Extending TADDM discovery with Jython Scripts

Document version 0.2

Andrzej Wrobel(andrzej.wrobel@pl.ibm.com)

Table of Contents

Introduction

This paper describes how to extend the core functionality of IBM® Tivoli® Application Dependency Discovery Manager (TADDM), which is discovery using Custom Server Extensions and Jython scripts.

This paper covers all types of discovery, including SNMP. A collection of scripts is proved that addresses different requirements. Some of the examples are already available on TADDM product web pages. Each script is for a different domain and clearly describes techniques to store any object through Custom Server Extensions.

Overview

When deploying an application like TADDM in a complex IT environment, it is common to create Custom Server Templates (CST) to discover more configuration items. Using a CST you can attach a Jython script with any logic that is required to meet business goals. Moreover, a CST contains a method to store instances of a Model Object. This helps you to discover and store almost any object i defined in Common Data Model (CDM).

Instructions for importing scripts into TADDM are described in Appendix 1. You can look at the product documentation to learn more about CST definitions in the Discovery Management Console at http://pic.dhe.ibm.com/infocenter/tivihelp/v46r1/topic/com.ibm.taddm.doc_721fp1/UserGuide/c_cmdb_customserversextending.html.

All scripts were tested on TADDM version 7.2.1.2 .

Each topic contains only a snippet from the original script. The following code is a prerequisite for any CSX Jython script:

```
import sys
import java

from java.lang import System
from java.lang import Class
from java.lang import Long
from com.collation.platform.util import ModelFactory
from com.collation.platform.logger import LogFactory
from java.util import Properties
from java.io import FileInputStream

coll_home = System.getProperty("com.collation.home")

System.setProperty("jython.home",coll_home + "/external/jython-2.1")
System.setProperty("python.home",coll_home + "/external/jython-2.1")

jython_home = System.getProperty("jython.home")
sys.path.append(jython_home + "/Lib")
sys.path.append(coll_home + "/lib/sensor-tools")
sys.prefix = jython_home + "/Lib"

import traceback
import string
import re
import jarray

# Local App Imports

import sensorhelper
```

```
#--------------------------------------------------------------------------
# Define some CONSTANTS, print some info
#--------------------------------------------------------------------------

#--------------------------------------------------------------------------
# Define the Functions
#--------------------------------------------------------------------------

########################
# LogError      Error logger
########################
def LogError(msg):
    '''
    Print Error Message using Error Logger with traceback information
    '''
    log.error(msg)
    (ErrorType, ErrorValue, ErrorTB) = sys.exc_info()
    traceback.print_exc(ErrorTB)

########################
# LogDebug      Print routine for normalized messages in log
########################
def LogDebug(msg):
    '''
    Print Debug Message using debug logger (from sensorhelper)
    '''
    # assuming SCRIPT_NAME and template name are defined globally...
    # point of this is to create a consistent logging format to grep
    # the trace out of
    log.debug(msg)
########################
# LogInfo Print routine for normalized messages in log
########################
def LogInfo(msg):
    '''
    Print INFO level Message using info logger (from sensorhelper)
    '''
    # assuming SCRIPT_NAME and template name are defined globally...
    # point of this is to create a consistent logging format to grep
    # the trace out of
    log.info(msg)
```

Note that a list of imports can vary. It is recommended that you download the scripts to avoid any potential failures. To download the scripts, click the links provided on the pages of this paper.

