:
  Enter certificate subject name fields in the following.
      Common Name (required)................> Redbook CA
      Organization (required)...............> IBM
      Organization Unit (optional)..........> ITSO
      City/Locality (optional)..............> Poughkeepsie
      State/Province (optional).............> NY
      Country Name (required 2 characters)..> US
  ```

  Note: US needs to be in uppercase for this, which is different from the certificate subject fields we created for our fictitious CA we created in A.1.5, “Create a self-signed certificate” on page 186.

  ```
  Please wait while key pair is created...
  ```

```
Your request has completed successfully,
exit ikeyman? (1 = yes, 0 = no) [0]:
```

We want to keep working with this key database, so answer “0” for “no.”

- IKEYMAN will come back with the same main option panel as shown in Figure 109 on page 189. We selected option 11 to “Store encrypted database password.”

  Note: This is a critical and required step when using OS/390.

  The utility will reply with the name of the “stash” file:

  ```
  The encrypted password has been stored in file
  /usr/lpp/internet/bin/redbook.sth
  ```

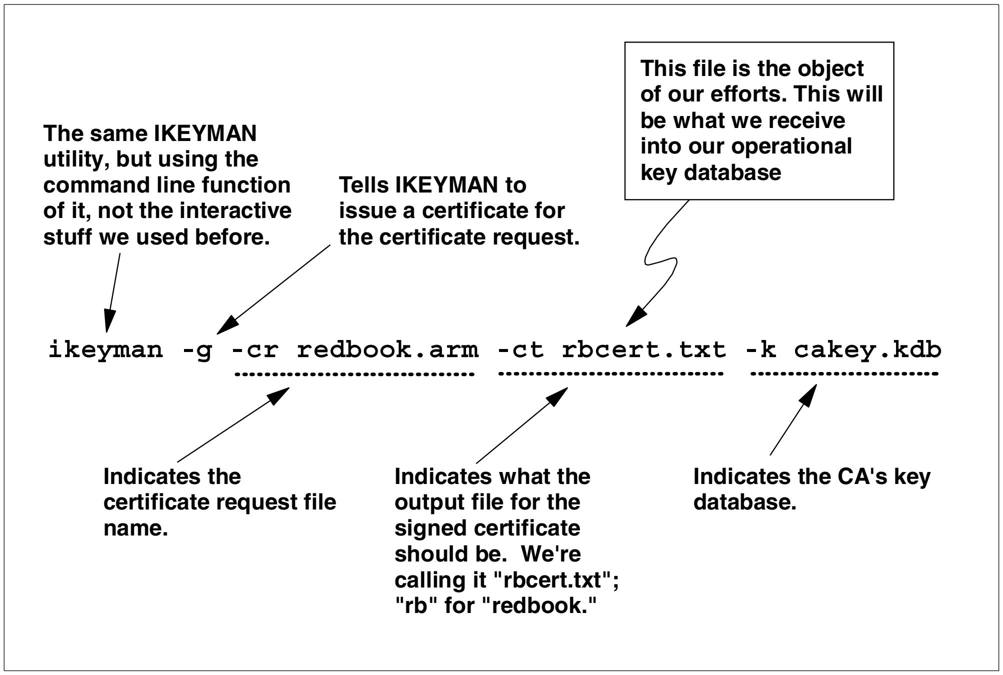
- The utility will then say:

  ```
  Your request has completed successfully,
  exit ikeyman? (1 = yes, 0 = no) [0]:
  ```

  Respond with “1” for “Yes.” We need to issue some OMVS commands and we need to exit IKEYMAN to get to the OMVS prompt.

A.1.8 Sign the operational certificate

We’ve created a Certificate Authority key database (cakey.kdb) and a self-signed certificate (cakey.arm). Now we’ll use that to sign the operation certificate request that we just created (redbook.arm). This is done with an OMVS command as illustrated in Figure 110.



This file is the object of our efforts. This will be what we receive into our operational key database

The same IKEYMAN utility, but using the command line function of it, not the interactive stuff we used before.

Tells IKEYMAN to issue a certificate for the certificate request.

```
ikeyman -g -cr redbook.arm -ct rbcert.txt -k cakey.kdb
```

Indicates the certificate request file name.

Indicates what the output file for the signed certificate should be. We're calling it "rbcert.txt"; "rb" for "redbook."

Indicates the CA's key database.

Figure 110. IKEYMAN command to sign the operational certiticate

Do the following:

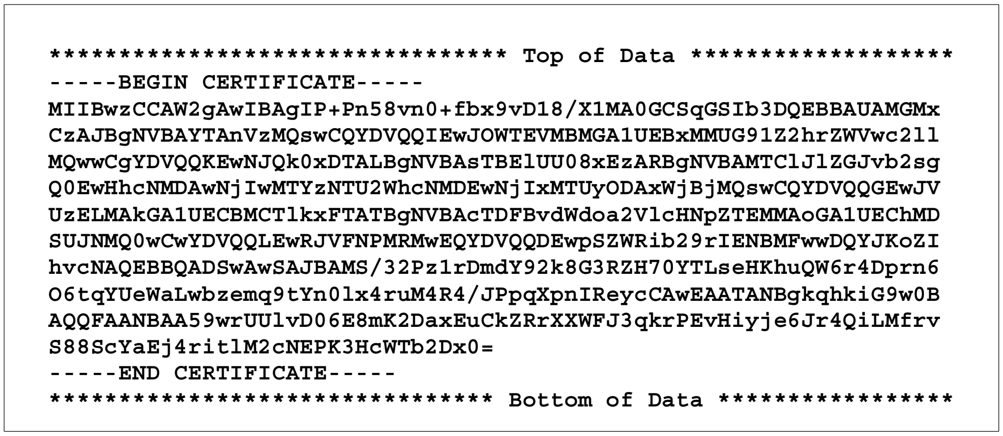
- From the OMVS command line, issue the command shown in Figure 110 (changing file names as appropriate to fit your situation).

- The utility will reply with:

  ```
  Enter password for the key database.......>
  ```

  This refers to the password for the Certificate Authority’s database, which in this example is cakey.kdb. This is the password you entered back in A.1.4, “Create the Certificate Authority key database and key pair” on page 185.
- The utilty will reply with:

  ```
  Please wait while a certificate is generated for the request......
  Your request has completed successfully!
  ```
- Verify that the output file (`rbcert.txt` in this example) exists. If you do an `obrowse` on that file, the contents should look something like what is shown in Figure 111.



