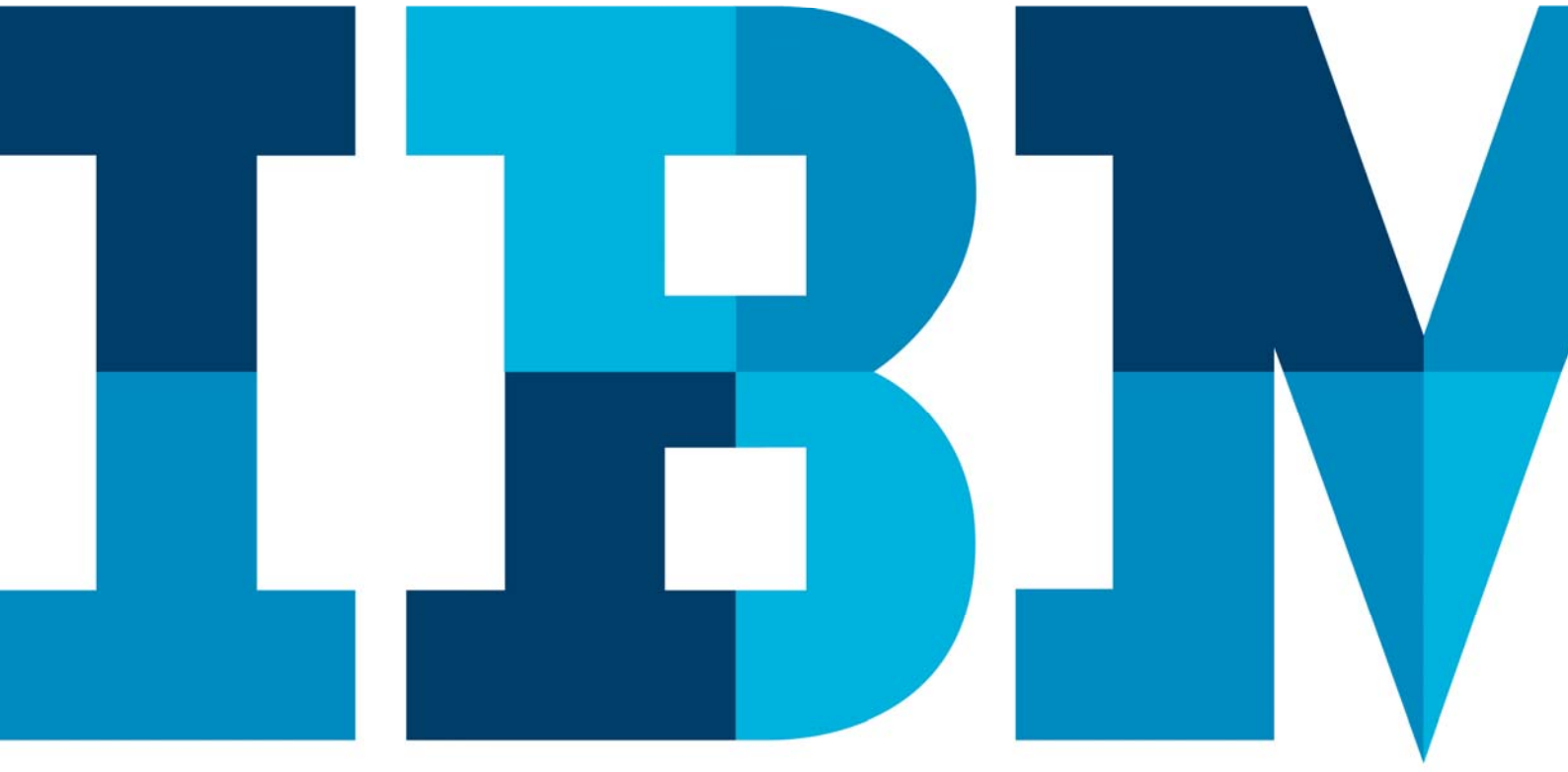
IBM Blockchain Hands-On
Blockchain Unchained

Lab Three –Bluemix –Exercises



IBM

Contents

Overview

The aim of this lab is to introduce you to chaincode development by showing you how to create and deploy your first chaincode to a new Blockchain service in Bluemix.

