Blockchain Development Fundamentals:

Linux Foundation Hyperledger Fabric and Fabric Composer

Matt Lucas - @mqmatt
Global Blockchain Enablement, IBM
The purpose of this lab is to introduce you to development of applications on a blockchain (Linux Foundation Hyperledger Fabric).

- **Section 1** shows you how to use Fabric Composer to quickly create and test a blockchain car auction in a web browser.

- **Section 2** shows you how to run the Hyperledger Fabric and car auction on your local machine.

- **Section 3** shows you how to get started with Hyperledger Fabric on IBM Bluemix.

There are no dependencies between the sections so they can be tackled in any order, although you are recommended to start with section 1.

The tools and technologies you will use in this lab are in the early stages of development and are subject to change. Keep an eye out for any late breaking errata as you progress through.

Thanks, and have fun!
Section 1. Using Fabric Composer

Fabric Composer (www.fabric-composer.org) is an open-source set of tools designed to make building blockchain applications easier.

It allows users to model the business networks, assets and transactions that are required for blockchain applications, and to implement those transactions using simple JavaScript functions. The blockchain applications run on instances of Linux Foundation Hyperledger Fabric (www.fabricnetworks.org). The main page for Linux Foundation Hyperledger is www.hyperledger.org.

The purpose of this lab is to introduce you to the concepts of a blockchain by showing you how a blockchain transfers assets between participants in a business network. We will use the implementation of a simple blind car auction as the scenario for the demo.

The car auction business network has a set of known participants (buyers and sellers), assets (cars and car listings) and transactions (placing bids and closing auctions). We will model these using Fabric Composer and test the business logic that makes the auction work.

Crucially, a blockchain could be used to bring together the buyers and sellers of these assets without needing any trusted third party. However, an auctioneer could be used to provide visibility and governance of the network if required.

Introduction to this section of the lab

Skill requirements:
- There are no skill prerequisites to completing this section although some background knowledge of object-oriented concepts and/or JavaScript is desirable.

Technical pre-requisites:
- Internet Connection
- Any Modern Browser (IE9+)
- A Github ID (register at www.github.com)

This section of the lab takes place entirely in the web browser using Composer Playground. A GitHub logic is required to import the sample into the Composer Playground.

Playground simulates the entire blockchain network within the browser by providing a sandpit environment to define, test and explore business networks defined using Composer. It is possible to connect to a live blockchain Hyperledger Fabric instance, or install the Composer Playground on a local machine for more developer friendly tools.

Fabric Composer Playground is one method to use Fabric Composer, other methods are also available at www.fabric-composer.org.

1.1. Open the Playground

Open a web browser and go to http://composer-playground.mybluemix.net
By default, files are saved to local browser storage. If you have previously run this lab or edited files within this web page, then you might be guided how to clear this local storage. This will ensure that the asset, participant and transaction registries are empty, and that any changes made to the files are discarded.

### 1.2. Import the Car Auction Sample

1. **Click the Import/Replace button.**

   Press the “Authenticate with GitHub” link if present:
3. Log in with your Github ID if necessary to review the available samples directly from Github. Select the carauction-network sample and press the Deploy button:

1.3. Explore the Editor Views

1.3.1. Model File
4. Click the Model File (lib/org.acme.vehicle.auction.cto) to open it, if it is not already open.

This .cto file models the assets, participants and transactions for this blockchain application.

5. Look at the Vehicle asset:

This uses the Fabric Composer Modeling Language which will be looked at more later. An asset is anything of worth that will be transferred around the blockchain. Here we can see the asset class is called ‘Vehicle’ and will have an associated vin and a reference (indicated by “--->”) to a ‘Member’ participant that we will call ‘owner’.

6. Type and add some characters in an appropriate point to show the live validation of the model.

7. Scroll down and look at the abstract ‘User’ participant.

The participant will be the people or companies within the business network. Each User participant will be defined as having a email, firstName and lastName. As the class is abstract instances of it cannot be created; instances are instead implemented by the Member and Auctioneer classes.
Here the user can become a Member requiring a balance, or an Auctioneer that does not.

Look at the Offer and CloseBidding transaction definitions:

The transaction definitions give a description of the transactions that can be performed on the blockchain. They are implemented in a Transaction Processor file using the Javascript language.