```
********************************* Top of Data *******************
-----BEGIN CERTIFICATE-----
MIIBwzCCAW2gAwIBAgIP+Pn58vn0+fbx9vD18/X1MA0GCSqGSIb3DQEBBAUAMGMx
CzAJBgNVBAYTAnVzMQswCQYDVQQIEwJOWTEVMBMGA1UEBxMMUG91Z2hrZWVwc2ll
MQwwCgYDVQQKEwNJQk0xDTALBgNVBAsTBElUU08xEzARBgNVBAMTClJlZGJvb2sg
Q0EwHhcNMDAwNjIwMTYzNTU2WhcNMDEwNjIxMTUyODAxWjBjMQswCQYDVQQGEwJV
UzELMAkGA1UECBMCTlkxFTATBgNVBAcTDFBvdWdoa2VlcHNpZTEMMAoGA1UEChMD
SUJNMQ0wCwYDVQQLEwRJVFNPMRMwEQYDVQQDEwpSZWRib29rIENBMFwwDQYJKoZI
hvcNAQEBBQADSwAwSAJBAMS/32Pz1rDmdY92k8G3RZH70YTLseHKhuQW6r4Dprn6
O6tqYUeWaLwbzemq9tYn0lx4ruM4R4/JPpqXpnIReycCAwEAATANBgkqhkiG9w0B
AQQFAANBAA59wrUUlvD06E8mK2DaxEuCkZRrXXWFJ3qkrPEvHiyje6Jr4QiLMfrv
S88ScYaEj4ritlM2cNEPK3HcWTb2Dx0=
-----END CERTIFICATE-----
******************************** Bottom of Data *****************
```

Figure 111. Contents of signed certificate request (rbcert.txt)

You do not need to worry about what the contents of this file represent. We’re showing it here so you can visually verify that your signed certificate looks okay.

A.1.9 Import and Receive certificate into operational key database

With the signed certificate in hand, you must now get it into your operational key database. To do that, do the following:

- Get back into IKEYMAN.
- Select option 2 to “Open a key database”.
- Provide the name of your operational key database. In the example we’re showing you here, that would be “redbook.kdb.”
- Selection option 8, “Import keys.”
- The utility will respond with the “Import key menu:”

  ```
  Current key database is /usr/lpp/internet/bin/redbook.kdb

      1  - Import keys from another key database
      2  - Import keys from a PKCS12 file

      0  - Exit program

  Enter option number (or press ENTER to return to the parent menu):
  ```

Select “1” to import keys from another key database. You’re going to be importing the keys from the Certificate Authority key database you created in A.1.4, “Create the Certificate Authority key database and key pair” on page 185. In this example that would be the file “cakey.kdb”.

- The utility will respond with:

  ```
  Enter the key database name to be imported:
  ```

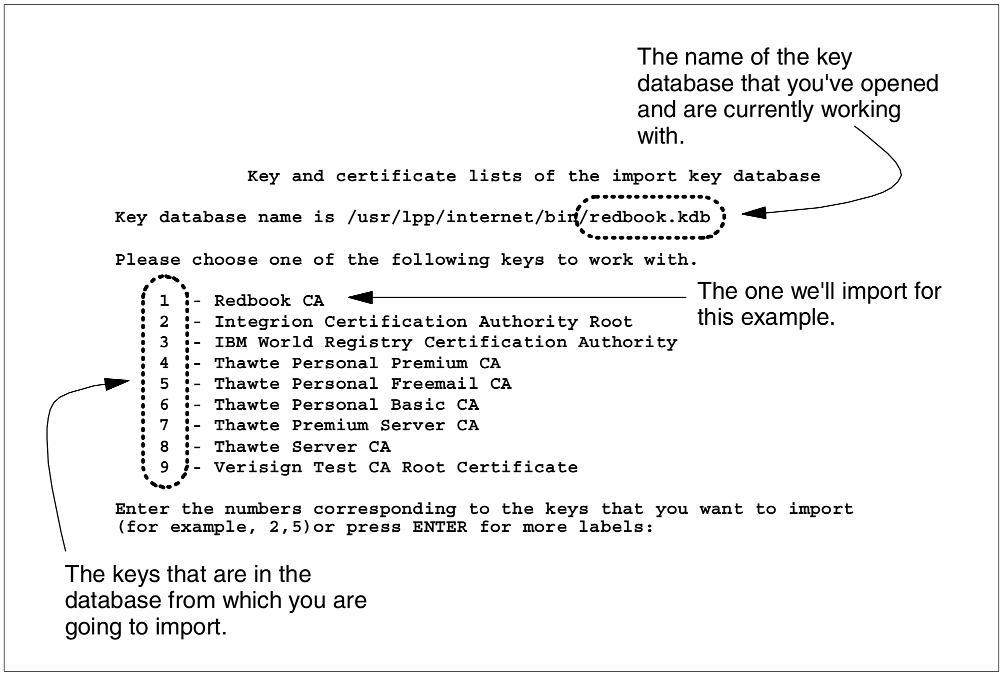
  Provide the name of your CA key database file. Again, in this example it would be “cakey.kdb”.

- The utility will respond with:

  ```
  Enter password for the key database to be imported:
  ```

  Provide the appropriate password you created in A.1.4, “Create the Certificate Authority key database and key pair” on page 185.

- The utility will reply with the screen shown in Figure 112.



```
                Key and certificate lists of the import key database

Key database name is /usr/lpp/internet/bin/redbook.kdb

Please choose one of the following keys to work with.

    1  - Redbook CA
    2  - Integrion Certification Authority Root
    3  - IBM World Registry Certification Authority
    4  - Thawte Personal Premium CA
    5  - Thawte Personal Freemail CA
    6  - Thawte Personal Basic CA
    7  - Thawte Premium Server CA
    8  - Thawte Server CA
    9  - Verisign Test CA Root Certificate

Enter the numbers corresponding to the keys that you want to import
(for example, 2,5)or press ENTER for more labels:
```

The name of the key database that you've opened and are currently working with.

The one we'll import for this example.

The keys that are in the database from which you are going to import.

Figure 112. The menu IKEYMAN displays for importing keys

- Select the option number that represents the key you wish to import. In this example it would be option 1 for “Redbook CA”.

- The utility will reply with:

  ```
  Your request has completed successfully, exit ikeyman?
  (1 = yes, 0 = no) [0]:
  ```

  Answer “0” for “No.”

- You will return to the “Key database menu” as illustrated in Figure 108 on page 186. Select option “4” to “Receive a certificate issued for your request.”

- The utility will reply with:

  ```
  Enter certificate file name or press ENTER for "cert.arm":
  ```

In our example, the response would be “rbcert.txt”. This is the file that was created by the IKEYMAN command used to sign the operational certificate as illustrated in Figure 110 on page 190.

- The utility will reply with:

```
Do you want to set the key as the default in your key database?
(1 = yes, 0 = no) [1]:
```

Answer “1” for “Yes.”

- The key file has been created, and it is paired with the stash file that contains the key database password. The last task is to copy the files to the Webserver’s “server_root,” which is typically:

```
/usr/lpp/internet/server_root
```

A.1.10 Update httpd.conf and restart the Webserver

Now that you have a new key database and self-signed certificate, do the following:

- Update the `httpd.conf` file with the directives shown in Figure 113.

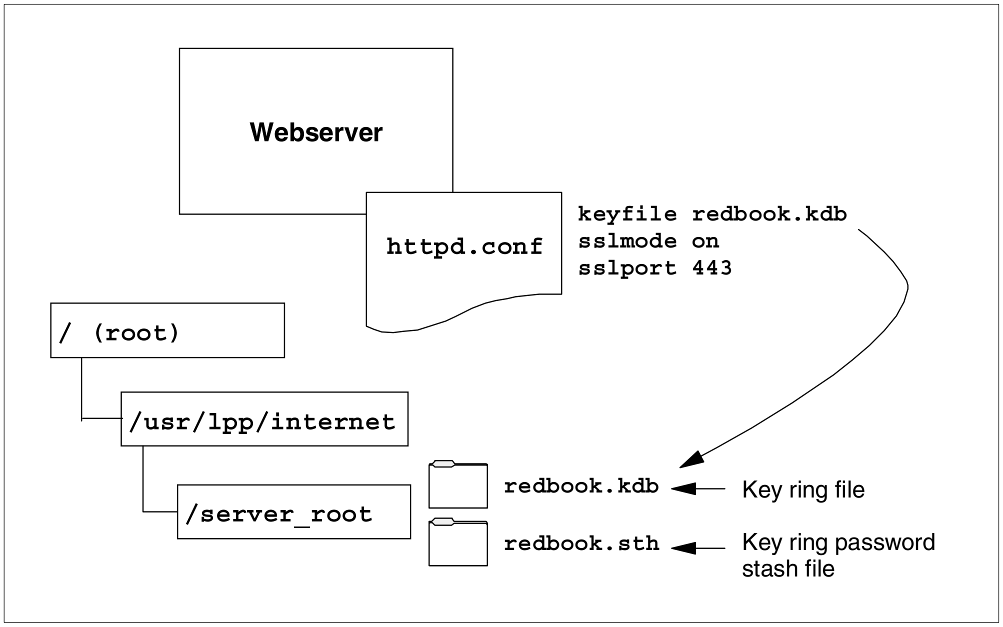
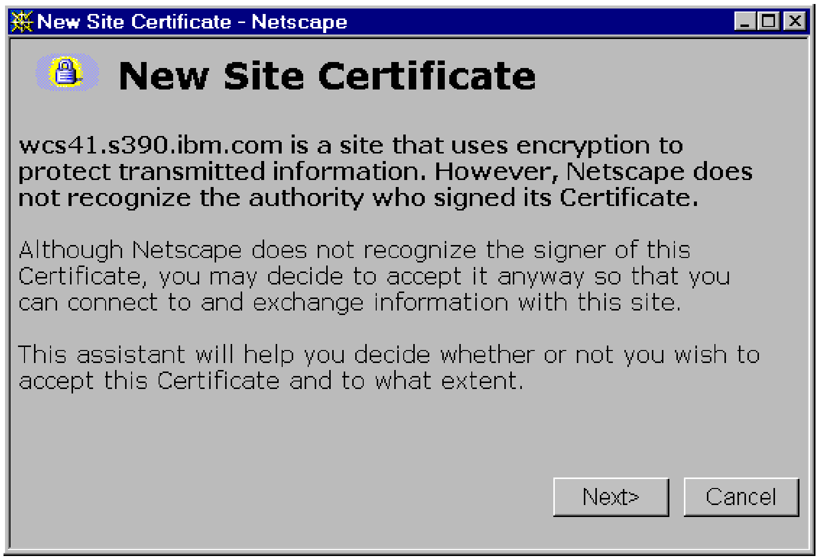


Figure 113. Webserver’s key ring directives and files

Note: The keyfile directive accepts a fully-qualified path definition for the key ring “kdb” file. If you’re at all uncertain about where the Webserver’s “server root” is, then fully-qualify the location of the key file and stash file, such as `/usr/lpp/internet/server_root/redbook.kdb`.

- Restart the Webserver to pick up the changes to the `httpd.conf` file.
- Verify that SSL works by accessing the site with a browser. Because you’re using a “self-signed” certificate, your browser will present a panel asking you to inspect the certificate the server has issued and that the browser does not recognize. An example of the first panel of this inspection process for Netscape is shown in Figure 114 on page 194.



New Site Certificate

wcs41.s390.ibm.com is a site that uses encryption to protect transmitted information. However, Netscape does not recognize the authority who signed its Certificate.

Although Netscape does not recognize the signer of this Certificate, you may decide to accept it anyway so that you can connect to and exchange information with this site.

This assistant will help you decide whether or not you wish to accept this Certificate and to what extent.

Figure 114. Netscape certificate inspection panel

A.2 Problems running CMNCONF

A.2.1 ncommerce.envvars not copied down to /etc

CEE3501S The module nc3_common.so was not found.

A.2.2 “Creating Shared Object Files” step not done

CEE3501S The module nc3_common.so was not found.

A.3 Problems starting the WCSuite server

Following are some errors and their causes that may result when you attempt to start the WebSphere Commerce Suite started task.

A.3.1 db2_load_caf failed return_code 8, reason_code f30002

This error symptom, seen in the SYSPRINT of the started task of WCSuite, is an indication that the CAF connection to DB2 failed. This may be caused by either of two situations:

- The DB2 subsystem has not been started.
- You misspelled the name of your DB2 subsystem when configuring WCSuite. You can check this by looking in both the `ncommerce.conf` file and the `<instance_name>.conf` file and searching for the `MS_DBSSID` directive. The value that follows that directive is the subsystem name WCSuite will attempt to use.

A.3.2 CMN0548E RC -1019 in SYSPRINT of started task

This happens when you try to start WCSuite when the database has not yet been created. The error message looks like this:

```
ERR:CMN0548E Reporting error.  Internal Return code is '-1019'.
DB2 error message follows:
DSNT408I SQLCODE = -204, ERROR: CMNSRV.POOLS IS AN UNDEFINED NAME
DSNT418I SQLSTATE = 42704 SQLSTATE RETURN CODE
DSNT415I SQLERRP = DSNXOTL SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD = -500 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD = X'FFFFFE0C' X'00000000' X'00000000' X'FFFFFFFF' X'00000000’
SQL DIAGNOSTIC INFORMATION
```

To fix this, perform the steps to create and load the database as provided in 2.4.12.6, “CMNCONF Database Create option” on page 49 and 2.4.12.7, “CMNCONF Database Load option” on page 51.

A.3.3 IEF450I-ABEND=S000 U4093 REASON=00000090 starting WCSuite

This message comes out on the system console, and most likely has one of two causes:

- The RACF userid under which the WCSuite server runs hasn’t been created. For information on how to create the group and userid, see 2.4.3, “Create group ID and user ID under which WCSuite will run” on page 33.
- The JCL start procedure has been copied to the proclib, but you have not created a started class profile. See 2.4.15, “Create a STARTED class profile” on page 54 for more information on the commands to do this.

A.4 Problems accessing the Metropolitan Demomall

This section focuses on problems seen when trying to access the basic functions of the WCSuite Demomall.

A.4.1 Server Error 404 accessing Demomall’s basemall.htm page

An “Error 404” indicates the Webserver was unable to locate the page requested. If your request was for the Demomall’s home page of `basemall.htm`, then this error is most likely the result of the `Pass` directives added to the Webserver’s `httpd.conf` file not taking effect.

Check the following:

- Browse the `httpd.conf` file. Is the WCSuite block of directives located at the top of the file? If not, then the CMNCONF utility wrote them to a different file. Rerun CMNCONF and make sure you point to the appropriate configuration file for the Webserver.
- If the WCSuite directives are located at the top of the `httpd.conf` file, then restart the Webserver so the updates to the file can be picked up by the Webserver.

A.4.2 Server Error 500 accessing mall_dir.d2w macro

An Error 500 might be issued by the Webserver for a couple of reasons. Here’s two that we found:

- If the error message says, “Service handler performed no action; contact the server administrator,“ then did you misspell the word “commerce” or the word “command” in the URL used to invoke this macro? If so, you will get an Error 500 page because the Webserver was unable to locate the service being requested.
- If the error message says, “IMW0241E Access denied - surrogate user setup error” then it means you probably forgot to provide program control for the WCSuite load data set (see 2.4.2, “Enable program control on datasets” on page 33). Further validation of this as the problem can be found by searching the Webserver’s SYSOUT when the “-vv” trace is enabled and seeing if the following message is present:

  ```
  Failed access as Surrogate: PUBLIC, Errno: 139, Errno2: 0be802af
  ```

A.4.3 CMN0950E- Cmnd Execution Failure accessing mall_dir.d2w macro

A “Command Execution Failure” implies a general problem trying to execute the command request. If you were following the checklist provided in 2.4.20, “Checklist to validate basic WCSuite functionality” on page 58 when you encountered this, then the command you were attempting to execute was `ExecMacro`.

Check the following:

- Is the WebSphere Commerce Suite server up and running? If the WCSuite server is not up but the Webserver is, then you’ll get the Demomall’s home page okay, but fail when you try to execute the command to display the “Mall Directory.”
- Did you spell the command exactly as indicated in 2.4.20, “Checklist to validate basic WCSuite functionality” on page 58? Even something as seemingly inconsequential as having the upper-case “M” in ExecMacro set as a lower-case “m” will cause the “Command Execution Failure” message. Make sure the spelling is correct.
- Is it possible that you created the database but failed to load it with data? If so, look in the SYSPRINT of the WCSuite started task. If you see the following error condition, then you must return to the step shown in 2.4.12.7, “CMNCONF Database Load option” on page 51.

  ```
  CMN0411I THE NET.COMMERCE SERVER IS READY FOR CONNECTIONS
  ERR:CMN0548E Reporting error.  Internal Return code is '-1013'.
    DB2 error message follows:
  DSNT404I SQLCODE = 100, NOT FOUND: ROW NOT FOUND FOR FETCH, UPDATE,
    OR DELETE, OR THE RESULT OF A QUERY IS AN EMPTY TABLE
  DSNT418I SQLSTATE = 02000 SQLSTATE RETURN CODE
  DSNT415I SQLERRP = DSNXRFF SQL PROCEDURE DETECTING ERROR
  DSNT416I SQLERRD = -110 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
  DSNT416I SQLERRD = X'FFFFFF92' X'00000000' X'00000000' X'FFFFFFFF'
     X'00000000' X'00000000'
  SQL DIAGNOSTIC INFORMATION
  ERROR  CMN0102E: The KeyManager could not initialize properly.
  ERROR  CMN0001E: The database returned the error code -1013.
  ERROR  CMN0101E: The Command Manager could not initialize properly.
  ```

A.4.4 DTWP001E: Net.Data is unable to locate the macro file message

This is a very sparse-looking browser message and indictates a problem on the part of Net.Data to find the macro named on the command. The good news is that Net.Data has been invoked, which means almost everything is working properly.

If you encountered this problem while running through 2.4.20, “Checklist to validate basic WCSuite functionality” on page 58, then this is most likely caused by your misspelling the name of the macro on the URL as you typed it.

Note: A similar message is “DTWP029E: Net.Data is unable to locate the HTML block,” which is an indication that the very last portion of the command URL (which names the Net.Data report block that serves as the entry point for this running of the macro) is mispelled.

Appendix B. REXX sample programs

In this appendix we provide you with copies of the REXX sample programs we used during the migration described previously. See Chapter 4, “Migrating the database” on page 109 for a detailed description of the necessary migration steps and how and when to use the REXX sample programs.

B.1 DB2 check data utility job

This REXX program generates a single DB2 CHECK DATA UTILITY JOB for all TABLESPACES given as input.