Populating software components

The goal of this scenario is to extend the default discovery of software components by a Linux sensor. The case is that not all packages are returned using the **`rpm -qa`** command. You can manually create additional components in the script or parse the output of the command.

```
(os_handle,result,server,seed,log) = sensorhelper.init(targets)
sc_list = []
list(sc_list)
```

Create SoftwareComponent using newModelObject: All attributes that form the naming rule must be populated (parent, name, software version).

```
webmod = sensorhelper.newModelObject("cdm:sys.SoftwareComponent")
webmod.setName("WebModule")
webmod.setSoftwareVersion("1.0")
webmod.setParent(server.getOSRunning())
sc_list.append(webmod)
```

Instead of manually creating software components, a command can be executed on the endpoint and returned data is parsed to create a software component list:

```
    raw_components = os_handle.executeCommand("cat /etc/sc_list")
    component_list = raw_components.split("\n")
        for component in component_list:
          parts = component.split(":")
          if len(parts) < 3:
                log.debug("Skipping: " + component)
                continue
          name = parts[1].strip()
          version = parts[2].strip()
          description = parts[7].strip()
          if re.search("\S",name) == None or re.search("\S",version) == None:
                log.debug("Bad Data :" + name + " " + version)
                continue
          log.debug("Got SoftwareComponent: name=" + name + " version=" + version)
          sc =
ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.sys.SoftwareComponent"))
          sc.setParent(computersystem.getOSRunning())
          sc.setName(name)
          sc.setSoftwareVersion(version)
          sc.setDescription(description)
          sc_list.append(sc)
```

The following code presents how to get a list of software components that were discovered by the native Linux sensor (the same logic applies to other OS sensors). It is required that you merge the two lists. Otherwise, the **Packages** tab in the user interface would contain a list from the script only, and the

data from the sensor would be overwritten.

```
#Retrieve SC list from sensor
extRes = result.getExtendedResults()
sensorSC = extRes[0].getOSRunning().getSoftwareComponents()
sc_list.extend(sensorSC)
```

```
#There are no arrays in python, need to convert the sequence to
#a java array so that it can be used by TADDM
sc_array = jarray.array(sc_list, Class.forName("com.collation.platform.model.topology.sys.SoftwareComponent"))
server.getOSRunning().setSoftwareComponents(sc_array)
result.setComputerSystem(server)
```

Setting an internal relationship based on a KVM discovery

The version of TADDM used in this paper does not support KVM discovery. KVM is a virtualization infrastructure for the Linux kernel. The Custom Server Extension for KVM is a good example to show how to store internal dependencies (called implicit) between model objects. The use case is to discover a KVM host system and to create shallow computer systems that are virtualized by the host system. To do that, at least one Naming Rule must be populated. In the case of KVM, UUID is available. You can run a discovery of the guest system directly through Level 2 sensors and data will be reconciled.

```
 try:
      host = os_handle.getComputerSystem()
      guestRaw = sensorhelper.executeCommand("virsh list --all | grep running | awk '{print $2}'")
      component_list = guestRaw.split("\n")
      guest_list = []
      list(guest_list)
      for guestName in component_list:
            csGuest =
ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.sys.ComputerSystem"))
            csGuest.setName(guestName.strip())
            guestUUID = sensorhelper.executeCommand("cat /etc/libvirt/qemu/" + guestName + ".xml | grep -i uuid | awk
-F\">\" '{print $2}' | awk -F\"<\" '{print $1}'")
            csGuest.setsetUUID(guestUUID)
            csGuest.setHostSystem(host)
```

```
            guest_list.append(csGuest)
```

In the prior code example, the host variable is a reference to an object that was discovered by the native sensor.

To find correct name of the method to call, you can look at the CDM web page and append the **set** prefix.

Figure 1 CDM

```
      log.info("Found " + str(len(guest_list)) + " Guests")

      #There are no arrays in python, need to convert the sequence to
      #a java array so that it can be used by TADDM
      sc_array = jarray.array(guest_list, Class.forName("com.collation.platform.model.topology.sys.ComputerSystem"))
      computersystem.setChildSystem(sc_array)
      result.setServer(appserver)

except:
      LogError("KVM.py:  Could not create KVM relationships")
```

Creating dependencies based on a Rational ClearCase discovery

The architecture of the Rational ClearCase is shown in Figure 2.