1.3.2. Transaction Processors

Click on the lib/logic.js file:

Scroll to the bottom of the file to review the logic used to make an offer on a car being auctioned:
This implements the `makeOffer` function, which is executed when the `Offer` transaction is invoked on the blockchain. (It is the `@param` comment above the function that links the full transaction name as defined by the model to the Javascript method that implements it.)

Other interesting areas of the function implementation include:

a) The logic that the vehicle must be for sale to submit an offer on it 
b) The retrieval and update of the asset registry a few lines later 
c) Saving the updated asset back to the registry

1.3.3. Access Control List

The final file that defines the blockchain application is the Access Control List, which describes the rules which govern which participants in the business network can work with which parts of the blockchain.

__11. Click the permissions.acl file:__

![Access Control permissions.acl](image)

__12. Look at the ACL rules defined:__
The rule allows or denies users to access aspects of the blockchain.

In the next section we will now work with this blockchain application as currently deployed. (It is also possible to make changes to these files and click the ‘Deploy’ button to make them live, although it might be first necessary to Reset the blockchain – see later.)

1.4. Add Three Participants

We will first instantiate three Member participants of the car auction business network:

- Alice Smith (alice@foo.com), who will make a bid on a car,
- Bob Jones (bob@foo.com), who will also make a bid on a car, and
- Charlie Brown (charlie@foo.com), who currently owns a car.

We will not instantiate an Auctioneer in this demo; this could be used in order to provide oversight of the network, although is not necessary.

13. Click the Test tab and then click on the Member participant registry:
The registry is empty as no members have currently been defined.

__14.__ Click Create New Participant to add a new Member.

__15.__ Type the correct values into the JSON data structure to add Alice to the business network. Let's give her a starting balance of 10000.
__16. Click Add to add Alice to the registry.

__17. Do the same for Bob. Let’s give him a starting balance of 5000.
Finally do the same for Charlie. He hasn't got so much money (he's selling his car, after all) so let's give him a starting balance of 100.
19. Verify that all participants in the business network have been correctly defined. Use the appropriate Edit button ( ) to make any changes.
1.5. Add an Asset

We will now add Charlie’s car to the Vehicle Asset registry.

__20. Click the Vehicle asset registry.

__21. This registry contains no assets currently. Click the Create New Asset button to add a new asset.
22. Instantiate the car by adding a vehicle identification number (VIN) of 1234 and assign it to Charlie by filling in the JSON object as follows. (We use his email address to identify him; this was specified as the key field in the User definition using the ‘identified by’ statement.)

23. View your newly added asset in the registry.
1.6. Add a Vehicle Listing

In this section we will put the car up for sale by creating a VehicleListing instance.

24. Click the VehicleListing asset registry. Once more, the VehicleListing registry should be empty.

25. Click the Create New Asset button to add the asset. Press the “Generate Random Data” button to populate the asset with random data.

26. Modify the listingId, reservePrice, description, state and vehicle fields. Also empty the offers array by removing everything between the square braces. Click Add to add the vehicle listing to the registry. Syntactic validation of the object occurs at this point, so correct any errors if necessary.
Create New Asset

In registry: **org.acme.vehicle.auction.VehicleListing**

**JSON Data Preview**

```json
{
    "$class": "org.acme.vehicle.auction.VehicleListing",
    "listingId": "listing1",
    "reservePrice": 500,
    "description": "one careful owner",
    "state": "FOR_SALE",
    "offers": [],
    "vehicle": "1234"
}
```

Just need quick test data?

Generate Random Data  
  Cancel  
  Create New
27. View the listing in the registry.

1.7. Submit offers on the vehicle

We will now let Alice and Bob bid on the vehicle.

28. Click on the Submit Transaction button

29. Let Alice put in a bid of 6000.
__30.  Let Bob put in a bid of 4000.

```json
{
  "$class": "org.acme.vehicle.auction.Offer",
  "bidPrice": 4000,
  "listing": "listing1",
  "member": "bob@foo.com"
}
```

__31.  Verify the transactions in the registry.
Note that the transactions cannot be edited or individually deleted once submitted; this is one of the defining characteristics of a blockchain.

1.8. Closing the bidding

To close the bidding on the listing we need to submit a `CloseBidding` transaction.