We will use a sample piece of chaincode (Smart Contract) called **example02** as the foundation for our lab. This sample is provided as part of the Hyperledger Fabric code and accessible directly through Bluemix.

Once deployed, the chaincode can be invoked and queried.

Introduction

Prerequisites:

- A Firefox or Chrome browser with access to www.bluemix.net.
- A Bluemix account
- In order to edit and compile chaincode yourself, you also need:
  - GitHub account andtools (from https://github.com/)
  - GO Lang compiler installed (from https://golang.org/dl/)

It is recommended that students have previously completed the Blockchain Explained and Blockchain Explored labs.

For V0.61 of Hyperledger in Bluemix

Section 1. Creating the Blockchain Service on Bluemix

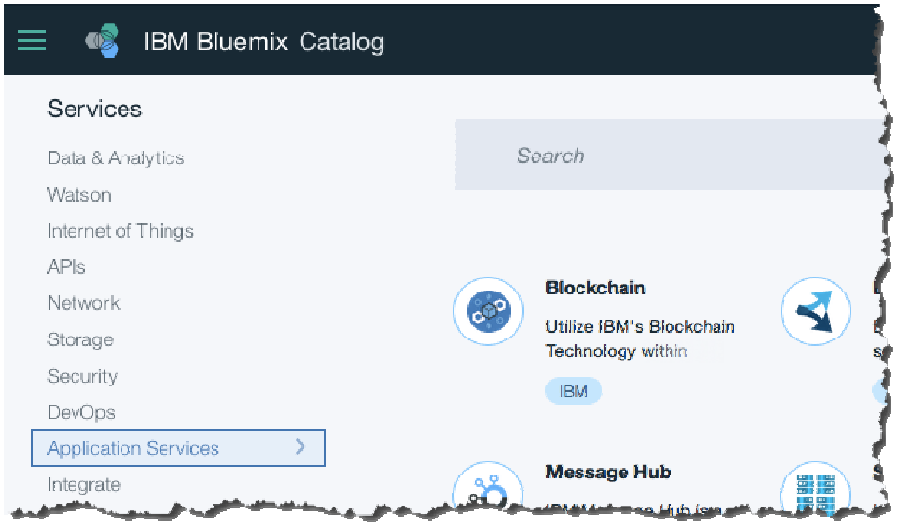
In this section we will use Bluemix to create a new Blockchain service. It does not require any services or applications to be pre-installed (such as the Car Leasing demo).

__1. Open a web browser (Firefox or Chrome are recommended) and go to www.bluemix.net.

__2. Click ‘Sign Up’ or ‘Log In’ to create a new Bluemix account or log into your existing account.

__3. Once you have successfully signed up and logged into Bluemix, select **Catalog** from the top bar.

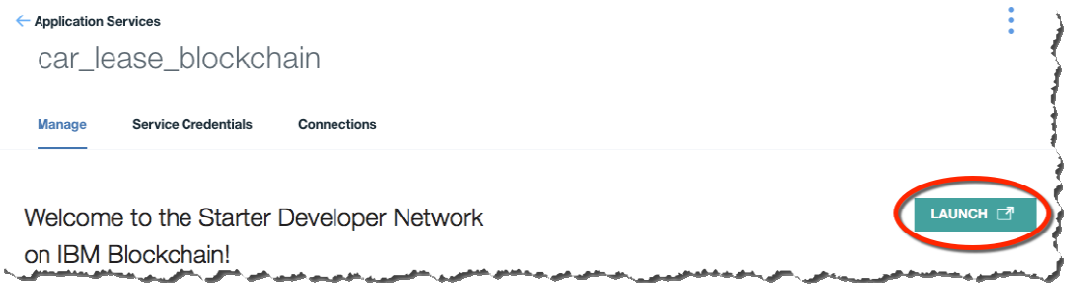
__4. In the ‘Services’ section of the sidebar, click ‘Application Services’ and select **Blockchain**.



__5. Review the service description and information about the service.

__6. Ensure that “Starter Developer plan” is selected and click **Create** accepting the prompts to create an “org” and a “space” if you are prompted to do so. Wait for the service to be created.