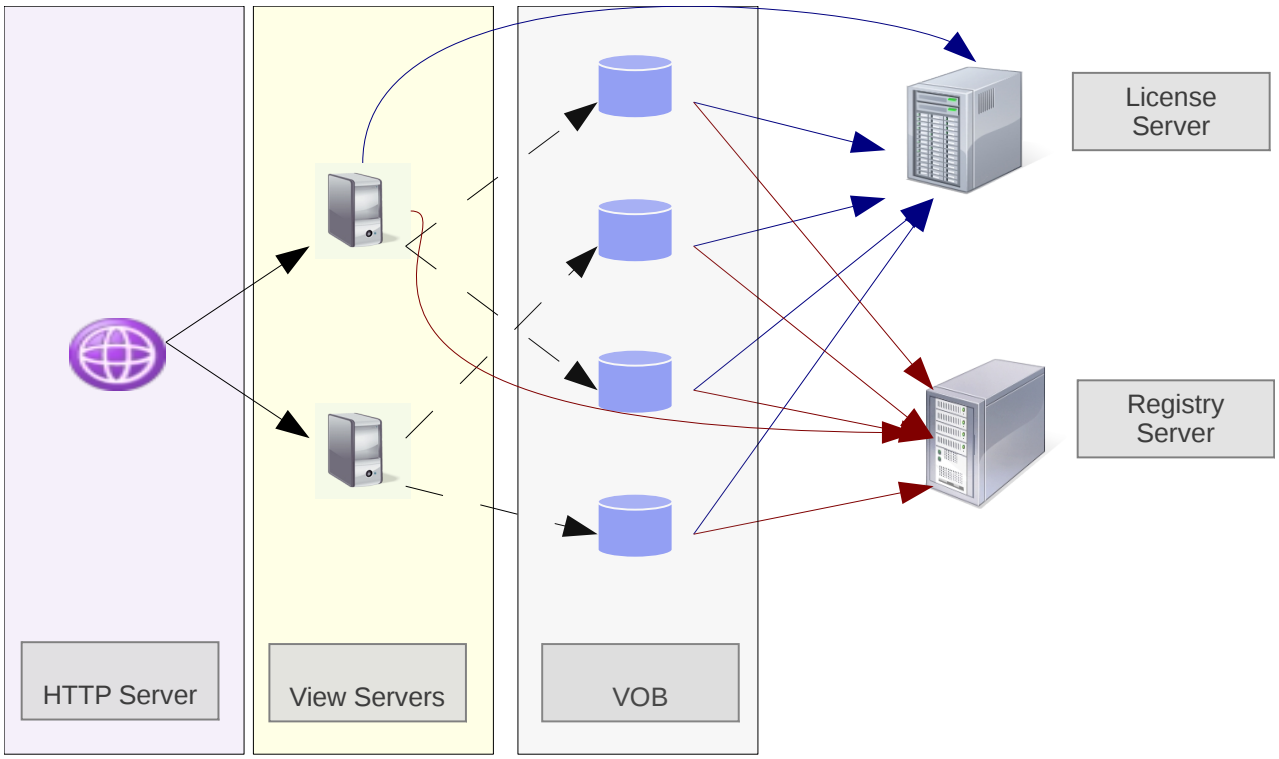


Figure 2 Rational ClearCase architecture

Because the number of components and dependencies is large, it is critical to have all of them discovered for impact analysis. Each component can run on a different server.