32. Submit a new transaction, this time selecting `org.acme.vehicle.auction.ClosingBid` from the drop-down 'Type' field.
33. Verify that the transaction has been added to the blockchain transaction registry.
Based on the bids we submitted, Alice should now be the owner as she put in the highest bid. We should also be able to verify that the owner of the car has changed and appropriate balances increased or decreased accordingly.

__34. Go to the Vehicle asset registry to see the vehicle owner has been updated to Alice.

___

__35. Go to the Member asset registry to see that Charlie’s balance has increased by the winning bid amount, and that Alice’s balance has decreased by the same.

1.9. Other UI Functions

Composer Playground has other features we can explore further in future labs, but to get an understanding of what is available, let’s look at the top right icons:
1.10. Connection Profiles

Click on in the top right of any page

You will see links for how to post questions related to Composer, as well as how to engage with the team. Composer Playground currently only supports the Web Browser connection profile. In the future you will be able to add connection profiles for Hyperledger Fabric instances running in the cloud.

![Connection Profiles](image)

1.11. Updating the Model (Advanced)

36. Try updating the model of the Car asset to include make and model fields. Add in new String fields to the Vehicle definition and click ‘Deploy’ to make the changes live.

Note that when you update the model, the syntax of any existing assets in the registry must be compatible with the new model. Use either the optional or default="…” qualifiers next to the new fields. If you make incompatible changes, you must first reset the demo.

1.12. Export the Business Network Archive
If you have made changes to the business network definition you can export your work to your local file system as a *Business Network Archive*. Business Network Archives contain your source artifacts and can also be deployed to a Hyperledger Fabric instance running locally by installing the composer-cli command line tools. More details on local installation at [http://fabric-composer.org](http://fabric-composer.org).
Section 2. (Optional, Advanced) Running Hyperledger Fabric and Fabric Composer on your local machine

It is possible to install and run Hyperledger Fabric and Fabric Composer on your local machine, and this section guides you through the online documentation sources to do this.

Skill requirements:
- This is a technical section of the lab and is suitable only for those comfortable with software installation, developer tools and the command line on either Linux or Mac.

Technical pre-requisites:
- Depending on what you want to install locally, you might need the following pre-requisites installed:
  - **Operating Systems**: Ubuntu Linux 14.04 LTS (64-bit) or Mac OS 10.12
  - **Docker Engine**: Version 1.12 or greater
  - **Docker-Compose**: Version 1.8 or greater
  - **Node**: 6.x (note 7.x is not supported)
  - **npm**: 4.0.x
  - **git**: 2.9.x

Hyperledger Fabric and Fabric Composer are still maturing technically. The detailed installation instructions are subject to significant change and so have not been printed here.

38. Go to [http://fabric-composer.org](http://fabric-composer.org) and learn more about the Fabric Composer project.


2.1. Installing the Fabric Composer UI (Playground)


At the time of writing, you need the following pre-requisites installed:

- **Docker Engine**: Version 1.12 or greater
- **Docker-Compose**: Version 1.8 or greater

41. The Car Auction will be pre-deployed in the UI, but you can use the UI to import additional samples from Github.

2.2. (Advanced) Installing the Fabric Composer Command-line

If you wish to run Fabric Composer from the commandline (e.g. for scripting) you need to install additional software. At the time of writing, you need the following pre-requisites installed:

- **Operating Systems**: Ubuntu Linux 14.04 LTS (64-bit) or Mac OS 10.12
- **Docker Engine**: Version 1.12 or greater
- **Docker-Compose**: Version 1.8 or greater
__42. Follow the quickstart instructions at https://fabric-composer.github.io/start/quickstart.html

__43. Follow the instructions at https://github.com/fabric-composer/sample-networks/tree/master/packages/CarAuction-Network to get the car auction application deployed locally.

As part of the installation instructions you will be guided to download and install the Hyperledger Fabric docker image.
Section 3. (Optional) Running Hyperledger Fabric on IBM Bluemix

Bluemix is IBM’s platform-as-a-service and provides a web-based playground in which you can run blockchain instances. As of the time of writing, it is not possible to connect Fabric Composer to an instance of Hyperledger Fabric running on Bluemix, although it is possible to use it to develop native blockchain applications, as well as deploy some pre-canned demos.

Skill requirements:
- Familiarity with blockchain concepts is desirable for this section.

Technical pre-requisites:
- Internet Connection
- Any Modern Browser (IE9+)

### 3.1. Deploying the sample application

In this section we will use Bluemix to deploy a copy of the car leasing demo application.