```
/*REXX*****************************************************************/
/*                                                                    */
/* LICENSED MATERIALS - PROPERTY OF IBM                               */
/* 5697-D32                                                           */
/*                                                                    */
/* (C) COPYRIGHT IBM CORP. 2000       ALL RIGHTS RESERVED.            */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                       */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                    */
/* SCHEDULE CONTRACT WITH IBM CORP.                                   */
/*                                                                    */
/**********************************************************************/
/*  THIS REXX GENERATES A SINGLE DB2 CHECK DATA UTILITY JOB FOR ALL   */
/*  TABLESPACES GIVEN AS INPUT                                        */
/*                                                                    */
/*  INPUTS:                                                           */
/*    1) A FILE CONTAINING TABLESPACE NAMES TO BE INCLUDED IN THE     */
/*       CHECK DATA INVOCATION.                                       */
/*                                                                    */
/*  OUTPUTS:                                                          */
/*    1) A PDS MEMBER TO CONTAIN THE GENERATED CHECK DATA JOB JCL     */
/*                                                                    */
/*  VARIABLES TO CONSIDER:                                            */
/*    1) INPUT DATASET = 'SYSADM.MIGRATE.MIGSTBSP.OUT'                */
/*    2) OUTPUT DATASET = 'SYSADM.JOBS(CHECK)'                        */
/*    3) JCL ATTRIBUTES OF THE GENERATED CHECK DATA INVOCATION        */
/*                                                                    */
/**********************************************************************/
"ALLOC DD(IN1) DS('MIG41.MIGSTBSP.EDIT.OUT') SHR REUSE"
"EXECIO * DISKR IN1 (STEM  LISTV."
IF RC^=0 THEN EXIT
"ALLOC DD(OUT) DS('MIG41.JOBS(CHECK41)') SHR REUSE"
 IF RC^=0 THEN EXIT
QUEUE "//CHECK   JOB CLASS=A,MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID "
  QUEUE "/*JOBPARM SYSAFF=SC54                                         "
  QUEUE "//CHECKTS EXEC DSNUPROC,SYSTEM=DB2P,UID='CHECKTS'             "
  QUEUE "//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SORTWK02  DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SORTWK03  DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SORTWK04  DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)              "
  QUEUE "//SYSERR    DD DSN=MIGR.CHECK.SYSERR,UNIT=SYSDA,              "
  QUEUE "//  DISP=(MOD,DELETE,CATLG),SPACE=(CYL,(5,5),RLSE)            "
  QUEUE "//SYSPRINT DD   SYSOUT=*                                      "
  QUEUE "CHECK DATA                                                    "
  DO I=1 TO LISTV.0
   PARSE VALUE LISTV.I WITH TNAME REST
   QUEUE "     TABLESPACE "TNAME
  END
  QUEUE "  SCOPE ALL                                                   "
  QUEUE "//                                                            "
"EXECIO " QUEUED() " DISKW OUT"
"EXECIO 0 DISKW OUT (FINIS"
"FREE DD(OUT)"
"EXECIO 0 DISKW IN1 (FINIS"
"FREE DD(IN1)"
EXIT
```

B.2 REXX program to generate a single REORG job

This REXX program generates a single job to use with the DB2 REOG utility.

```
/*REXX*****************************************************************/
/*                                                                    */
/* LICENSED MATERIALS - PROPERTY OF IBM                               */
/* 5697-D32                                                           */
/*                                                                    */
/* (C) COPYRIGHT IBM CORP. 2000       ALL RIGHTS RESERVED.            */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                       */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                    */
/* SCHEDULE CONTRACT WITH IBM CORP.                                   */
/*                                                                    */
/**********************************************************************/
/*  THIS REXX GENERATES A SINGLE JOB TO USE THE DB2 REORG UTILITY ON  */
/*  TABLESPACES GIVEN AS INPUT. EACH TABLESPACE IS REORGED IN A       */
/*  SEPERATE JCL EXEC STEP. THE GENERATED DB2 REORG SYNTAX INCLUDES   */
/*  INLINE IMAGE COPY AND STATISTICS COLLECTION.                      */
/*                                                                    */
/*  INPUTS:                                                           */
/*    * A FILE CONTAINING TABLESPACE NAMES TO BE REORGED              */
/*                                                                    */
/*  OUTPUTS:                                                          */
/*    * A PDS MEMBER TO CONTAIN THE GENERATED REORG JOB JCL           */
/*                                                                    */
/*  VARIABLES TO CONSIDER:                                            */
/*    * INPUT DATASET = 'SYSADM.MIGRATE.MIGSTBSP.OUT'                 */
/*    * OUTPUT DATASET = 'SYSADM.JOBS(REORG)'                         */
/*    * JCL ATTRIBUTES OF THE GENERATED REORG INVOCATION              */
/*                                                                    */
/*  HINT:                                                             */
/*    FOR BETTER PERFORMANCE, PARALLELISM CAN BE ACHIEVED BY THE      */
/*    ADDITION OF MULTIPLE JCL JOB STATEMENTS TO THE GENERATED JOB.   */
/*    ADDITION OF JOB STATEMENTS WILL ALSO AVOID RECEIVING JES ERRORS */
/*    CONCERNING THE MAXIMUM NUMBER OF JCL EXEC STATEMENTS WHEN A     */
/*    LARGE NUMBER OF TABLESPACES ARE PROVIDED AS INPUT.              */
/**********************************************************************/
"ALLOC DD(IN1) DS('MIG41.MIGSTBSP.EDIT.OUT') SHR REUSE"
"EXECIO * DISKR IN1 (STEM  LISTV."
IF RC^=0 THEN EXIT
"ALLOC DD(OUT)  DS('MIG41.JOBS(CPYOFF41)') SHR REUSE"
```

```
 IF RC^=0 THEN EXIT
 QUEUE "//REPAIR  JOB CLASS=A,MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID  "
 QUEUE "/*JOBPARM SYSAFF=SC54                                          "
 QUEUE "//PENDOFF  EXEC DSNUPROC,SYSTEM=DB2P,UID=REPAIR                "
 QUEUE "//* TURN OFF COPY PENDING                                      "
 QUEUE "//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)               "
 QUEUE "//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(0,1),RLSE)               "
 QUEUE "//SYSPRINT DD   SYSOUT=*                                       "
 QUEUE "//SYSIN    DD   *                                              "
DO I=0 TO (LISTV.0-1)
  J=I+1
  PARSE VALUE LISTV.J WITH TNAME REST
  K=RIGHT(I,3,0)
     QUEUE " REPAIR SET TABLESPACE "TNAME " NOCOPYPEND                 "
     QUEUE " REPAIR SET INDEX (ALL) TABLESPACE "TNAME " NOCOPYPEND     "
  "EXECIO " QUEUED() " DISKW OUT"
END
"EXECIO 0 DISKW OUT (FINIS"
"FREE DD(OUT)"
"EXECIO 0 DISKW IN1 (FINIS"
"FREE DD(IN1)"
EXIT
```

B.3 REXX program to generate DB2 LOAD utility job

This REXX program generates one or more DB2 LOAD utility job(s) for all TABLESPACES given as input.