__7. Click “Launch” to launch the blockchain administration console.

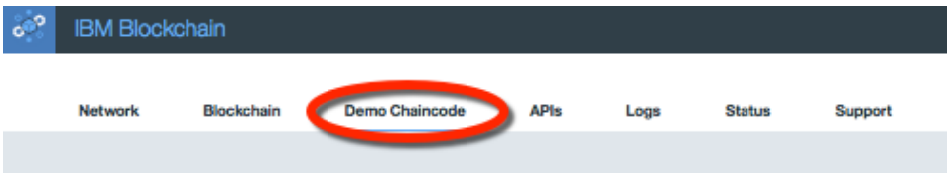


__8. Dismiss the welcome message.

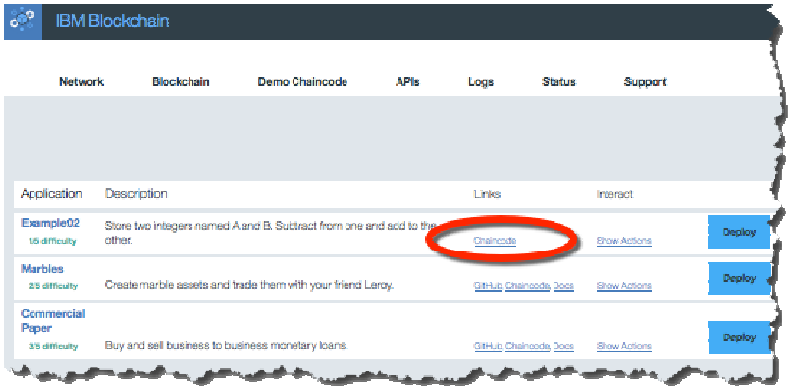
Section 2. Reviewing the Chaincode

In this section we will deploy some sample chaincode to the newly created blockchain.

__9. From within the blockchain administration console click the “**Demo Chaincode**” tab.



__10. Click the “**Chaincode**” link next to Example02.



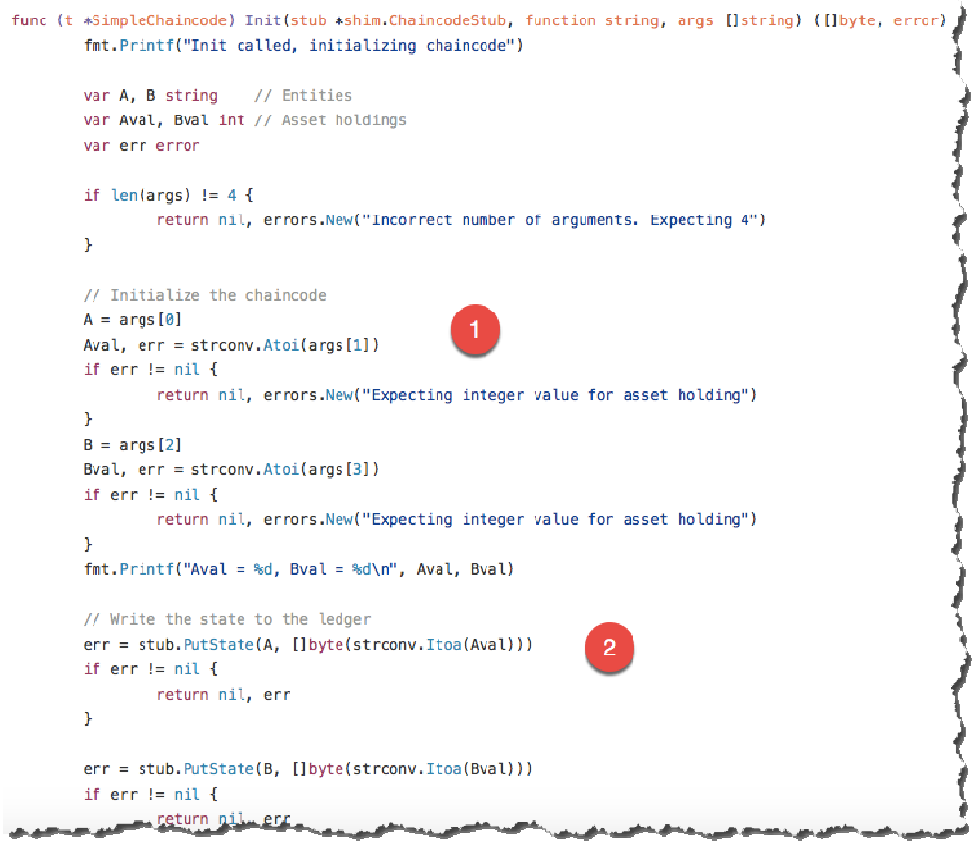
This will take you to the Github site that contains the source code for the sample.