Rational ClearCase components do not have their representation in the TADDM Common Data Model. However, it is still possible to model them using generic objects and present relationships using transactional dependencies. Rational ClearCase discovery can be split into two parts:

- View and VOB Server discovery
- License / Registry Server discovery through the albd_server process

View and VOB Server discovery

The code for discovering the View Server describes two important functions:

- Placeholders
- Transactional dependencies

Placeholders are new in version 7.2.1. You can use Placeholders to create shallow configuration items that use ManagedSystemName as a naming rule. When the **isPlaceholder** property value is set to true, it identifies such objects in the TADDM database. If the object is discovered by the native sensor, it

reconciles with the placeholder. In case of the View or VOB Server, we know the name of the License and Registry server from the system commands. What you now know is the computer system that is hosting this function. Therefore, it is not possible to create an AppSever object with the primarySAP naming rule.

```
 log.debug("Create shallow AppServer for License Host with MSN=" + region+':'+licHost)
shLicHost = ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.app.AppServer"))
shLicHost.setManagedSystemName('License:' + region + ':' + licHost)
shLicHost.setObjectType('License Server')
shLicHost.setIsPlaceholder(1)
```

Important: The ManagedSystemName rule must be exactly the same for the two sources discovering the configuration item.

Transactional Dependencies in TADDM are mostly created by Topology Builder agents to show links between two components. They can also be added to a particular configuration item either manually (user interface) or with an API. The following code code snippet shows how to create them in a Jython script:

```
 licTrDep =
ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.app.dependencies.TransactionalDependency"))
licTrDep.setSource(appserver)
licTrDep.setTarget(shLicHost)
licTrDep.setType('TransactionalDependency')
result.addExtendedResult(licTrDep)
```

where appserver object is passed from sensor:

```
(os_handle, result, appserver, seed, log, env) = sensorhelper.init(targets)
```

License / Registry Server discovery

A discovery of License and Registry Server components reveals the storing of any Model Object without linking it with the AppServer object that triggered CSX.

```
cthostinfo = '/opt/rational/clearcase/bin/cleartool hostinfo -l'
rgyHost = sensorhelper.executeCommand("head -1 /var/adm/rational/clearcase/config/rgy_hosts.conf").strip()
licenseHost = sensorhelper.executeCommand("head -1 /var/adm/rational/clearcase/config/license_host").strip()
hostname = sensorhelper.executeCommand("hostname").strip()
region = sensorhelper.executeCommand(cthostinfo + " | grep region | awk '{print $NF}'").strip()
productInfo = sensorhelper.executeCommand(cthostinfo + " | grep \"Product:\"").strip()
```

```
    log.debug("hostname="+hostname)
    log.debug("rgyHost="+rgyHost)
    log.debug("licenseHost="+licenseHost)
    log.debug("productInfo="+productInfo)

    host = os_handle.getComputerSystem()

    product = productInfo.split(" ")
    if(rgyHost.startswith(hostname)) :
        shRegHost =
ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.app.AppServer"))
        shRegHost.setObjectType('Registry Server')
        shRegHost.setName(rgyHost)
        shRegHost.setProductName(product[1])
        shRegHost.setProductVersion(product[2])
        shRegHost.setVendorName('IBM')
        msn = 'Registry:' + region + ':' + rgyHost
        log.debug("MSN for Registry Host is " + msn)
        shRegHost.setManagedSystemName(msn)
        shRegHost.setHost(host)
        result.addExtendedResult(shRegHost)
```

At this point, the extended result contains an object created inside the Jython script with a set of attributes. Similar logic is created for License Server.

The interesting part is at the end of the code:

```
    result.setServer(None)
```

The outcome of such an operation is that the CSX result will not store the appserver object, but it will store any object in the extended result. This is needed when you do not want to create additional configuration items that only increase noise in the database.

Extending SNMP discovery

the following example shows the discovery of new data using an SNMP protocol. The script was designed to extend product functionality when scanning Juniper Netscreen devices. Query of the Interface MIB was required to complete the correct values for the hardware address of each L2 interface.

The input for extending SNMP discovery is always OID where data is reachable:

```
ipAddressTableEntryOID = ".1.3.6.1.4.1.3224.9.1.1"
```

the above entry represents NETSCREEN-INTERFACE-MIB (v1)

You must have adefinition for each attribute to correctly map MIB types.

```
nsIfIndex = IndexDef(1, MibIndexType.INTEGER)

nsIfMAC = ValueDef(11, MibValueType.PHYSADDRESS)
nsIfName = ValueDef(2, MibValueType.STRING)
nsIfDescr = ValueDef(22, MibValueType.STRING)
nsIfStatus = ValueDef(5, MibValueType.INTEGER)
```

If the type is incorrectly assigned, an error occurs during discovery.