```
/*REXX*****************************************************************/
/*                                                                    */
/* LICENSED MATERIALS - PROPERTY OF IBM                               */
/* 5697-D32                                                           */
/*                                                                    */
/* (C) COPYRIGHT IBM CORP. 2000       ALL RIGHTS RESERVED.            */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                       */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                    */
/* SCHEDULE CONTRACT WITH IBM CORP.                                   */
/*                                                                    */
/**********************************************************************/
/*  THIS REXX GENERATES ONE OR MORE DB2 LOAD UTILITY JOB(S) FOR ALL   */
/*  TABLESPACES GIVEN AS INPUT. FOR JOB CREATED, TWO EXEC             */
/*  STATEMENTS ARE OUTPUT:                                            */
/*    * AN IEFBR14 DELETE OF THE SYSERR DATASET TO BE CREATED         */
/*    * A DB2 LOAD UTILITY INVOCATION FOR UP TO 100 TABLESPACES       */
/*  THE GENERATED JOB IS MATCHED TO THAT GENERATED BY THE REXX        */
/*  UNLOAD SAMPLE. THE 100 TABLESPACE MAXIMUM PER JOB GENERATED WAS   */
/*  INPLEMENTED BECAUSE THIS LIMITATION IS FOUND IN THE DSNTIAUL      */
/*  DB2 SAMPLE USED TO UNLOAD TABLESPACES.                            */
/*  IF MORE THAN 100 TABLESPACES ARE PROVIDED AS INPUT, MULTIPLE      */
/*  LOAD JOBS ARE GENERATED, EACH LOADING UP TO 100 TABLESPACES.      */
/*                                                                    */
/*  INPUTS:                                                           */
/*    * A FILE CONTAINING TABLESPACE NAMES TO BE LOADED               */
/*                                                                    */
/*  OUTPUTS:                                                          */
/*    * A PDS MEMBER TO CONTAIN THE GENERATED LOAD JOB JCL            */
/*                                                                    */
/*  VARIABLES TO CONSIDER:                                            */
/*    * INPUT DATASET = 'SYSADM.MIGRATE.MIGSTMIG.EDIT.OUT'            */
/*    * OUTPUT DATASET = 'MIG41.JOBS(LOAD)'                           */
/*    * JCL ATTRIBUTES OF THE GENERATED LOAD INVOCATION               */
/*                                                                    */
/**********************************************************************/
"ALLOC DD(IN1) DS('MIG41.MIGSTMIG.EDIT.OUT') SHR REUSE"
"EXECIO * DISKR IN1 (STEM  LISTV."
IF RC^=0 THEN EXIT
"ALLOC DD(OUT) DS('MIG41.JOBS(LOAD41)') SHR REUSE"
 IF RC^=0 THEN EXIT
 X=LISTV.0/100
 DO Y = 0 TO X
  J = RIGHT((Y),1,0)
  QUEUE "//LOAD"J"   JOB CLASS=A,MSGLEVEL=(1,1),REGION=0M,             "
  QUEUE "//          NOTIFY=&SYSUID                                    "
  QUEUE "/*JOBPARM SYSAFF=SC54                                          "
  QUEUE "//DELETE  EXEC PGM=IEFBR14                                    "
  QUEUE "//SYSPRINT  DD SYSOUT=*                                       "
  QUEUE "//SYSUT1    DD DSN=MIG41.LOAD.SYSERR"J",UNIT=SYSDA,           "
  QUEUE "//          DISP=(MOD,DELETE,DELETE),SPACE=(TRK,(1,1))        "
  QUEUE "//SYSUT2    DD DUMMY                                          "
  QUEUE "//SYSIN     DD DUMMY                                          "
  QUEUE "//LOAD"J"   EXEC DSNUPROC,SYSTEM=DB2P,UID='LOAD"J"'           "
  QUEUE "//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SORTWK02  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SORTWK03  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SORTWK04  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(5,1))                   "
  QUEUE "//SYSERR    DD DSN=MIG41.LOAD.SYSERR"J",UNIT=SYSDA,           "
  QUEUE "//          DISP=(NEW,CATLG,KEEP),SPACE=(CYL,(1,1))           "
  QUEUE "//SYSPRINT  DD   SYSOUT=*                                     "
  DO I=(100*Y) TO MIN((LISTV.0-1),(100*Y)+99)
   K=RIGHT(I,3,0)
   L=RIGHT(I,2,0)
   QUEUE "//SYSREC"L"  DD DSN=MIG41.LRECS"K",DISP=SHR                  "
  END
  QUEUE "//SYSIN     DD   DSN=MIG41.LOAD"J",DISP=SHR                   "
 END
"EXECIO " QUEUED() " DISKW OUT"
"EXECIO 0 DISKW OUT (FINIS"
"FREE DD(OUT)"
"EXECIO 0 DISKW IN1 (FINIS"
"FREE DD(IN1)"
EXIT
```

B.4 REXX program sample to use with REORG utility

This sample REXX program generates a single job to use the DB2 reorg utility on TABLESAPCES given as input.

```
/*REXX*****************************************************************/
/*                                                                    */
/* LICENSED MATERIALS - PROPERTY OF IBM                               */
/* 5697-D32                                                           */
/*                                                                    */
/* (C) COPYRIGHT IBM CORP. 2000       ALL RIGHTS RESERVED.            */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                       */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                    */
/* SCHEDULE CONTRACT WITH IBM CORP.                                   */
/*                                                                    */
/**********************************************************************/
/*  THIS REXX GENERATES A SINGLE JOB TO USE THE DB2 REORG UTILITY ON  */
/*  TABLESPACES GIVEN AS INPUT. EACH TABLESPACE IS REORGED IN A       */
/*  SEPERATE JCL EXEC STEP. THE GENERATED DB2 REORG SYNTAX INCLUDES   */
/*  INLINE IMAGE COPY AND STATISTICS COLLECTION.                      */
/*                                                                    */
/*  INPUTS:                                                           */
/*    * A FILE CONTAINING TABLESPACE NAMES TO BE REORGED              */
/*                                                                    */
/*  OUTPUTS:                                                          */
/*    * A PDS MEMBER TO CONTAIN THE GENERATED REORG JOB JCL           */
/*                                                                    */
/*  VARIABLES TO CONSIDER:                                            */
/*    * INPUT DATASET = 'SYSADM.MIGRATE.MIGSTBSP.OUT'                 */
/*    * OUTPUT DATASET = 'SYSADM.JOBS(REORG)'                         */
/*    * JCL ATTRIBUTES OF THE GENERATED REORG INVOCATION              */
/*                                                                    */
/*  HINT:                                                             */
/*    FOR BETTER PERFORMANCE, PARALLELISM CAN BE ACHIEVED BY THE      */
/*    ADDITION OF MULTIPLE JCL JOB STATEMENTS TO THE GENERATED JOB.   */
/*    ADDITION OF JOB STATEMENTS WILL ALSO AVOID RECEIVING JES ERRORS */
/*    CONCERNING THE MAXIMUM NUMBER OF JCL EXEC STATEMENTS WHEN A     */
/*    LARGE NUMBER OF TABLESPACES ARE PROVIDED AS INPUT.              */
/**********************************************************************/
"ALLOC DD(IN1) DS('HAUSER.MIGSTBSP.OUT') SHR REUSE"
"EXECIO * DISKR IN1 (STEM  LISTV."
IF RC^=0 THEN EXIT
"ALLOC DD(OUT)  DS('MIG41.JOBS(REORG)') SHR REUSE"
 IF RC^=0 THEN EXIT
 QUEUE "//REORG   JOB CLASS=A,MSGLEVEL=(1,1),REGION=0M,                "
 QUEUE "//        NOTIFY=&SYSUID                                       "
 QUEUE "/*JOBPARM SYSAFF=SC54                                          "
DO I=0 TO (LISTV.0-1)
  J=I+1
  PARSE VALUE LISTV.J WITH TNAME REST
  K=RIGHT(I,3,0)
     QUEUE "//REORG"K" EXEC DSNUPROC,SYSTEM=DB2P,UID='REO"K"'          "
     QUEUE "//SORTWK01  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SORTWK02  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SORTWK03  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SORTWK04  DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(5,1))                "
     QUEUE "//SYSPRINT DD   SYSOUT=*                                   "
     QUEUE "//SYSCOPY DD DSN=MIG41.RCOPY"K",DISP=(NEW,CATLG,DELETE),   "
     QUEUE "//        UNIT=SYSDA,SPACE=(TRK,(5,1),RLSE)                "
     QUEUE "//SYSREC  DD DSN=MIG41.RRECS"K",DISP=(NEW,DELETE,KEEP),    "
     QUEUE "//        UNIT=SYSDA,SPACE=(TRK,(5,1),RLSE)                "
     QUEUE "//SYSIN    DD   *                                          "
     QUEUE " REORG TABLESPACE "TNAME " LOG NO                          "
     QUEUE "       SORTDATA SORTKEYS COPYDDN(SYSCOPY)                  "
     QUEUE "       STATISTICS TABLE ALL INDEX ALL                      "
  "EXECIO " QUEUED() " DISKW OUT"
END
"EXECIO 0 DISKW OUT (FINIS"
"FREE DD(OUT)"
"EXECIO 0 DISKW IN1 (FINIS"
"FREE DD(IN1)"
EXIT
```

B.5 REXX program to use with DB2 unload sample DSNTIAUL

This REXX program generates one or more jobs to invoke the DB2 unload sample DSNTIAUL.

```
/*REXX*****************************************************************/
/*                                                                    */
/* LICENSED MATERIALS - PROPERTY OF IBM                               */
/* 5697-D32                                                           */
/*                                                                    */
/* (C) COPYRIGHT IBM CORP. 2000       ALL RIGHTS RESERVED.            */
/* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                       */
/* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                    */
/* SCHEDULE CONTRACT WITH IBM CORP.                                   */
/*                                                                    */
/**********************************************************************/
/*  THIS REXX GENERATES ONE OR MORE JOBS TO INVOKE THE DB2 UNLOAD     */
/*  SAMPLE DSNTIAUL. DSNTIAUL HAS A LIMIT OF 100 TABLES MAXIMUM IN    */
/*  A SINGLE INVOCATION. THE SAME LIMITATION IS USED CORRESPONDINGLY  */
/*  BY THIS REXX IN GENERATING THE DSNTIAUL INVOCATIONS.              */
/*  FOR EACH GENERATED JOB, TWO JOB EXEC STATEMENTS ARE CREATED:      */
/*    * AN IEFBR14 DELETE OF THE UNLOAD DATASETS                      */
/*    * A DB2 DSNTIAUL UNLOAD INVOCATION FOR A GROUP OF TABLES        */
/*                                                                    */
/*  INPUTS:                                                           */
/*    * A FILE CONTAINING TABLE NAMES TO BE UNLOADED                  */
/*                                                                    */
/*  OUTPUTS:                                                          */
/*    * A PDS MEMBER TO CONTAIN THE GENERATED UNLOAD JOB JCL          */
/*                                                                    */
/*  VARIABLES TO CONSIDER:                                            */
/*    * INPUT DATASET = 'HAUSER.MIGSTMIG.EDIT.OUT'                    */
/*    * OUTPUT DATASET = 'MIG41.JOBS(UNLOAD31)'                       */
/*    * JCL ATTRIBUTES OF THE GENERATED COPY INVOCATION               */
/*                                                                    */
/**********************************************************************/
/*TRACE I   */
/*"NEWSTACK"*/
"ALLOC DD(IN1) DS('MIG41.MIGSTMIG.EDIT.OUT') SHR REUSE"
"EXECIO * DISKR IN1 (STEM  LISTV."
IF RC^=0 THEN EXIT
"ALLOC DD(OUT) DS('MIG41.JOBS(UNLOAD31)') SHR REUSE"
 IF RC^=0 THEN EXIT
 X=LISTV.0/100
 DO Y = 0 TO X
   J = RIGHT((Y),1,0)
   QUEUE "//UNLOAD"J" JOB CLASS=A,MSGLEVEL=(1,1),REGION=0M,          "
   QUEUE "//          NOTIFY=&SYSUID                                 "
   QUEUE "/*JOBPARM SYSAFF=SC54                                       "
   QUEUE "//*********************************************************"
   QUEUE "//DELETE  EXEC PGM=IEFBR14                                 "
   QUEUE "//SYSPRINT  DD SYSOUT=*                                    "
   QUEUE "//SYSUT1    DD DSN=MIG41.LOAD"J",DISP=(MOD,DELETE,DELETE), "
   QUEUE "//          UNIT=SYSDA,SPACE=(TRK,(1,1))                   "
  DO I=(100*Y) TO MIN((LISTV.0-1),(100*Y)+99)
    K=RIGHT((I),3,0)
   QUEUE "//          DD DSN=MIG41.LRECS"K",DISP=(MOD,DELETE,DELETE),"
   QUEUE "//          UNIT=SYSDA,SPACE=(CYL,(1,1))                   "
  END
   QUEUE "//UNLOAD  EXEC PGM=IKJEFT01,DYNAMNBR=20                    "
   QUEUE "//SYSTSPRT DD  SYSOUT=*                                    "
   QUEUE "//SYSTSIN  DD  *                                           "
   QUEUE " DSN SYSTEM(DB2P)                                          "
   QUEUE " RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB61)    -                "
   QUEUE "      LIB('DB2V610P.RUNLIB.LOAD')                          "
   QUEUE "//SYSPRINT DD SYSOUT=*                                     "
   QUEUE "//SYSUDUMP DD SYSOUT=*                                     "
   QUEUE "//SYSPUNCH DD DSN=MIG41.LOAD"J",                           "
   QUEUE "//         UNIT=3390,SPACE=(TRK,(1,1),RLSE),DISP=(,CATLG)  "
  DO I=(100*Y) TO MIN((LISTV.0-1),(100*Y)+99)
    K=RIGHT(I,3,0)
    L=RIGHT(I,2,0)
    QUEUE "//SYSREC"L" DD DSN=MIG41.LRECS"K",                        "
    QUEUE "//         UNIT=3390,SPACE=(CYL,(5,5),RLSE),DISP=(,CATLG) "
  END
   QUEUE "//SYSIN     DD *                                           "
  DO I=(100*Y)+1 TO MIN((LISTV.0),(100*(Y+1)))
    PARSE VALUE LISTV.I WITH TNAME  REST
    QUEUE TNAME
  END
   QUEUE "//                                                         "
 END