Upon initialization, this chaincode creates two key/value pairs in the world state. Each key is an identifier string (e.g. “A”) whose value is an associated integer balance (e.g. 100). Transactions that invoke this chaincode will increment the balance of one identifier while decrementing the other.

For example:

Initializing the chaincode with [“a”, “100”, “b”, “200”] will set up “a” to be “100” and “b” to be “200”.
Invoking a transaction with [“a”, “b”, “10”] will decrement the value of “a” by 10 and increment “b” by the same amount.
Querying the value of “a” and “b” at this point will yield “a” to have the value “90” and “b” to have “210”.

__11. Scroll down to the **Init** method.



```
func (t *SimpleChaincode) Init(stub *shim.ChaincodeStub, function string, args []string) ([]byte, error) {
        fmt.Printf("Init called, initializing chaincode")

        var A, B string    // Entities
        var Aval, Bval int // Asset holdings
        var err error

        if len(args) != 4 {
                return nil, errors.New("Incorrect number of arguments. Expecting 4")
        }

        // Initialize the chaincode
        A = args[0]
        Aval, err = strconv.Atoi(args[1])
        if err != nil {
                return nil, errors.New("Expecting integer value for asset holding")
        }
        B = args[2]
        Bval, err = strconv.Atoi(args[3])
        if err != nil {
                return nil, errors.New("Expecting integer value for asset holding")
        }
        fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

        // Write the state to the ledger
        err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
        if err != nil {
                return nil, err
        }

        err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
        if err != nil {
                return nil, err
```


Highlighted section (1) of the code is extracting the four initialization parameters (“A”, “100”, “B”, “200”). The code checks that they are of the required types.

Highlighted section (2) is storing those key/value pairs into the blockchain.

__12. Scroll down to the **invoke** method.

```
// Transaction makes payment of X units from A to B
func (t *SimpleChaincode) invoke(stub *shim.ChaincodeStub, args []string) ([]byte, error) {
        fmt.Printf("Running invoke")

        var A, B string    // Entities
        var Aval, Bval int // Asset holdings
        var X int          // Transaction value
        var err error

        if len(args) != 3 {
                return nil, errors.New("Incorrect number of arguments. Expecting 3")
        }

        A = args[0]
        B = args[1]

        // Get the state from the ledger
        // TODO: will be nice to have a GetAllState call to ledger
        Avalbytes, err := stub.GetState(A)
        if err != nil {
                return nil, errors.New("Failed to get state")
        }
        if Avalbytes == nil {
                return nil, errors.New("Entity not found")
        }
        Aval, _ = strconv.Atoi(string(Avalbytes))

        Bvalbytes, err := stub.GetState(B)
        if err != nil {
                return nil, errors.New("Failed to get state")
        }
        if Bvalbytes == nil {
                return nil, errors.New("Entity not found")
        }
        Bval, _ = strconv.Atoi(string(Bvalbytes))

        // Perform the execution
        X, err = strconv.Atoi(args[2])
        Aval = Aval - X
        Bval = Bval + X
        fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

        // Write the state back to the ledger
        err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
        if err != nil {
                return nil, err
        }

        err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
        if err != nil {
                return nil, err
        }

        return nil, nil
}
```

Highlighted section (1) of the code retrieves the current values of the two keys from the world state.

Highlighted section (2) of the code increments and decrements the values of the keys accordingly.

Highlighted section (3) of the code writes the updated key/value pairs back to the world state.

__13. Scroll down to the **Query** method.

```
// Query callback representing the query of a chaincode
func (t *SimpleChaincode) Query(stub *shim.ChaincodeStub, function string, args []string) ([]byte, error) {
        fmt.Printf("Query called, determining function")

        if function != "query" {
                fmt.Printf("Function is query")
                return nil, errors.New("Invalid query function name. Expecting \"query\"")
        }
        var A string // Entities
        var err error

        if len(args) != 1 {
                return nil, errors.New("Incorrect number of arguments. Expecting name of the person to query")
        }

        A = args[0]

        // Get the state from the ledger
        Avalbytes, err := stub.GetState(A)
        if err != nil {
                jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
                return nil, errors.New(jsonResp)
        }

        if Avalbytes == nil {
                jsonResp := "{\"Error\":\"Nil amount for " + A + "\"}"
                return nil, errors.New(jsonResp)
        }

        jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) + "\"}"
        fmt.Printf("Query Response:%s\n", jsonResp)
```

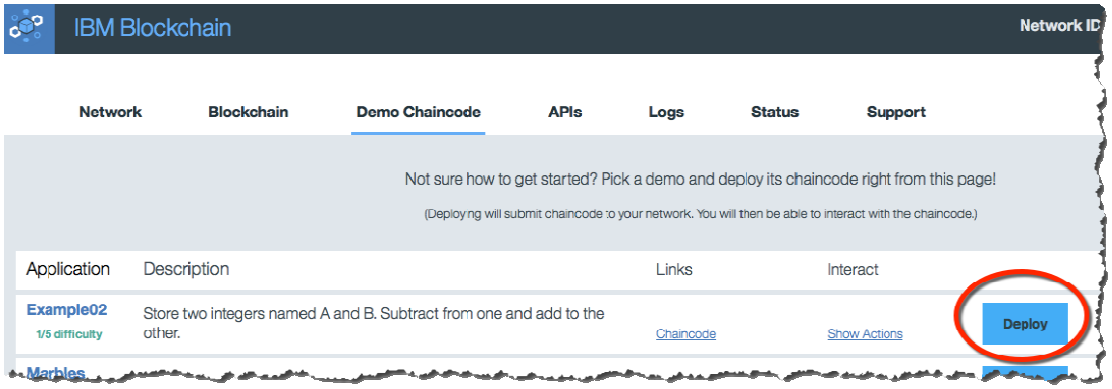
1

Highlighted section (1) retrieves the value of a key from the world state. This is converted into a JSON data structure and returned to the caller.

Section 3. Deploying, Invoking and Querying the Chaincode

We will now deploy the chaincode to our Blockchain service and make sure it works.

__14. Return to the blockchain administration console in Bluemix and click on the '**Deploy**' button next to the Example02 application.

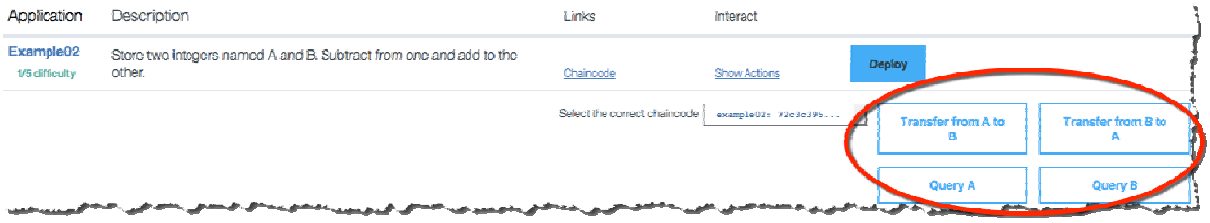


The panel at the bottom of the web page will show the process of deployment. Notice that a "user" is logged into the blockchain service before the chaincode can be deployed to the validating peers. The entire process should only take a few seconds to complete.

```
Registering enrollID dashboarduser_type1_0
  Success - registering enrollID

Deploying chaincode https://github.com/masterDev1985/hyperledger_chaincode/chaincode_example02
  Success - deployment (wait for the cc to start up)...
```

Once the deployment has completed, the web page will be updated to show the actions that can be performed against the chaincode.