__44. Open a web browser and go to [www.bluemix.net](http://www.bluemix.net).

__45. Click ‘Sign Up’ or ‘Log In’ to create a new Bluemix account or log into your existing account.

__46. Once you have successfully signed up and logged into Bluemix, select Catalog from the top bar.

__47. In the ‘Services’ section of the sidebar, click ‘Application Services’ and select Blockchain.

![](image)

__48. Review the service description and information about the service.

__49. Click ‘View Docs’ and learn about the process of creating a blockchain environment.
__50. Click ‘Samples and Tutorials’ on the right of the page.

__51. Click against the Car Lease demo. Log in to Bluemix again if necessary.

__52. If you are asked to pick an alias (usually the first time a given Bluemix ID deploys a sample), come up with a unique username and review and accept the terms of use. Click Create and then Continue on the subsequent page.

__53. Wait a few seconds for the default field values to be populated. Then leave the App Name, Region, Organization and Space default (unless told otherwise) and click Deploy.

This will cause the car leasing demo to be deployed into your Bluemix environment, and will probably take a few minutes to complete.
54. Once you see the ‘Success!’ message click to see a description of the new car leasing application (and associated Blockchain service) you created.

3.1.1. Initializing the Car Leasing Demo

55. Click the blue hyperlink under the ‘Route’ column of your application, and this will load the Car Leasing demo webpage. (Do not click elsewhere on this line, as this will load the administration interface for the application, which we will look at later).

You will now see the front page of the car leasing demo.
56. From the Car Leasing demo front page, click ‘Admin Console’ and ‘Create Simple Scenario’.

This will preload the blockchain with a set of transactions. (The Full Scenario works fine too; the difference between the Simple Scenario and the Full Scenario is that in the Full Scenario more assets are initially loaded onto the blockchain; this takes longer to initialize, however.)

Wait a few moments for the initialization to complete.

57. Click ‘OK’ to close the Creating Scenario log, and then dismiss the ‘Scenario Creation complete’ by clicking the check mark.

58. (Optional) Use the demo to transfer the ownership of cars between the different business network participants. See how the blockchain is used to track the transactions.
3.2. Managing the sample application

We will now use the monitoring tools available inside the Bluemix environment to view and manage the blockchain.

__59. Return to the Bluemix dashboard, either by selecting the icon in the top left of Bluemix and selecting 'Dashboard' or by going directly to https://console.ng.bluemix.net/dashboard/applications.

__60. Click on the car_lease_blockchain service in the Services section of the dashboard.

__61. Review the details and select 'Launch Dashboard' to launch the blockchain administration console.
You should now see the administration page with seven tabs down the left-hand side. The 'Network' tab will be selected by default.

The blockchain is a replicated, shared ledger. This blockchain is shared among all the participants of the network. Each participant still has their own copy of the ledger, and replication ensures that the copies are kept synchronised.

The blockchain network that has been set up for us in this demo contains four participants (“Validating Peers”) as well as a Membership Services component that we will look at later. Applications submit transactions into just one validating peer, and peer-to-peer technology is used to replicate the transaction elsewhere.

62. Verify that the four validating peers each have the same block height.

63. Return to the car leasing demo and invoke another transaction. Verify that the block height increases by one for all four validating peers.

3.2.1. Viewing the Blockchain

The Blockchain tab shows a visual representation of the state of the Blockchain.

64. Click the ‘Blockchain’ tab at the left of the page.
The icons show:

- **Total number of blocks in the chain**
- **Average number of blocks per hour**
- **Average number of transactions per block**
- **Number of deployment calls made to deploy chaincode**
- **Number of invoke requests made within this blockchain**

Each block contains a set of transactions. In Hyperledger Fabric, a transaction is the record of the request to interact with chaincode (a smart contract). The two most important transaction types are:

- **DEPLOY**: The request to deploy a piece of chaincode across all validating peers, so that it can be executed at a later date.
- **INVOKE**: The request to invoke a piece of chaincode (for example, invoke the chaincode to transfer the ownership of a car)

Other request types exist (e.g. query). Not all request types are recorded on the blockchain.
The blocks also include when that block was committed to the blockchain.