"EXECIO " QUEUED() " DISKW OUT"
/*"DELSTACK"*/
"EXECIO 0 DISKW OUT (FINIS"
"FREE DD(OUT)"
"EXECIO 0 DISKW IN1 (FINIS"
"FREE DD(IN1)"
EXIT
```

Appendix C. Special notices

This publication is intended to help system programmers and system administrators to install, customize and migrate to the new WebSphere Commerce Suite V4.1. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Commerec Suite V4.1. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Commerce Suite V4.1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)®
IBM ®
AS/400
AT
CICS
CT
Current
DB/2
DRDA
MQSeries
Net.Data
Netfinity
OS/390
OS/400
RACF
Redbooks
Redbooks Logo
RS/6000
S/390
SP
System/390
WebSphere
Wizard
XT
400
Lotus
Domino
eSuite
Notes

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix D. Related publications and resources

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 IBM Education for WebSphere Commerce Suite

The following courses are all geared toward the NT or AIX WebSphere Commerce Suite Pro version of the product. Because there is much in common between the platforms, these will serve you well if you are interested in the learning the function of the WCSuite product.

EB080	WebSphere Commerce Suite Overview Web Book (CD-ROM)
EB110	What's New in WebSphere Commerce Suite Version 4.1 Web Book (CD-ROM)
EB040	Starting an E-commerce Web site with WebSphere Commerce Suite (Web Book)
EB010	Starting an E-commerce Site with WebSphere Commerce Suite for Windows NT Workshop
EB020	Starting an E-commerce Site with WebSphere Commerce Suite for AIX Workshop
EB050	Adding Professional E-commerce Features with WebSphere Commerce Suite
EB150	Customizing an E-commerce Site with WebSphere Commerce Suite Workshop
N3395	Net.Commerce Hosting Server Implementation

D.2 IBM Websites related to WebSphere Commerce Suite V4.1

The WebSphere Commerce Suite V4.1 Pro Edition library containst literature about the product. The URL is:

```
http://www.ibm.com/software/webservers/commerce/wcs_pro/lit.html
```

A link off that page will take you to the “Technical Libraries”, which contains a collection of technical documents (in PDF format). That URL is:

```
http://www.ibm.com/software/webservers/commerce/wcs_pro/lit-tech-general.html
```

The URL for the OS/390 Net.Commerce site is:

```
http://www.s390.ibm.com/nc/ecommerce/doc.html
```

The URL for the Net.Commerce backend integration sample code:

```
http://www.s390.ibm.com/nc/ecommerce/sampcode.html/
```

D.3 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 211.

- *Building e-commerce Solutions with Net.Commerce: A Project Guidebook*, SG24-5417-00
- *AS/400 e-commerce: Net.Commerce*, SG24-2129-00
- *Net.Commerce V3.2 for AS/400: A Case Study for Doing Business in the New Millennium*, SG24-5198-00

- *Migrating Net.Commerce Applications to OS/390*, SG24-5438-00
- *Integrating Net.Commerce with Legacy Applications*, SG24-4933-00
- *IBM WebSphere Commerce Suite SPE Customization*, SG24-5958-00
- *e-Commerce Patterns Using WebSphere Commerce Suite, Patterns for e-business Series*, SG24-6156-00
- *Payment Server V1.2 for AS/400: Secure Transactions in e-commerce*, SG24-5199-00
- *Exploring Net.Commerce Hosting Server*, SG24-5505-00
- *Net.Commerce for OS/390*, SG24-5154-00
- *B2B Collaborative Commerce with Sametime, QuickPlace and WebSphere Commerce Suite*, SG24-6218-00
- *Net.Commerce: Develop on NT or AIX, Deploy on S/390*, SG24-5516-00
- *Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*, SG24-4978-00
- *e-Marketplace Pattern using WebSphere Commerce Suite, MarketPlace Edition Patterns for e-business Series*, SG24-6158-00
- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755-00

D.4 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

D.5 Other resources

These publications are also relevant as further information sources:

- *WebSphere Application Server Standard Edition Planning, Installing, and Using Manual,* GC34-4806
- *WebSphere Application Server for OS/390 HTTP Server Planning, Installing, and Using*, SC31-8690
- *IBM Net.Commerce for OS/390 Configuring and Getting Started*, GC24-5862

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** **ibm.com**/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	**Order Number**	**Quantity**

First name　　Last name

Company

Address

City　　Postal code　　Country

Telephone number　　Telefax number　　VAT number

□ Invoice to customer number

□ Credit card number

Credit card expiration date　　Card issued to　　Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5683-00 IBM WebSphere Commerce Suite V4.1 for OS/390: First Steps
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM Redbooks

IBM WebSphere Commerce Suite V4.1 for OS/390: First Steps

(0.2”spine)
0.17”<->0.473”
90<->249 pages

IBM WebSphere Commerce Suite V4.1 for OS/390: First Steps

Leverage the strength of OS/390 for an e-commerce solution

Provide personalized internet auctions

The most scalable and secure e-commerce solution available

WebSphere Commerce Suite V 4.1 is the latest offering in support of e-commerce for OS/390. It is a flexible, scalable, secure application suite that makes it easy to produce functionality for net-based commerce solutions.

This IBM Redbook describes the architecture and features of WebSphere Commerce Suite, as well as presenting a review of the features carried forward from the predecessor product, Net.Commerce V3.1.2 for OS/390.

This redbook helps you plan for, install, configure, and tailor WebSphere Commerce Suite for OS/390. Detailed instructions for migrating from Net.Commerce to WebSphere Commerce Suite are provided. Step-by-step examples of creating and publishing a Web-based ìstore,î creating a customized on-line auction, and designing personalized pages are also presented.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-5683-00 ISBN 0738418951