Next, a reference to MIB Query Capability is retrieved, to execute SNMP queries.

```
capability_factory = sensorhelper.getSimpleCapabilitiesFactory(address)

mibQueryCapability = capability_factory.getMibQueryCapability(None)
vDef = [nsIfIndex, nsIfMAC, nsIfName, nsIfDescr, nsIfStatus]
l2Interfaces_list = []
list(l2Interfaces_list)
sensorL2Int = computer_system.getL2Interfaces()
rSet = mibQueryCapability.queryTable(ipAddressTableEntryOID, vDef)
```

The result contains a list of objects. Each entry is handled to do further data modifications (like creating Common Data Model objects).

```
iter = rSet.iterator()
while(iter.hasNext()):
    row = iter.next()
    Index = row.get(0).getString()
    nsIfMAC = row.get(1).getString()
    nsIfName = row.get(2).getString()
    nsIfDescr = row.get(3).getString()
    nsIfStatus = row.get(4).getString()
```

Retrieving an Access List in Jython Script

The last topic in this paper details the of retrieving of Access List entries to execute queries that require additional authorization. The example focuses on Oracle Discovery, where an active JDBC connection is not passed to the CSX from the sensor. In such cases, you must create a new JDBC connection and pass all required parameters such as the user and password.

```
connStr = "jdbc:oracle:thin:@%s:%s:%s" % (seed.getPrimaryIpAddress().getStringNotation(),port,sid)
driver = OracleDriver()
DriverManager.registerDriver(driver)
authList = AuthManager.getAuth(AuthManager.ORACLEAUTH, seed.getPrimaryIpAddress())
for auth in authList:
    try:
        conn = DriverManager.getConnection(connStr,auth.getUserName(),auth.getPassword())
```

Important: The types for AuthManager are not publicly available. For specific types, contact IBM

Support.

If the credentials are not correct, the getConnection method raises an exception which is caught at the end of execution block:

```
        except:
          LogError("Couldn't connect, trying another user from access list")
```

When the connection is successfully established, you can start executing queries:

```
            stmt = conn.createStatement()
          rsetGN = stmt.executeQuery("select global_name from global_name")
          rsetGN.next()
          output = rsetGN.getString(1)
          print "getGlobalName: oracle global database name: " + str(output)
          oraDB.setName(output)

          # Get parameters from v$sys_optimizer_env
          rset = stmt.executeQuery("SELECT name,value from v$sys_optimizer_env")
          while (rset.next()):
             strName = rset.getString(1).strip()
             if len(strName) == 0:
                continue

             else:
                optimizer =
ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.app.db.oracle.OracleInitValue"))
                optimizer.setParent(oraDB)
                optimizer.setName(strName)
                optimizer.setValue(rset.getString(2))
                oracleOptimizerList.append(optimizer)
                print "Name: %s Value: %s" % (rset.getString(1),rset.getString(2))
```

The returned data is flat and well structured. Therefore, the script contains simple parse logic which in the result creates required CDM objects (such as OracleInitValue) and populates all required attributes.

Summary

The scripts explained in this paper present different aspects of implementing Custom Server Extensions and show how TADDM can be extended to increase breadth and depth of the discovery. Because each IT environment is unique, each deployment requires adjustments. The goal of this paper is to support deployments to faster achieve expected results. If any of the aspects have not been covered in this paper, please contact the author.