__65. Click on a block that contains at least one invocation request.__

![Block Activity Table](image)

__66. Look through the list of transactions that are contained within the block.__

<table>
<thead>
<tr>
<th>Transaction Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
</tr>
<tr>
<td>11/22 03:33p m UTC</td>
</tr>
</tbody>
</table>

Each line of information is a transaction stored within the block. A block may contain multiple transactions but in this demo there will often only be one transaction per block due to the low frequency of transactions being made. The information displayed is:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>The date the transaction was submitted.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of transaction taking place (e.g. INVOKE or DEPLOY).</td>
</tr>
<tr>
<td>UUID</td>
<td>The unique identifier for each transaction.</td>
</tr>
<tr>
<td>Chaincode ID</td>
<td>Refers to the chaincode that is being invoked or deployed.</td>
</tr>
<tr>
<td>Payload</td>
<td>The input parameters to the chaincode.</td>
</tr>
</tbody>
</table>

__67. Repeat this for other blocks to understand how the transactions are stored.__

*When the Blockchain service is initialised for the car leasing application, the first block in the chain should contain a ‘DEPLOY’ transaction, where the chaincode is deployed to the validating peers.*

*View these blocks if you’re willing to scroll down the Blockchain explorer that far!*
3.2.2. Understanding the Blockchain Peers

We are now going to review the logs associated with the peers. This is useful for understanding how the blockchain works, and for diagnosing problems.

__68. Click on the ‘Logs’ tab.__

![Image of a network interface with logs tab selected](image)

By looking at the logs for each peer you can verify that every node has executed every transaction.

__69. Click the Logs button against one of the validating peers.__

This will show the logs for the selected peer in a new window.
3.2.3. Interacting with the peers

It is possible to invoke the management APIs that interact directly with the peers. In this section we will be trying out these APIs directly from the Bluemix environment.

Note that the APIs concern operationally managing the Blockchain service – this is not the same as adding and invoking transactions through chaincode!

70. Click on the ‘APIs’ tab on the dashboard.
This page allows you to invoke APIs that will directly interrogate and manage the blockchain. First we will use the API interface to query the height of the Blockchain (the number of blocks).

_71._ Click the ‘Blockchain’ section.

This reveals the GET /chain operation which is a valid operation to call on the peer.

_72._ Select the operation to view information about it.

This reveals the input and output data formats.
__73. Click ‘**Try It Out**’ to invoke the API.

**Curl**

```bash
curl -X GET --header "Accept: application/json" "https://6code68ce53c435798950c27cf547b-vp0.us.blockchain.ibm.com:5001/chain"
```

**Request URL**

`https://6code68ce53c435798950c27cf547b-vp0.us.blockchain.ibm.com:5001/chain`

**Response Body**

```json
{
  "height":85,
  "currentBlockHash": "53PwwW1nMwKNrRk5Sb+aGiZlgCz2EAaGFZuwWvRglVeWH/GYUomGOmgrE4DvAjjwOGkKz8aAV6xGc6Af4w==",
  "previousBlockHash": "7KeBuJdAt+6HEz0I+PppqJ3RfVAAWtBxglg2Q9n2T7Xoa8Kpsw41UNRUYLPcn4UVQRqdy8gDHMbs54ovRldw=="
}
```

**Response Code**

200

**Response Headers**

```json
{"content-type": "application/json"}
```

Review the displayed fields:

- The **Curl** field shows how to perform the same request from a command-line or script.
• The Request URL shows the URL that was invoked, including the endpoint information of the peer (hostname:port) and the method call (/chain).

• The Response Body shows the information that was returned including, importantly, the height of the blockchain.

• The Response Code 200 shows that the request was successful.

• The Response Headers confirms that the response body has been returned in a JSON data structure.

The blockchain is immutable: it is append-only and transactions cannot be modified or deleted once committed to the blockchain. Hash functions are used to link the blocks in the chain together; each block is linked to the previous block by a hash of the previous block’s contents. If transactions are tampered with, the hash function returns a different value which renders the blockchain un-navigable.

A hash function is simply a function that is applied to a data set that produces a consistent output. It is usually used to map data of an arbitrary size to data of a fixed size. Importantly for blockchain, any change to the input data set will produce a different hash output, which can be used to easily detect any modifications to a block.

In the JSON response body, the ‘height’ field shows the number of blocks in the blockchain; the ‘currentBlockHash’ is a hash function that has run over the most recent block and ‘previousBlockHash’ is the same for the block before it.

__74. Note the first few characters of the value of the currentBlockHash (“53Pwe…” in the previous screenshot).

__75. Invoke another transaction in the car leasing demo to force another block to be created.

__76. Re-run the GET /chain operation; verify that the height has increased by 1 and that the new previousBlockHash is the same as the previous block’s currentBlockHash (“53Pwe…”).

__77. Click the ‘Block’ section and click on the ‘GET /chain/blocks/{Block}’ operation. In the ‘Block’ text field, enter the number one less than the current height of the chain (for example if the height was 86, enter 85).
The returned JSON structure contains several elements, including:

(a) a transactions array, which describes the set of transactions in the block. The description of each transaction includes its type (1/invoke or 2/deploy), the unique identifier of the associated chaincode and the encoded input parameters to it (payload).

(b) a stateHash, which is the result of running a hash function over the transaction output,

(c) a previousBlockHash, which is the result of running a hash function of the previous block in the chain, and

(d) a nonHashData element, which contains data that will not be used in the computation of the next block’s previousBlockHash.

Note that the previousBlockHash field matches the previousBlockHash returned by the GET /chain operation.

When a new block is created, a hash function is run over the entire previous block (except the nonHashData element) and the result stored in the previousBlockHash element. This way, if any earlier block in the chain is tampered with, subsequent blocks will be invalid.

Particularly, note that the previousBlockHash element is itself used to calculate the hash of subsequent blocks. This means that even a small change to one of the first blocks in the chain can be detected in any future block.
We will look at some other fields in this data structure in the next section.

__78. Copy the txid field of a transaction from a block; this will be a unique identifier of the form "04421f7d-652a-491d-90b0-7bc9f29b2d85".

__79. Click the ‘Transactions’ section.

This reveals the GET /transactions/{UUID} operation which is a valid method to call on the peer.

__80. Paste the transaction UUID and click ‘Try it out’.

The ‘payload’ field is base64 encoded (use a web tool such as http://www.base64decode.org for decoding this information); when decoded you’ll see that the payload includes the chaincode ID of the smart contract being called together with its input parameters. For example:

Note that this application does not encrypt the transactions, so the payloads are visible (albeit base64 encoded) to all.
3.2.4. Viewing the Service Status, Support Contacts and Samples

81. Click on the ‘Status’ tab at the top of the service page.
   This page shows you the recent availability of the Blockchain service on Bluemix, and also the version of Hyperledger Fabric that is being used by your network.

82. Click on the ‘Support’ tab at the top of the service page.
   This page shows you how to get more help with IBM Bluemix and the Blockchain service.

83. Click on the ‘Demo Chaincode’ tab at the top of the service page.
   This page gives the opportunity to deploy more samples to the Blockchain service, and also some how to get started with writing your own blockchain applications and chaincode.

We will look at chaincode development in more detail in the follow-on lab “Blockchain Unchained”.

3.3. Removing the sample application

The final section of this lab aims to stop and remove the Blockchain service you created.

__84. Return to the Bluemix Dashboard (https://console.ng.bluemix.net/dashboard/applications).

__85. Click the three vertical dots at the right of the Car Leasing application Settings icon in the car lease demo application and select ‘Delete App’ from the menu.

__86. Ensure that the ‘car_lease_blockchain’ service is selected for deletion and click ‘Delete’.

__87. Wait for the items to be stopped and deleted. Once this is done, both the application and the associated service will no longer be visible in the Bluemix dashboard.
Appendix A. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

<table>
<thead>
<tr>
<th>IBM</th>
<th>AIX</th>
<th>CICS</th>
<th>ClearCase</th>
<th>ClearQuest</th>
<th>Cloudscape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube Views</td>
<td>DB2</td>
<td>developerWorks</td>
<td>DRDA</td>
<td>IMS</td>
<td>IMS/ESA</td>
</tr>
<tr>
<td>Informix</td>
<td>Lotus</td>
<td>Lotus Workflow</td>
<td>MQSeries</td>
<td>OmniFind</td>
<td></td>
</tr>
<tr>
<td>Rational</td>
<td>Redbooks</td>
<td>Red Brick</td>
<td>RequisitePro</td>
<td>System i</td>
<td></td>
</tr>
<tr>
<td>System z</td>
<td>Tivoli</td>
<td>WebSphere</td>
<td>Workplace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.