__15. Click '**Query A**' to show the value of the "A" key. This is displayed in the log.

```
Querying function - query ["a"]
  Success - query 100
```

__16. Click '**Transfer from A to B**' to decrement the value of "A" by 5 and increment "B" by the same amount.

```
Invoking function - invoke
  Success - invocation 933b18be-d305-4e2e-a97b-2daf378376ca
```

__17. Click '**Query A**' to view the updated value of "A".

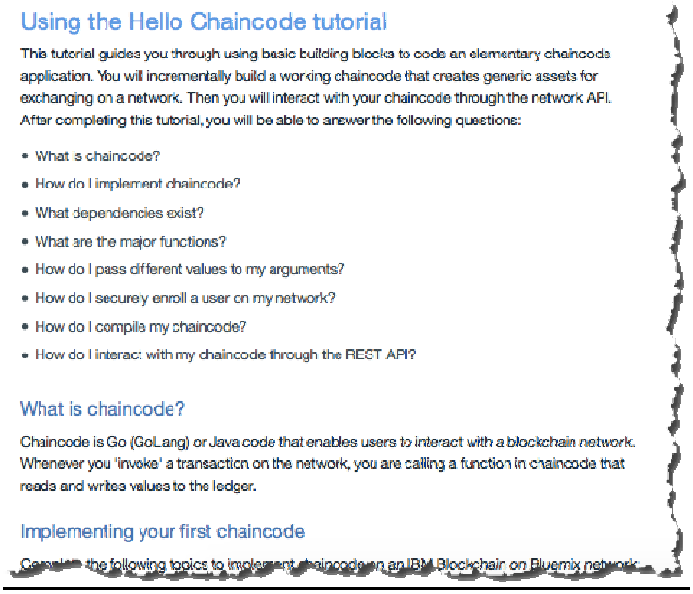
```
Querying function - query ["a"]
  Success - query 95
```

In the next section of the lab, we will edit, compile and redeploy the chaincode on your local machine. If you are not in a chaincode development role, it is recommended you end the lab now as this is an advanced section that has a number of software pre-requisites for you to install.

Section 4. Extending the Chaincode

In this advanced section of the lab, we will now go and modify the chaincode. This requires you to have the ability to do local development on your machine.

__1. Return to the Bluemix Blockchain service documentation “Samples and tutorials” page (https://console.ng.bluemix.net/docs/services/blockchain/ibmblockchain_tutorials.html). Scroll down to the “Using the Hello Chaincode tutorial” section.



Using the Hello Chaincode tutorial

This tutorial guides you through using basic building blocks to code an elementary chaincode application. You will incrementally build a working chaincode that creates generic assets for exchanging on a network. Then you will interact with your chaincode through the network API. After completing this tutorial, you will be able to answer the following questions:

- What is chaincode?
- How do I implement chaincode?
- What dependencies exist?
- What are the major functions?
- How do I pass different values to my arguments?
- How do I securely enroll a user on my network?
- How do I compile my chaincode?
- How do I interact with my chaincode through the REST API?

What is chaincode?

Chaincode is Go (GoLang) or Java code that enables users to interact with a blockchain network. Whenever you 'invoke' a transaction on the network, you are calling a function in chaincode that reads and writes values to the ledger.

Implementing your first chaincode

Complete the following topics to implement a chaincode on an IBM Blockchain on Bluemix network.

__2. Follow the “Implementing your first chaincode” section of this documentation. In summary you will need to:

a. Install and configure Golang

b. Set up GitHub and fork the example “Hello Chaincode” repository.

c. Clone the “Hello Chaincode” fork to your local system. Make sure you clone the forked version rather than the original.If you accidently clone the original repository, then you can point to your fork using:

```
git remote set-url origin https://github.com/<yourGitID>/learn-chaincode
```

d. Edit the chaincode_start.gofile to add in new capability for invoking and querying the chaincode.

e. Commit your local changes with:

```
git commit –a
```

f. Upload your changes back to your forked repository with "**`git push`**".

g. Using the Blockchain service API in Bluemix, deploy, invoke and query the new version of chaincode.

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.

Appendix B. Trademarks and copyrights

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

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

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.

NOTES

NOTES

IBM Software