Appendix 1

To import KVM Custom Server Extension, complete the following steps:

1. Place the KVM.csx file in the /tmp/ directory.
2. As a TADDM user, go to the TADDM Server directory at dist/bin.
3. Run the templateloader.jy script.

```
[taddmusr@localhost bin]$ ./templateloader.jy -u administrator -p password import /tmp/KVM.csx
Extracting Manifest
Extracting XML
Extracting Descriptor
Extracting Script
Validating template
Validating Seed Class
Placing template for KVM  into /opt/IBM/taddm/dist/etc/templates/templates/KVM.xml
Placing descriptor for KVM  into /opt/IBM/taddm/dist/etc/templates/commands/KVM
Creating /opt/IBM/taddm/dist/etc/templates/commands/extension-scripts/KVM.py
Restart TADDM Server in order for the changes to take effect.
```

4. Restart the TADDM Server.
5. Confirm that the Custom Server Template is imported correctly.

Figure 3 Custom Servers section

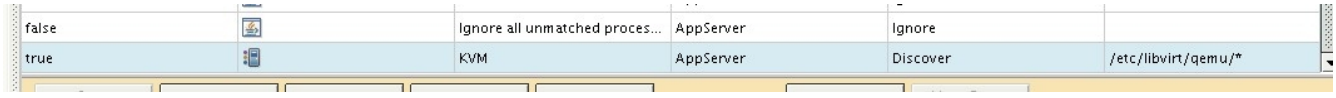


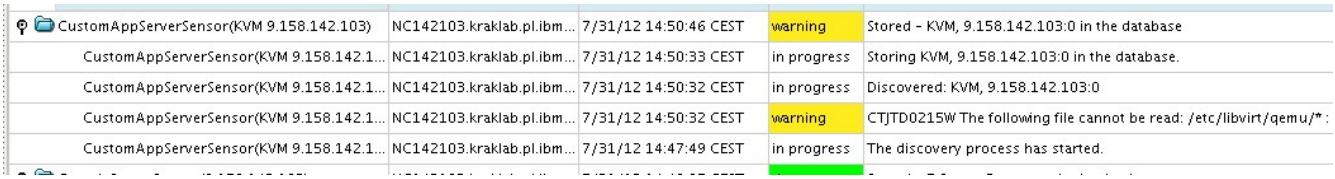
Figure 4 KVM Custom Server Template

Important: The KVM libvirtd process, which is the entry point for starting Custom Server Template does not have any TCP port open. Therefore , the default forced server list for UNIX systems must be

extended:

com.collation.platform.os.UnixOs.forcedServerList=amqzxma0;vxconfigd;clstrmgr**;libvirtd**

6. Run a discovery with CustomAppServerSensor enabled in the profile.



CustomAppServerSensor(KVM 9.158.142.103)	NC142103.kraklab.pl.ibm...	7/31/12 14:50:46 CEST	warning	Stored - KVM, 9.158.142.103:0 in the database
CustomAppServerSensor(KVM 9.158.142.1...	NC142103.kraklab.pl.ibm...	7/31/12 14:50:33 CEST	in progress	Storing KVM, 9.158.142.103:0 in the database.
CustomAppServerSensor(KVM 9.158.142.1...	NC142103.kraklab.pl.ibm...	7/31/12 14:50:32 CEST	in progress	Discovered: KVM, 9.158.142.103:0
CustomAppServerSensor(KVM 9.158.142.1...	NC142103.kraklab.pl.ibm...	7/31/12 14:50:32 CEST	warning	CTJTD0215W The following file cannot be read: /etc/libvirt/qemu/* :
CustomAppServerSensor(KVM 9.158.142.1...	NC142103.kraklab.pl.ibm...	7/31/12 14:47:49 CEST	in progress	The discovery process has started.

Figure 5 Discovery Results

A warning informs you that TADDM was not able to find any files in the /etc/libvirt/qemu directory.

You can remove the storing of those files by editing the Custom Server Template definition.

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 give 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:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan, Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 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 might 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

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

2Z4A/101

11400 Burnet Road

Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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 measurement 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 or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

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. 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 IBM‘s application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked