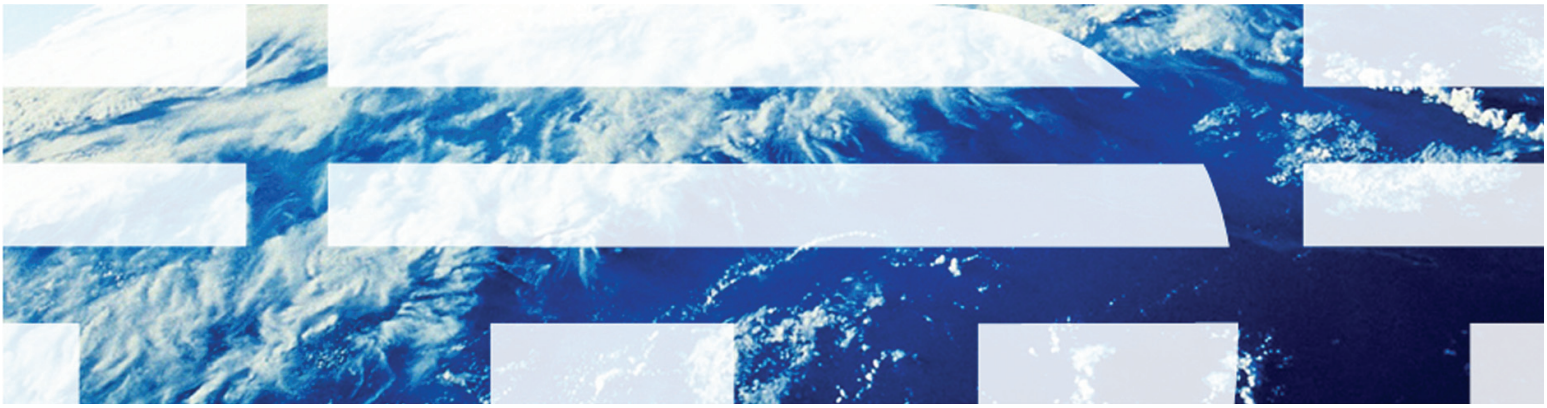


IBM Tivoli NetView for z/OS 6.1: Automation Techniques

Student's Training Guide

Course: TZ213 ERC: 1.0

August 2011

Printed in Ireland

Table of contents

Unit 1: Introduction to NetView automation

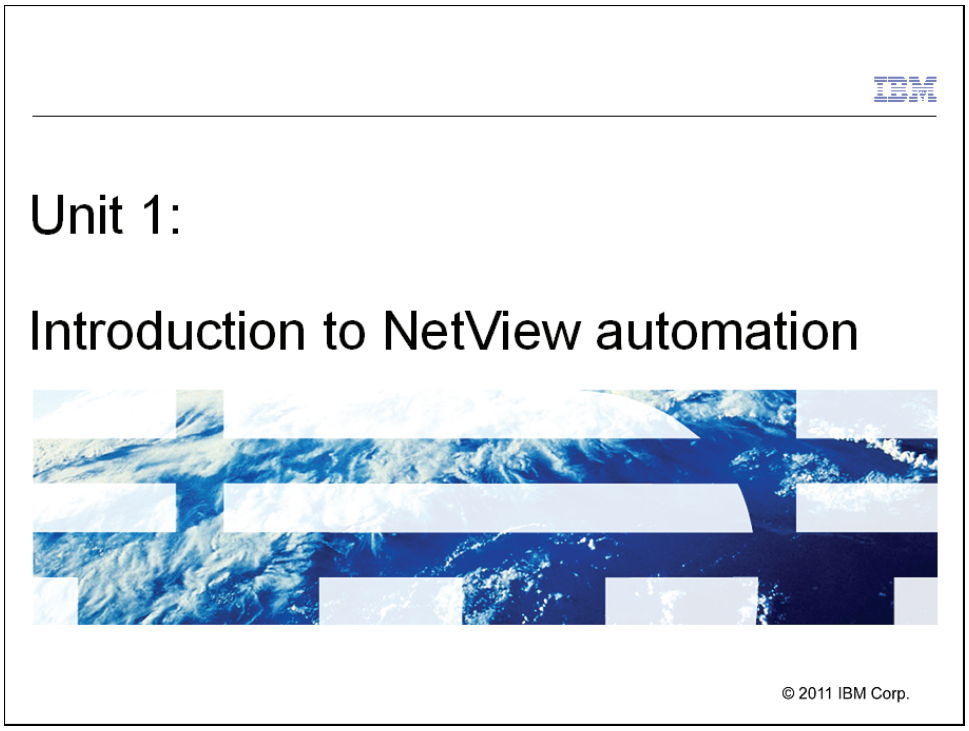
Unit 2: Message automation topics

Unit 3: NetView commands for facilitating automation

Unit 4: Managing the NetView automation table

Unit 5: Coding an automation table

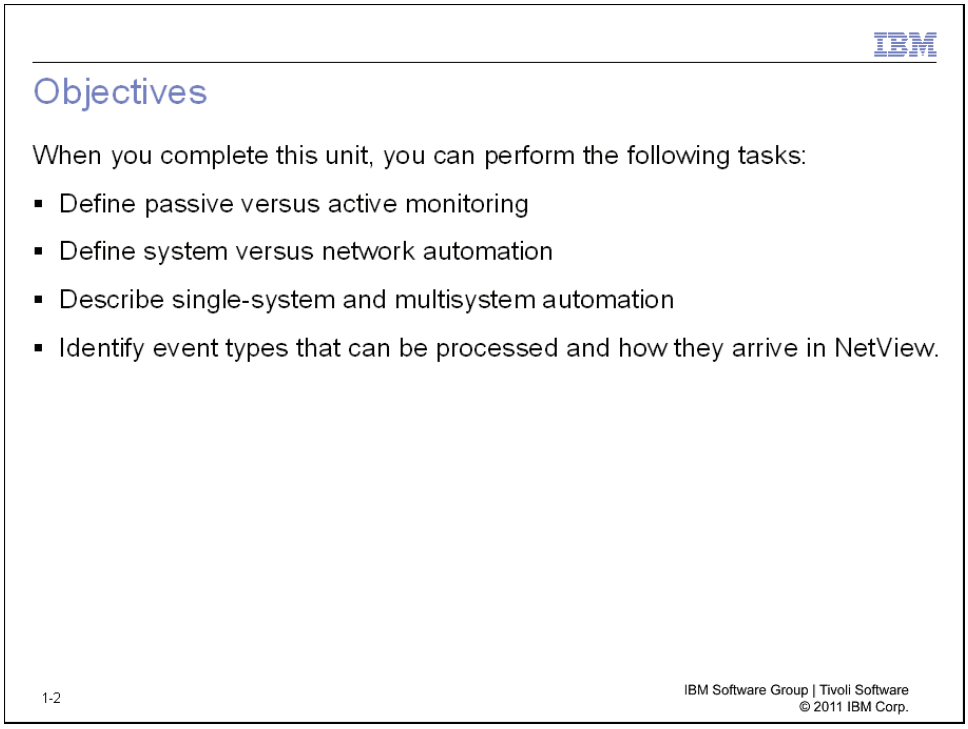
Unit 1: Introduction to NetView automation



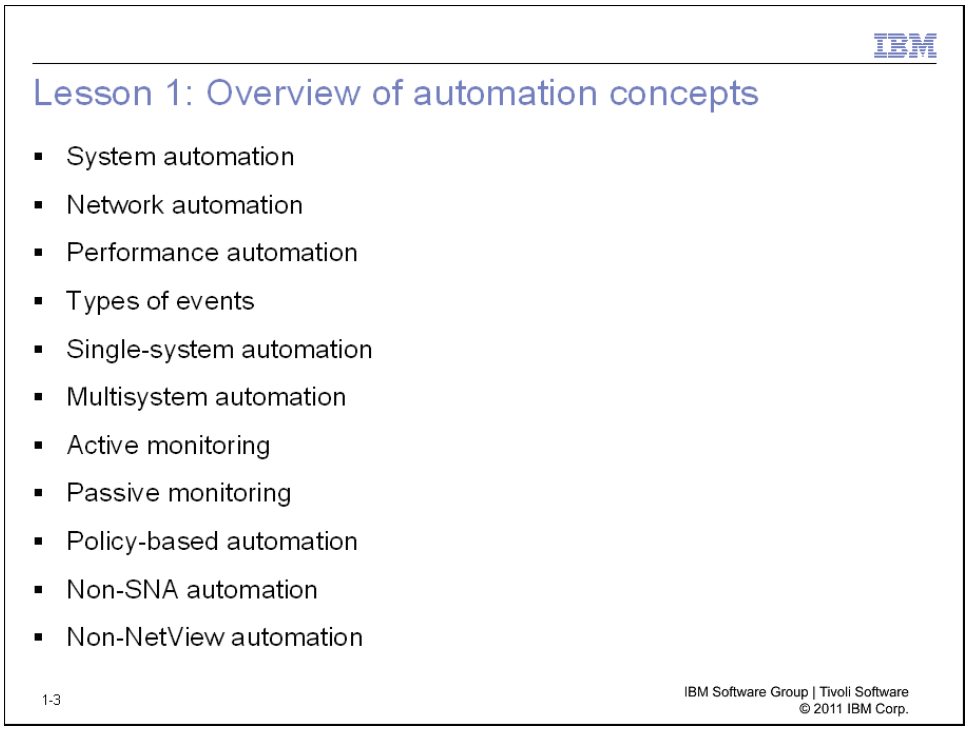
Introduction

This unit introduces you to many of the automation concepts, such as monitoring of resources, automation of system and network resources, single-system and multisystem automation, and so on.

Objectives



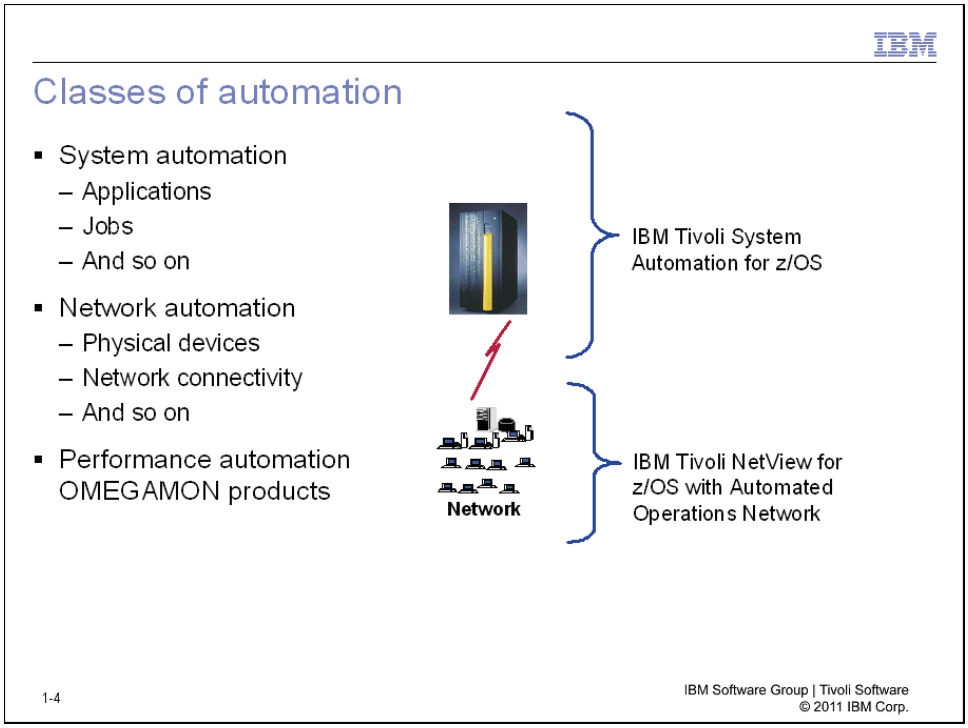
Lesson 1: Overview of automation concepts



This lesson introduces several automation topics as follows:

- System versus network automation
- Single-system versus multisystem automation
- Active monitoring versus passive monitoring
- And more

Classes of automation



System automation focuses on system resources, such as applications, batch jobs, started tasks, and so on. IBM® Tivoli® System Automation for z/OS® (SA z/OS) is an IBM product that you can purchase for automating your systems and sysplexes (system complexes).

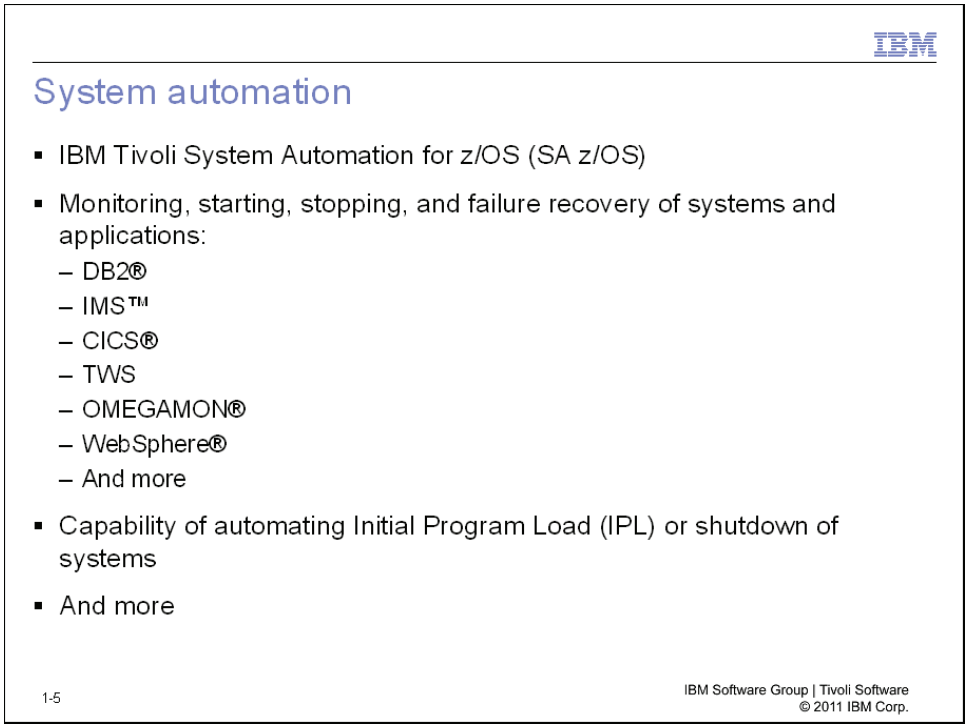
SMF type 30 record automation

The MVS IEFACTRT SMF installation exit receives control from the system when a job or job step ends, either normally or abnormally. The NetView® program provides a sample IEFACTRT exit (CNMSMF3E) that passes data across the program-to-program Interface (PPI) to a receiver. The receiver issues a message containing the data that can be processed by using NetView automation facilities. This enables quicker and more consistent responses for jobs that end abnormally.

Network automation focuses on network resources, such as System Network Architecture (SNA) lines, physical units (PUs), connectivity between systems, and so on. The Automated Operations Network (AON) component of IBM Tivoli NetView for z/OS provides automation for your VTAM and TCP/IP networks. AON is available with NetView.

Performance automation focuses on how well the resources handle their current workload. For example, with SA z/OS and OMEGAMON®, you can monitor for exception conditions and take additional actions. You can start another instance of an application to improve throughput.

System automation

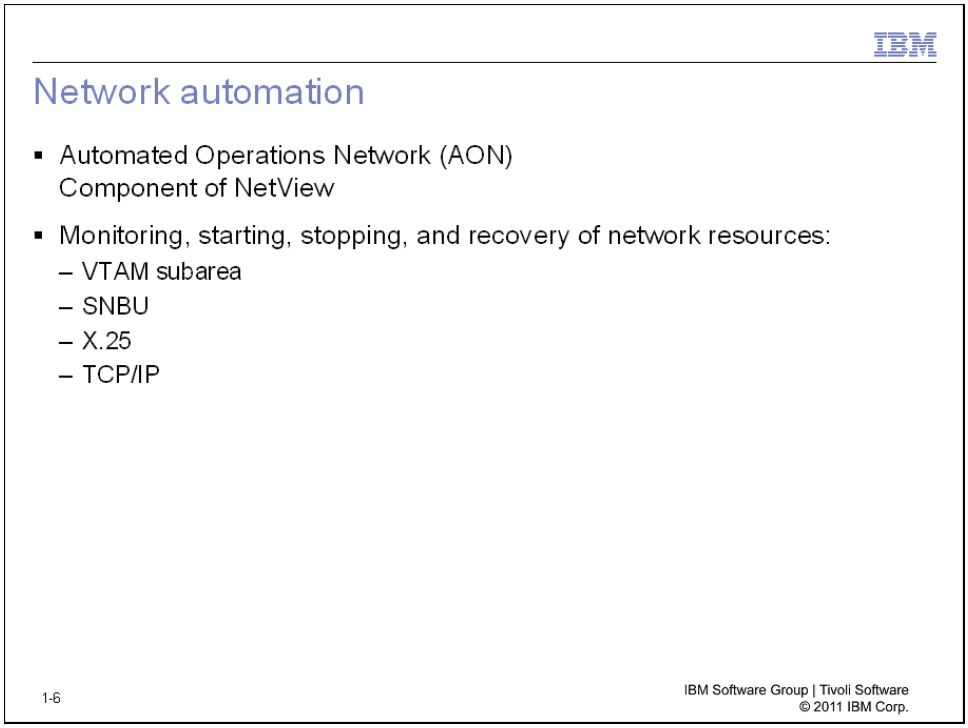


SA z/OS uses policy to automate resources in a system and sysplex (system complex). SA z/OS handles the starting, stopping, monitoring, and failure recovery of the resources for DB2®, IMS™, and so on, across all systems in your sysplex or subplex. Systems are also handled like a resource and can be started (IPL) and stopped (shutdown).

SA z/OS provides you with default policy that includes relationships. For example, you can issue a start request against an application, and SA z/OS starts other prerequisites or dependent applications.

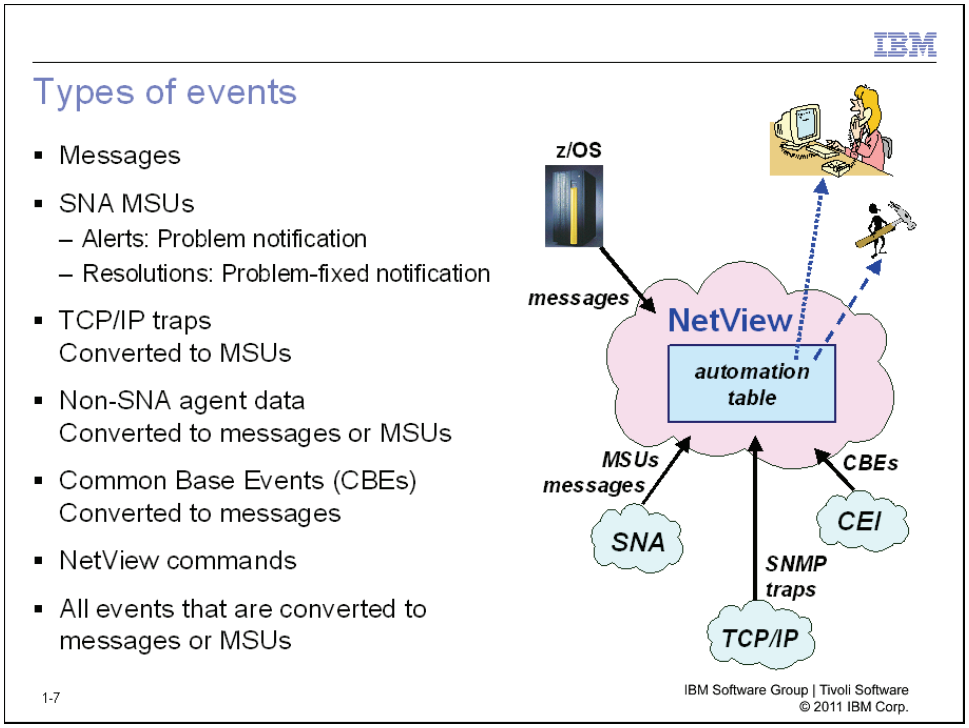
The SA z/OS automation policy is defined with the customization dialog, a set of Interactive System Productivity Facility (ISPF) panels.

Network automation



The AON component of NetView provides default automation for your network resources. AON monitors, starts, stops, and recovers your VTAM SNA and TCP/IP resources, for example.

Types of events



Each type of event is translated into a message or a Management Services Unit (MSU) that NetView automation is to process. The most common form of event is a message. Most z/OS products issue messages that pertain to events within the product, system, or network.

MSUs are SNA events that are represented as alerts (problem) or resolutions (problem fixed). MSUs use an architected structure that is similar to an envelope for data. TCP/IP traps, which are converted to MSUs, can be generated by agents and sent to NetView for automation. Non-SNA data is packaged in MSUs or as messages by an agent and sent to NetView for automation.

Common Base Events (CBEs) are a standardized representation of events across hardware and software. CBEs are converted to messages before being sent to NetView automation.

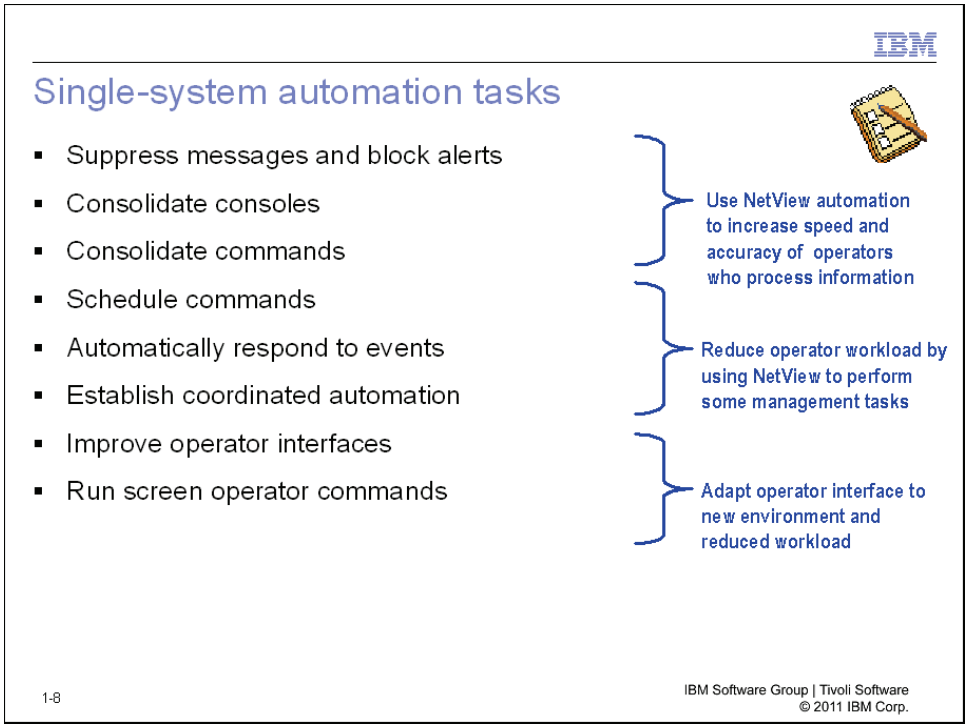
Commands are also eligible for automation. All NetView commands drive the NetView automation table. Most customers add a statement to the automation table to ignore command automation. This discussion occurs later in the class. NetView also provides a *command revision exit* that allows you to modify the text of a z/OS command before it is issued.

Events that arrive at NetView are tested against statements in the automation table. All events are translated to either a message or an MSU before being run through the automation table. If a match is found in the automation table, the event can be displayed or suppressed, and one or more actions can be taken.

This class shows you the following skills:

- How to trap events in the automation table
- Controlling the processing of the event (for example, display or suppress it)
- How to take actions
- Where to route the actions

Single-system automation tasks



This slide illustrates the steps in automation of a single z/OS system. The beginning stages involve reducing the amount of work your operations staff is asked to perform. Later stages involve automation to recover failing resources, further reducing the workload on your operations staff.

Many products send their messages to the system console. Most of those messages are informational. The first step in automation is determining the messages that are important and suppressing those that are not. By using automation to reduce the number of messages displayed on the console, the operations staff can be more effective.

The second step in automation is console consolidation. By reducing the number of messages displayed on the consoles in the first step, you reduce the number of consoles, further improving the effectiveness of your operations staff.

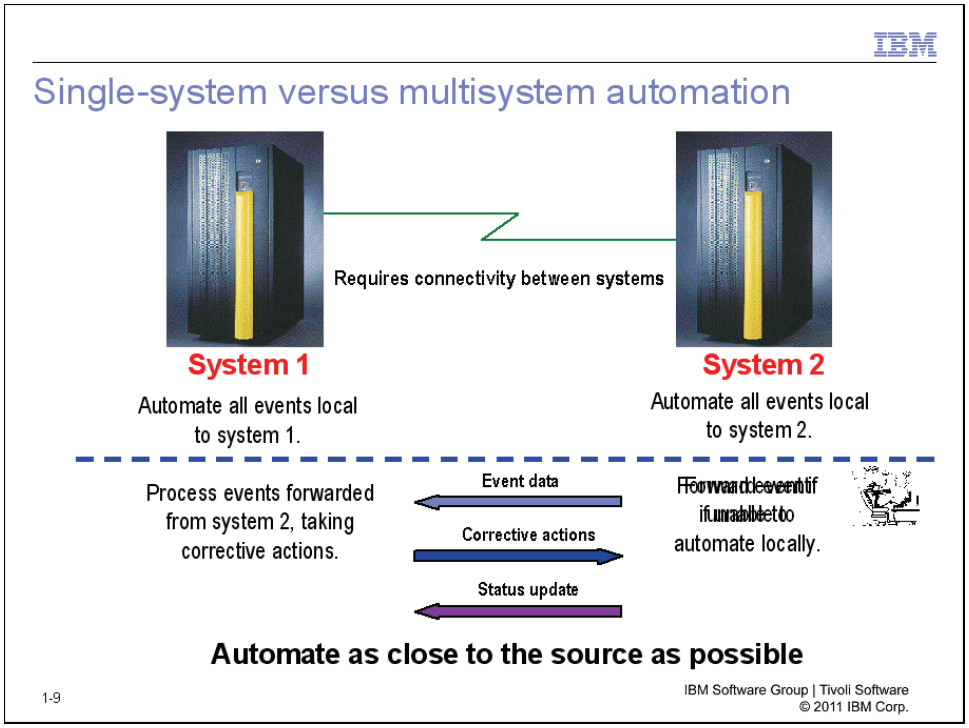
The third step in automation is consolidating commands into REXX EXECs. Suppose you have an application failure and your operations staff must issue the same sequence of commands each time. You can consolidate the sequence of commands into a REXX routine and have the operations staff run the REXX routine instead. Afterward, you can use the REXX EXECs and automate the recovery of the application when it fails instead of relying on the operations staff to manually perform the recovery.

You can create additional REXX EXECs to monitor critical resources by scheduling the EXECs with NetView timers. If resources fail, you can also automate a logical grouping that could include system and network resources.

Suppose an application fails, and you need to schedule a database archival to a workstation. You can coordinate the tasks required between the system, network, and distributed (workstation) resources to automate the recovery of the application as a logical entity. You can also coordinate the application recovery with an email notification to your shift supervisor.

NetView provides the infrastructure to accomplish all of your automation and also automation of your network resources. With SA z/OS, you can also automate your system and sysplex resources.

Single-system versus multisystem automation



After you complete automation of a single system, you can move to the next phase, which is multisystem automation. In multisystem automation, you use the single-system automation in each of the systems to automate the events for each system. This strategy is called automating as close to the source (of the event) as possible.

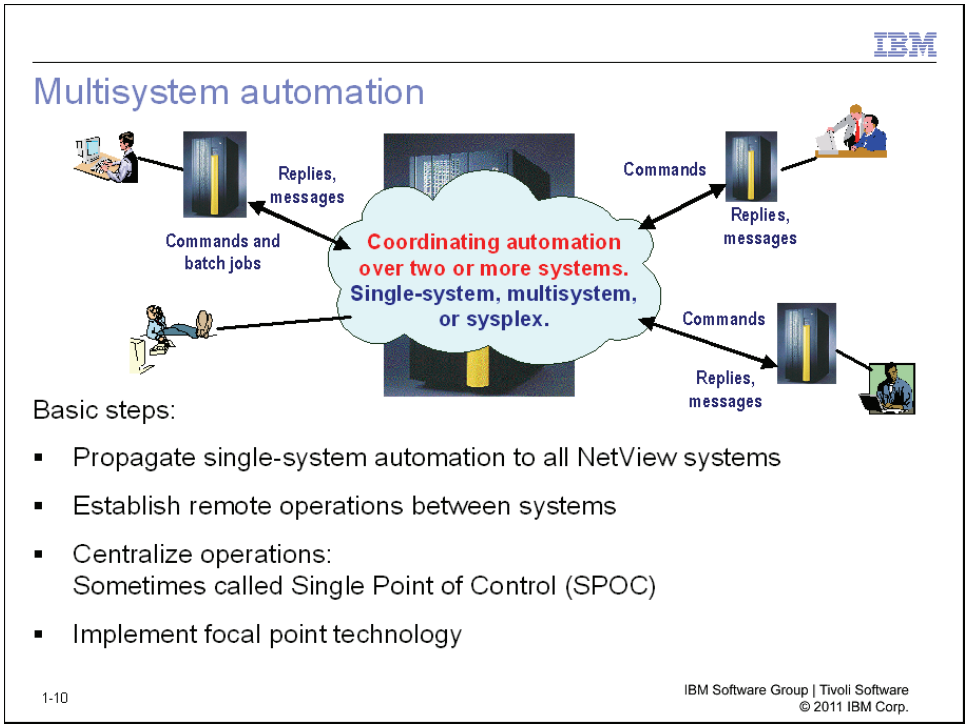
In multisystem automation, you designate one of the systems to be the focal point system. Each single system attempts to automate the events it receives. Some of the events might require that they be forwarded to the focal point for automation. The focal point processes the event and attempts to automate by sending commands to the remote system.

For multisystem automation, each of the systems must be connected to the focal point. For example, the connection could be a traditional VTAM CDRM, a TCP/IP connection, or an XCF connection if the systems are connected in a sysplex.

NetView-to-NetView connectivity must also be defined.

Note: Multisystem automation is not the same as sysplex automation. Sysplex automation occurs when you exploit the capabilities that a sysplex provides.

Multisystem automation



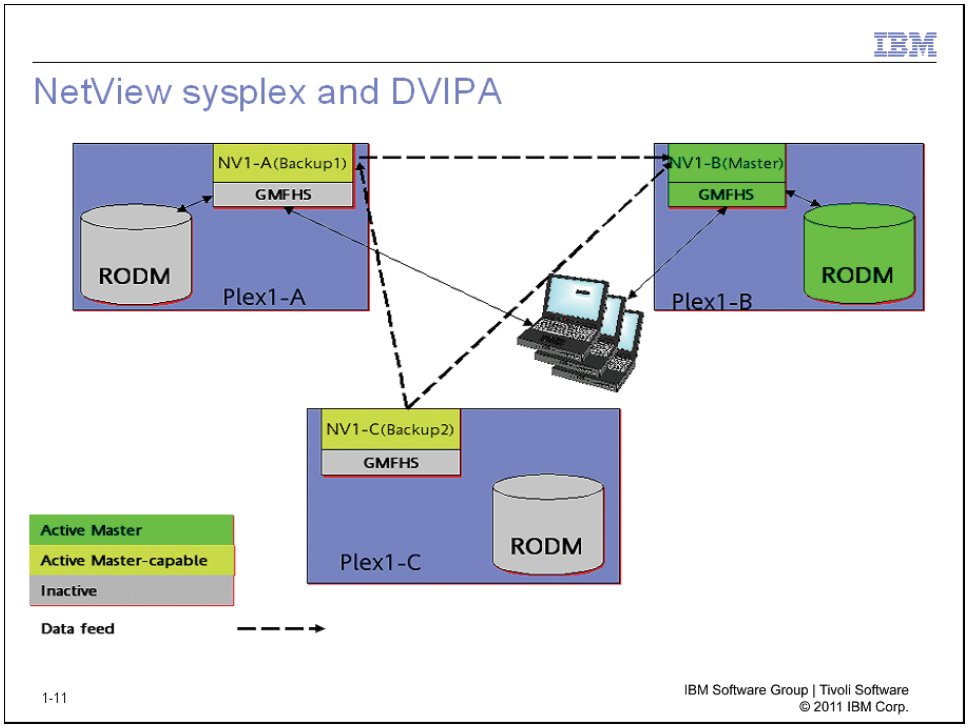
Multisystem automation, whether a single site or multiple sites, has a few additional steps to those discussed for single-system automation.

Multisystem automation involves establishing coordinated automation between two or more z/OS systems. The coordination enables the operation of resources that cannot be automated locally on a single system.

In many cases, the focus is to centralize operations, even when dealing with remote (distributed) host systems. In the case of no staff at the remote location, data pertaining to the remote systems is forwarded to the central site (focal point). The data could include problem reports that cannot be automated at the remote site.

In other cases, facilities with automation on one system can inquire about status of another system, forward information about other systems, initiate actions on other systems, and check the results of such actions. Multiple systems can be in different operational sites and function within different sysplexes.

NetView sysplex and DVIPA



The NetView program provides high availability sysplex management for complex system interactions. You can use it to manage and display information about your sysplex.

Automatic failover to another NetView program that can monitor the sysplex in case of outage is also provided. NetView can also monitor sysplex and system resources as follows:

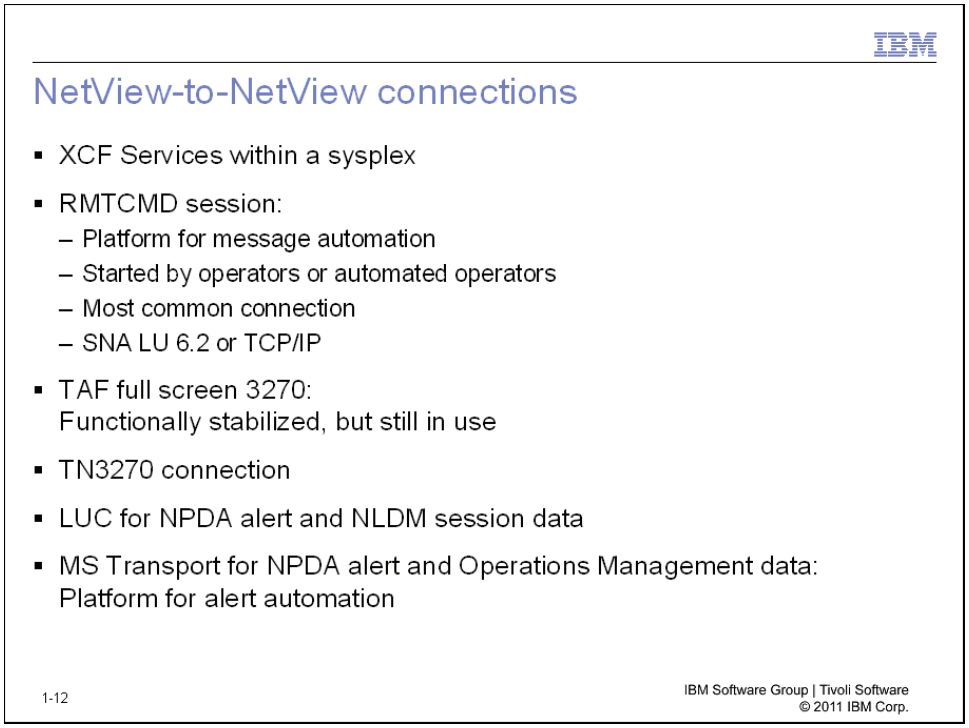
- Sysplexes
- Coupling facilities
- z/OS images
- TCP/IP stacks, IP interfaces
- Dynamic virtual IP addresses (DVIPAs)
- Telnet servers and ports
- Central processor complexes
- Logical partitions
- Open Systems Adapter (OSA)
- HiperSockets™ adapters

A master NetView program can provide management for systems outside of the sysplex as well as for another sysplex. This NetView program is known as an enterprise NetView program. Additional configuration is necessary for the enterprise NetView program to manage systems that are outside of the sysplex. DVIPA information is restricted to sysplex management.

A dynamic virtual IP address (DVIPA) can be defined for the master NetView program. This address is associated with the master NetView program, so that connection requests to this address always go to the master program. This address can be used by applications for contacting the master NetView program regardless of the system that it runs on within the sysplex.

XCF services can automatically provide information about other NetView programs within the sysplex. If an outage occurs, the NetView program is able to failover to another NetView program in case of an outage. By using XCF services, a master NetView program collects and processes data from other NetView programs within the sysplex to provide a single point of control.

NetView-to-NetView connections



By default, the NetView program participates in an XCF group that enables communication with other NetView programs in your sysplex environment for supporting discovery manager

Multisystem automation requires connectivity between your z/OS systems and your NetView domains. If you have NetView domains outside your sysplex, you can use RMTCMD to define the connections. The sessions can be over SNA LU6.2 or TCP/IP. In RMTCMD sessions, and you can connect to a target domain and issue commands.

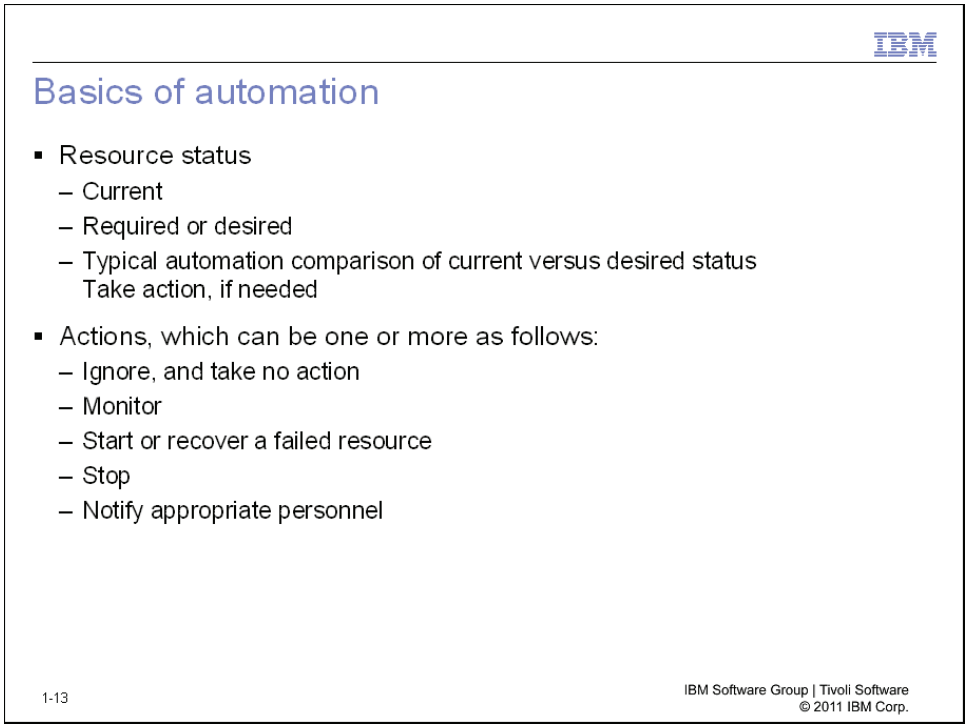
With the terminal access facility (TAF) component of NetView, you can connect to other z/OS applications, such as CICS, IMS, and NetView to receive messages and issue commands. TAF supports both line-mode and full-screen mode sessions.

Other types of connections that NetView uses include connections as follows:

- LUC sessions for NPDA and NLDM data
- MS Transport for alerts (MSUs) and Operations Management data

Note: SA z/OS uses XCF within the sysplex for communication between Automation Manager and Automation Agents. XCF is also used for communication between SA z/OS Automation Agents within the same XCF group. RMTCMD will be used between Automation Agents that are not members of the same XCF group.

Basics of automation



This slide shows the basic elements of any automation. Events that drive automation routines typically indicate the current status of the resource. In most cases, the automation routines also detect the desired status that defaults to *active*. In some cases, the desired status is for the resource to be down.The norm is for automation to attempt to activate or recover a resource.

Basic automation actions include:

- Take no action by ignoring the event.

  The event might be a result of a planned outage. Or the event might be a result of another outage already being automated.

- Monitor the resource.

  In some cases, the resource might recover by itself. If that occurs, do not use automation to attempt recovery. In other cases, you can monitor critical resources to prevent missed outages.

- Start (activate) the resource.

- Stop (deactivate) the resource.

- Recover from a failure.

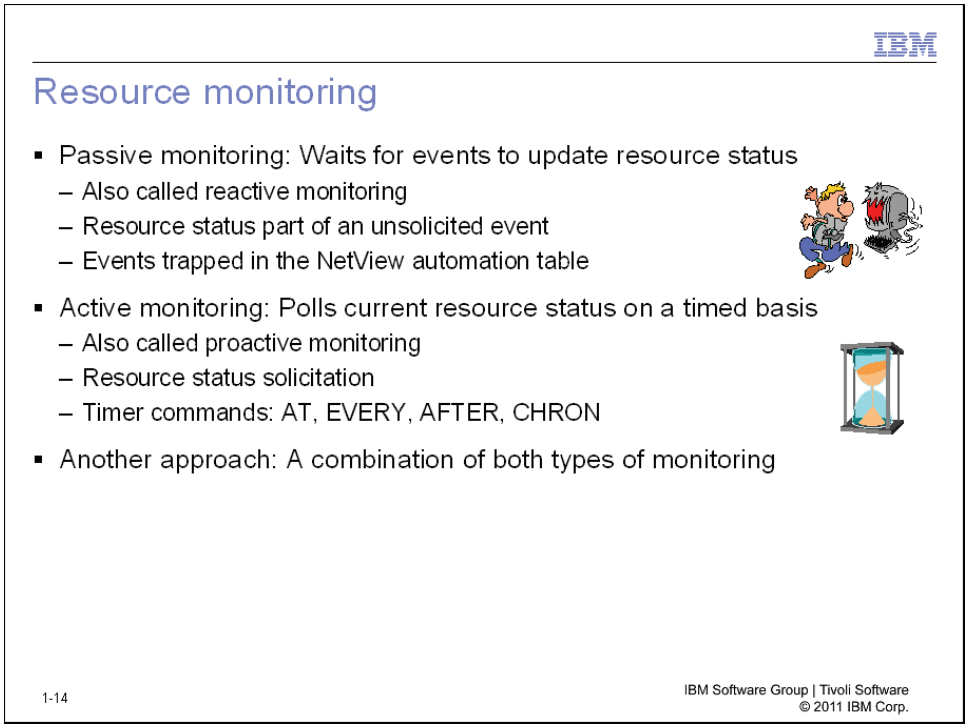
- Notify appropriate personnel. For example, send an email to another group in your shop or to a vendor. In the notification, inform them of an outage or the actions that automation took to recover from a failure.

NetView provides an INFORM policy for notifications. IBM Tivoli System Automation for Integrated Operations Management (SA IOM) also provides policy and functions for notifications.

Most automation products also provide additional features:

- Perform a *root cause analysis*, identifying if the current event is a symptom of another event.
- Perform *thresholding* to prevent automation from attempting resource recovery for a resource that fails continually. In that case, you have a problem that requires **operator intervention** before automation should be allowed to continue.
- Support *service periods* where you can define time intervals (for example, third shift) where it is acceptable for the resource to be down.

Resource monitoring

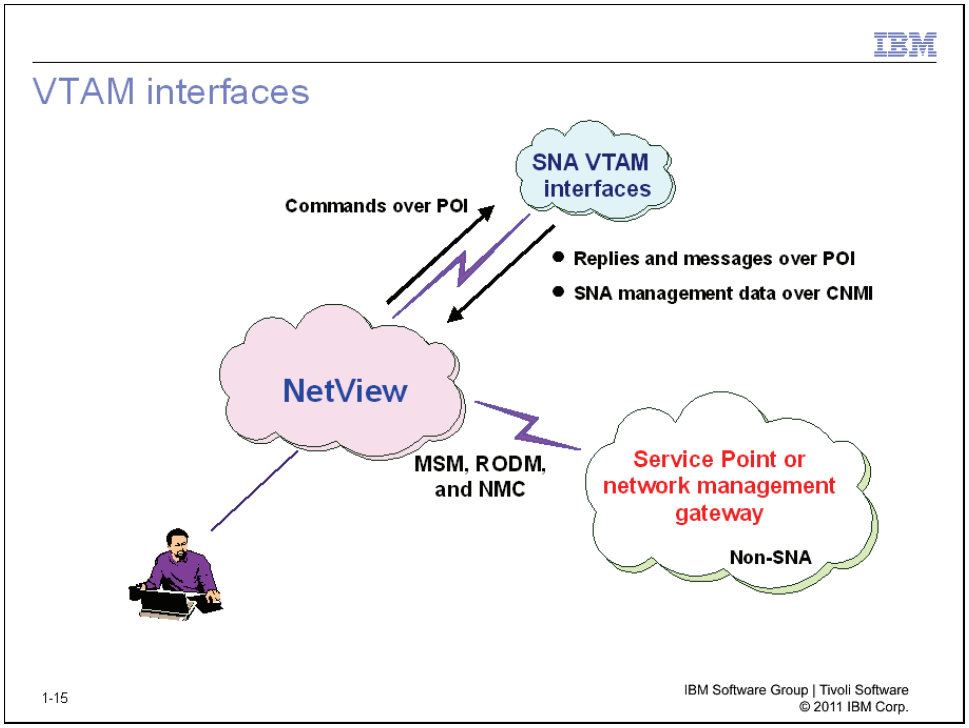


Up to now, this class has only discussed waiting for events to occur to drive automation. This is known as **passive monitoring**. This is also called *reactive monitoring* because the automation reacts to events as they are processed by the automation table. The events that drive passive monitoring are *unsolicited*. That is, they are the result of a resource failure and not related to any operator command or request.

You can monitor resource status by scheduling NetView timers to check on the status of one or more resources. This policy is known as **active monitoring**. It is also called *proactive monitoring* because the automation routines proactively check resource status instead of waiting for events. Active monitoring generates *solicited* events. That is, they are the result of a command that is issued to check the status of a resource.

Automation that performs both active and passive monitoring provides coverage because of polling for status and also reacting to events.

VTAM interfaces



NetView has access to SNA and non-SNA data. It communicates with SNA resources using two following VTAM interfaces:

- Programmed Operator Interface (POI) for messages and commands
- Communications Network Management Interface (CNMI) for SNA MSUs

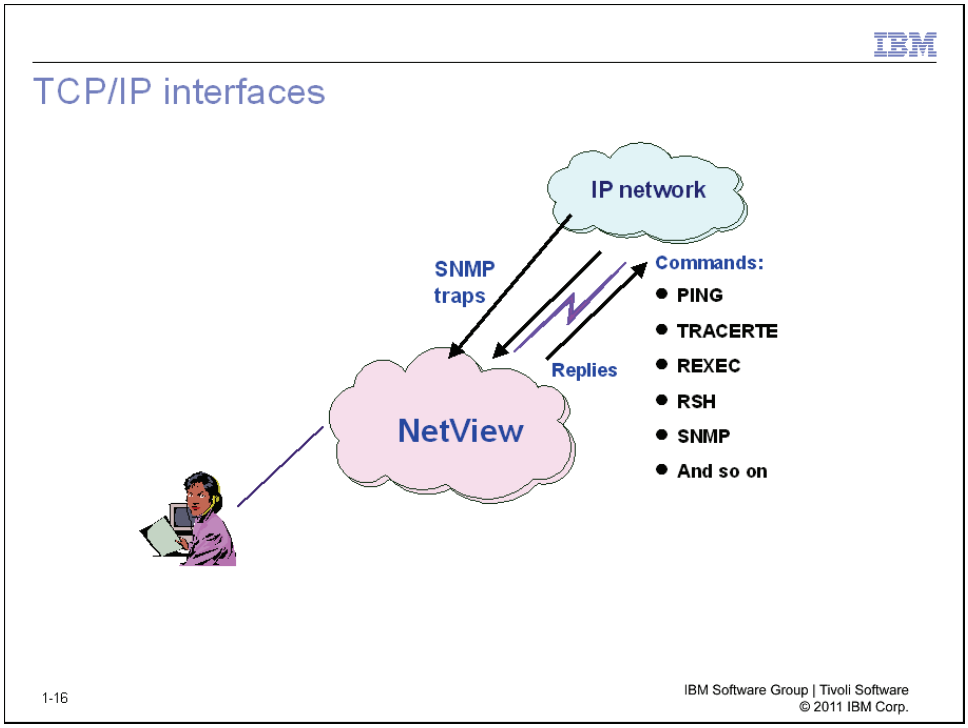
Note: If the NetView-VTAM POI connection is not active, z/OS uses MPF to handle VTAM messages. Discussion about MPF occurs later.

NetView traps both SNA data records, called Management Services Units (MSUs), and VTAM messages. It analyzes them and takes appropriate actions. NetView can automate actions pertaining to any SNA network resource.

For non-SNA networks, an IBM Service Point (called a network management gateway) provides access to the non-SNA resource data. The gateway converts the data to SNA MSUs, which are then transported to NetView, the focal point.

You use the MultiSystem Manager (MSM) component of NetView to obtain data from non-SNA environments. MSM acts as manager to the non-SNA networks that IBM Tivoli Network Manager or user-written Open Topology agent manage. Some agents can inform MSM of resource status changes. Others require MSM to poll for resource status.

TCP/IP interfaces



NetView also has access to TCP/IP network resource data. The Event/Automation Service (E/AS) is an optional address space that runs in the same z/OS system as NetView and the TCP/IP address space. Traps generated by TCP/IP devices can be automated in NetView as follows:

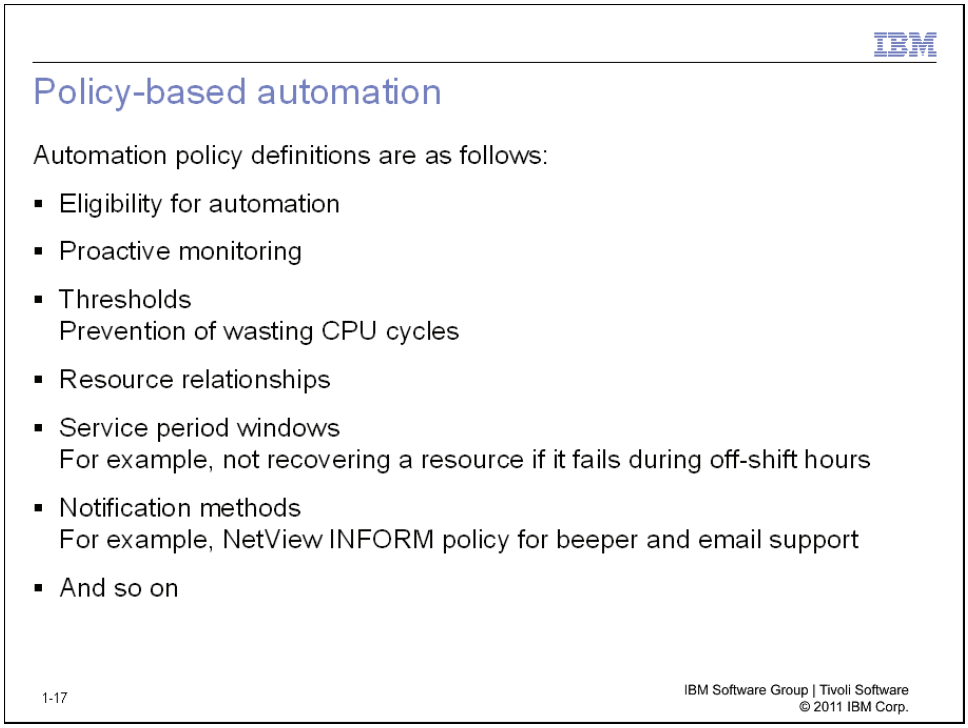
- E/AS can translate SNMP traps from the IP network to alerts and forward them to the NetView automation table.
- SNMP traps can arrive directly in the automation table without E/AS.

Proactive monitoring with the AON/TCP component of NetView is also possible.

You can take following actions against TCP/IP resources:

- Use TN3270 to log in to a remote system.
- Issue any Unix System Services command.
- Issue a PING or TRACERTE command to test the connectivity to the node.
- Issue an SNMP command to retrieve or set MIB values, which can assist with management of SNMP-supported resources.
- Issue a REXEC or RSH command to run UNIX commands at an IP node that uses differing levels of security.
- Issue the RMTCMD command to send system, subsystem, and network commands to a remote NetView host for processing.
- Issue an IPCMD command to issue a TCP/IP command directly to the local TCP/IP stack.
- Issue an IPLOG command to log a message at the IP node.
- Issue a TCPCONN command to display detailed TCP connection data.
- Issue a PKTS command to retrieve TCP/IP packet trace data for debugging purposes.
- Issue the SOCKET command to retrieve information about TCP/IP stacks or to run stack services. You can use it in TCP/IP applications that are based on the NetView program.

Policy-based automation



With policy-based automation, you can define parameters that pertain to the automation separately from the automation itself. When an event occurs, the automation routines consult the policy. The following determinations might occur:

- Automation
- Thresholds
- Notification recipients about the failure
- Method of notification
- Possible additional conditions

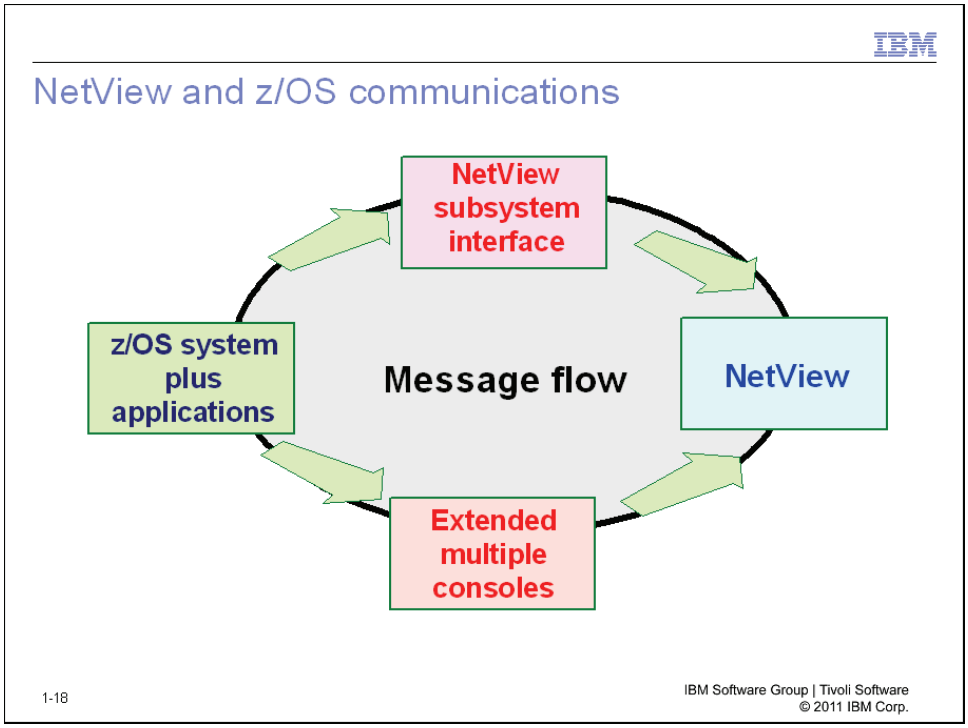
Notifications can use NetView INFORM policy, SA IOM, or other products.

Automation concepts apply to both system and network resources. Automation policy definitions and implementation can differ. For example, SA z/OS automation policy is defined using ISPF-based customization dialogs whereas AON automation policy is defined using flat files located in DSIPARM.

Note: Several NetView components use policy files located in DSIPARM and controlled by **POLICY.** statements in CNMSTYLE. For example, the sysplex topology function detects

TCP/IP stacks and updates stack policy definitions. The stack policy is then used by components, such as the Tivoli Enterprise Portal, web applications, automation, and so on.

NetView and z/OS communications



NetView is an application that runs directly in its own address space under the z/OS operating system. NetView is defined as a z/OS subsystem.

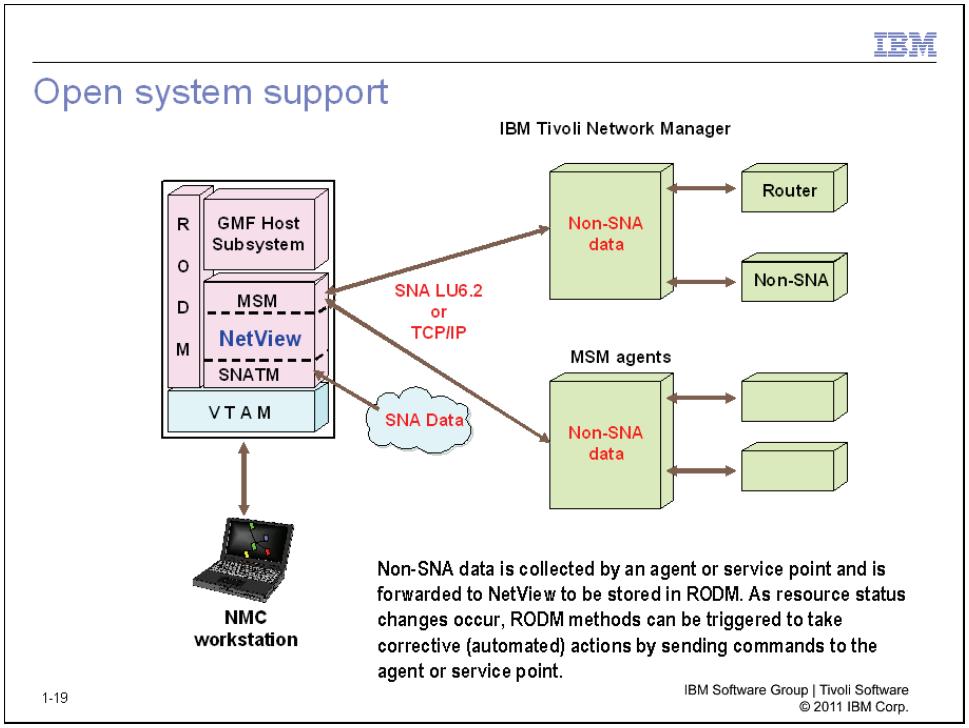
Communication with the z/OS operating system requires a second address space that buffers data to and from z/OS. This address space, called the NetView subsystem interface (SSI), is the direct interface to the z/OS subsystem interface. You can use the SSI to transfer commands and messages between z/OS and the NetView application.

As shipped, the NetView program uses extended multiple console support (EMCS) consoles for tasks that must issue MVS system operator commands. EMCS consoles are dynamically defined and not restricted to a maximum of 99 consoles.

EMCS consoles are used for sending commands from the NetView program to the MVS operating system and receiving messages from MVS. The console name must be unique for each MVS or each sysplex. To avoid console name conflicts, you can use the NetView program to assign a permanent, unique console for each operator by the ConsMask statement in CNMSTYLE.

These interfaces permit NetView access to messages that arrive from the z/OS system tasks and also the z/OS applications.

Open system support



Open systems support requires that any type of network can be managed and automated. NetView can receive and process data that are needed for management and automation of *open systems*. It uses components as follows:

- Resource Object Data Manager (RODM)

  RODM is a high-speed data cache that stores the current status of any resource within the complex in data spaces. Agents, such as those that MSM supplies, collect the necessary data and forward it to NetView to be stored in RODM. RODM methods can be triggered to act upon this data.

- MultiSystem Manager (MSM)

  MSM enables NetView to act as a manager to deal with IBM Tivoli Network Manager and user-written Open Network Topology agent in the enterprise. Data that MSG retrieves is stored in RODM.

- NetView Management Console (NMC)

  NMC is a graphical interface that displays configuration (topology) and status information that RODM stores on a Java-based workstation.

- Graphic Monitor Facility Host Subsystem (GMFHS)

  GMFHS provides communication between NMC and RODM.

- SNA Topology Manager (SNATM)

  SNATM collects SNA subarea and APPN data and stores it in RODM.

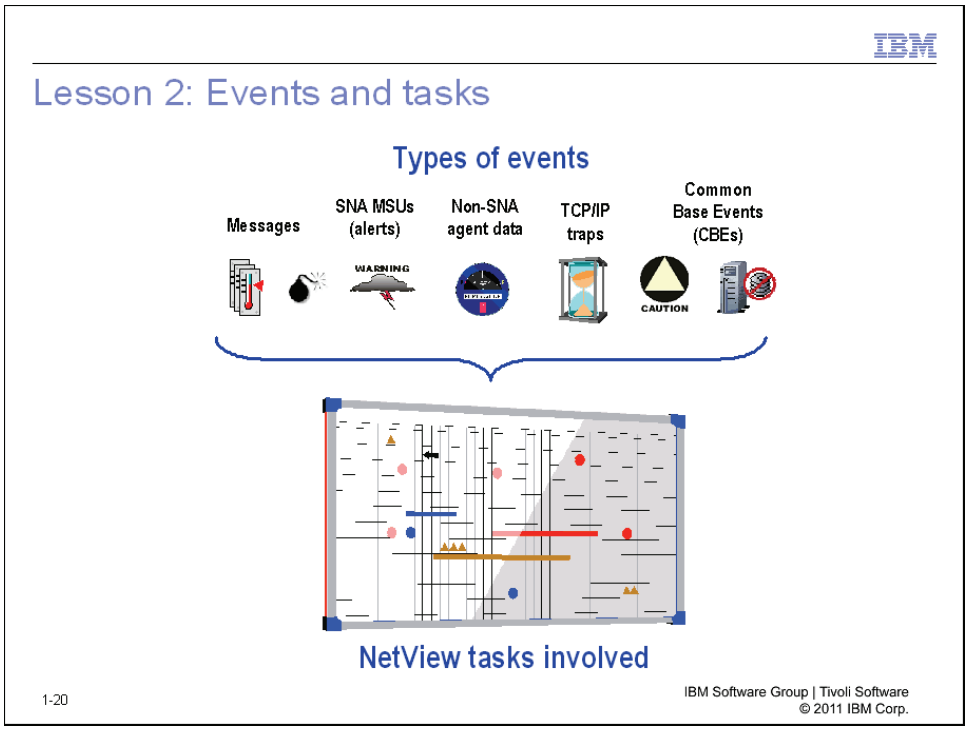
NetView can also forward data to other Tivoli products, such as Tivoli Netcool®/OMNIbus program, IBM Tivoli Enterprise Console, and IBM Tivoli Business Service Manager.

An agent is an application running on a network device that communicates with NetView over SNA LU 6.2 or TCP/IP at the mainframe host. The agent can send data (for example, status or topology data) to NetView and receive commands from NetView.

When RODM receives updates concerning the status of a resource, NetView can act on that information and automate responses. Two ways of triggering NetView automation are as follows:

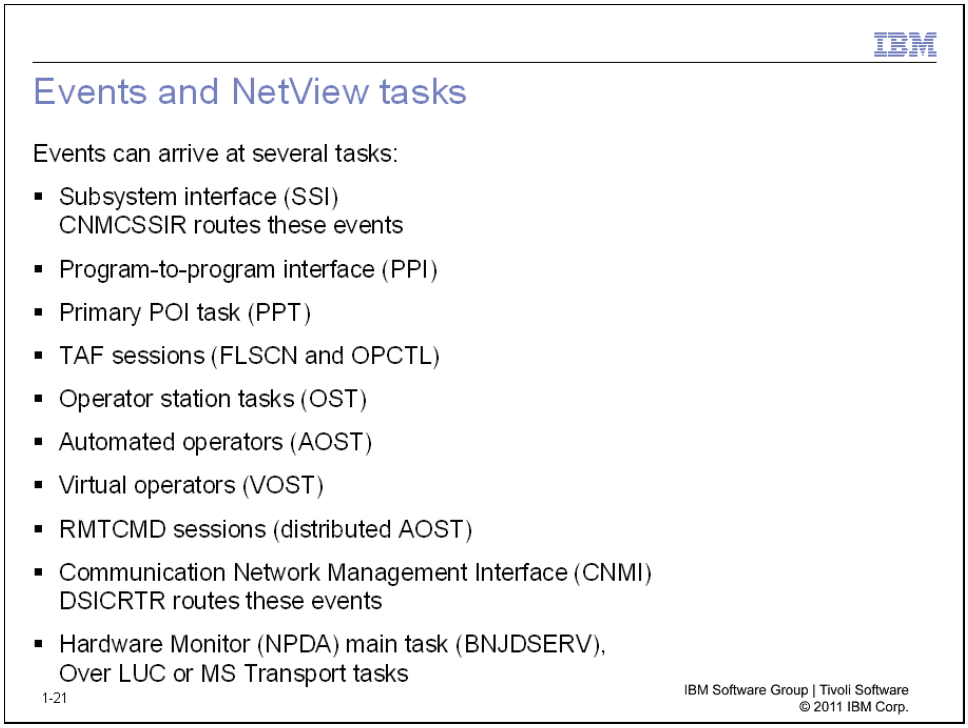
- An alert created for processing by the automation table
- A RODM method (program) invoked automatically

Lesson 2: Events and tasks



This lesson discusses the types of events that NetView can process and the tasks that are required. Each event translates into a message or an alert for processing by the automation table.

Events and NetView tasks

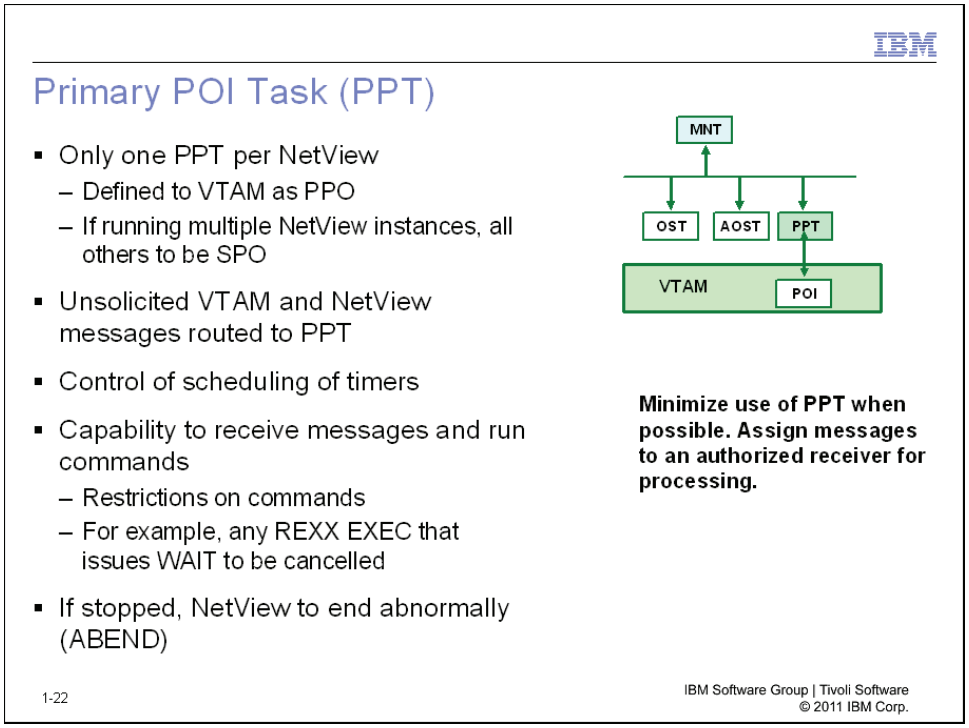


Events can arrive in NetView from several sources as this slide lists. Examples are as follows:

- z/OS messages coming across the SSI interface
- Applications sending and receiving data using queues within the PPI
- Unsolicited VTAM messages coming in through the PPT task
- Messages from remote systems arriving through RMTMCD sessions

This list is a small subset of the NetView tasks. This class discusses NetView tasks that pertain to automation. Issue a **LIST STATUS=TASKS** command to display all tasks that are defined to NetView.

Primary POI Task (PPT)



The Primary Programmed Operator Interface (POI) Task, otherwise known as PPT, processes unsolicited VTAM and a subset of NetView messages. If the system administrator who creates your automation table statements does not supply message routing logic, the PPT designates the recipient operator. If no operator is found, the message routes to the system console.

Each NetView has one PPT task. The PPT is always active and remains active even if VTAM terminates. Termination of the PPT task abends NetView.

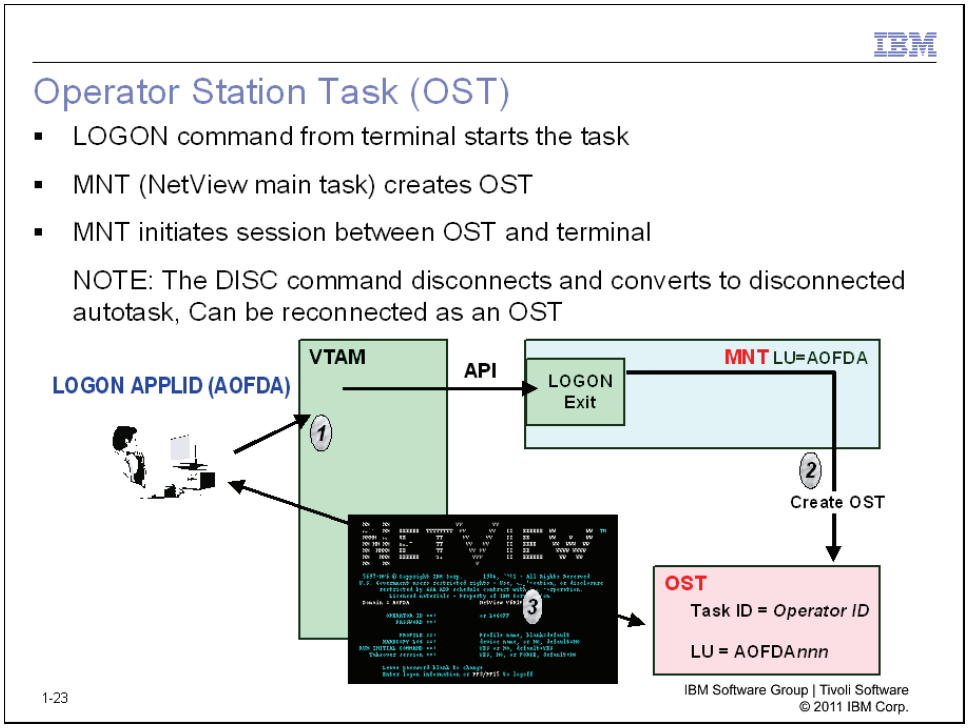
Because the PPT must always be present, it can be used for running certain commands that must always run. An example of such a command is one for controlling the scheduling of timers)

The PPT can receive and process messages, run commands and EXECs, and run timer-driven requests. The PPT plays a major role in the routing of VTAM messages. It cannot directly display messages, but can perform some automation functions, such as running EXECs

Important: .Be careful of the work that you schedule under the PPT task:

- Not all functions are supported under the PPT. For example, the PPT does not support waiting or trapping messages.
- Schedule automation to run under other tasks, such as automation tasks to keep the PPT task free to process messages and timers. Such strategy can help you avoid possible performance problems.

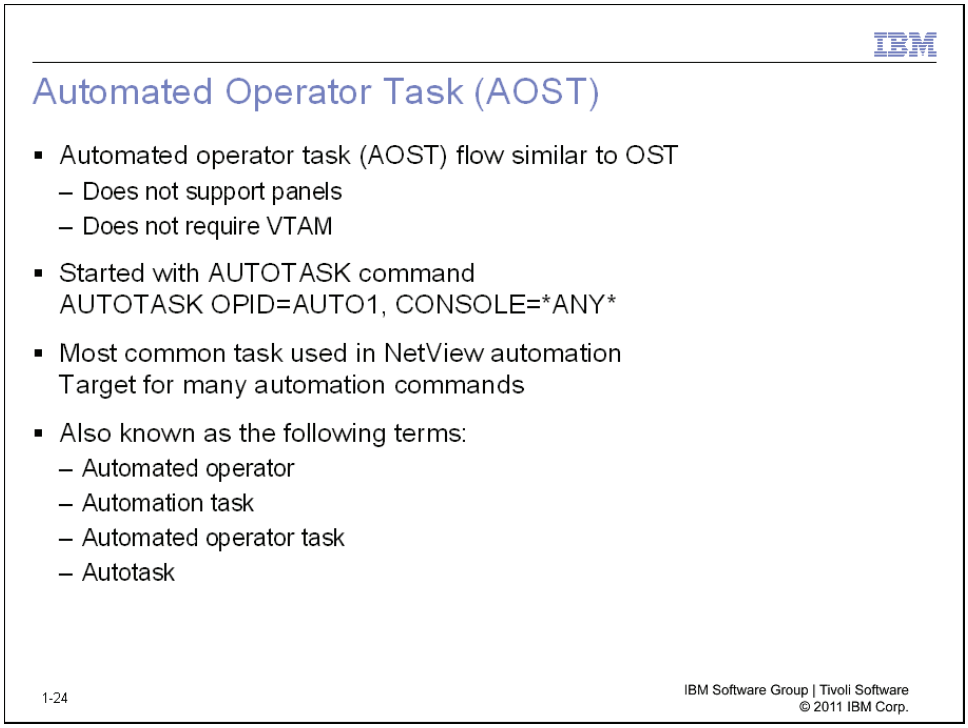
Operator Station Task (OST)



The NetView main task, DSIMNT, creates an Operator Station Task (OST) for each operator who logs on to NetView. An OST is an *attended* task. In VTAM terms, the OST pertains to the application (PLU) being in session with the terminal (SLU). If several operators are logged on, each operator can issue commands that run independently of each other. The execution is managed by the multitasking capabilities of z/OS.

- OST routines check operator authorization at logon and analyze commands that invoke appropriate processors. OSTs end at operator logoff or when NetView or VTAM terminates.
- The DISC command converts an OST into a *disconnected* autotask, which can be later reconnected or taken over as an OST again.
- OSTs can receive and display messages, run commands, run timer-driven requests; and run full-screen commands. All NetView commands must run within an OST.

Automated Operator Task (AOST)



Automation tasks (Autotasks) are OSTs that are used for *unattended* operations. An autotask is an OST that does not have a logged-on operator directly associated with it. An autotask has similar characteristics to an OST. It can receive (but not display) messages. It can run commands. It can run timer-driven commands. It cannot execute full-screen commands such as HELP, NLDM, or NPDA.

Autotasks can start during NetView initialization (using CNMSTYLE) or with the AUTOTASK command. An example is as follows:

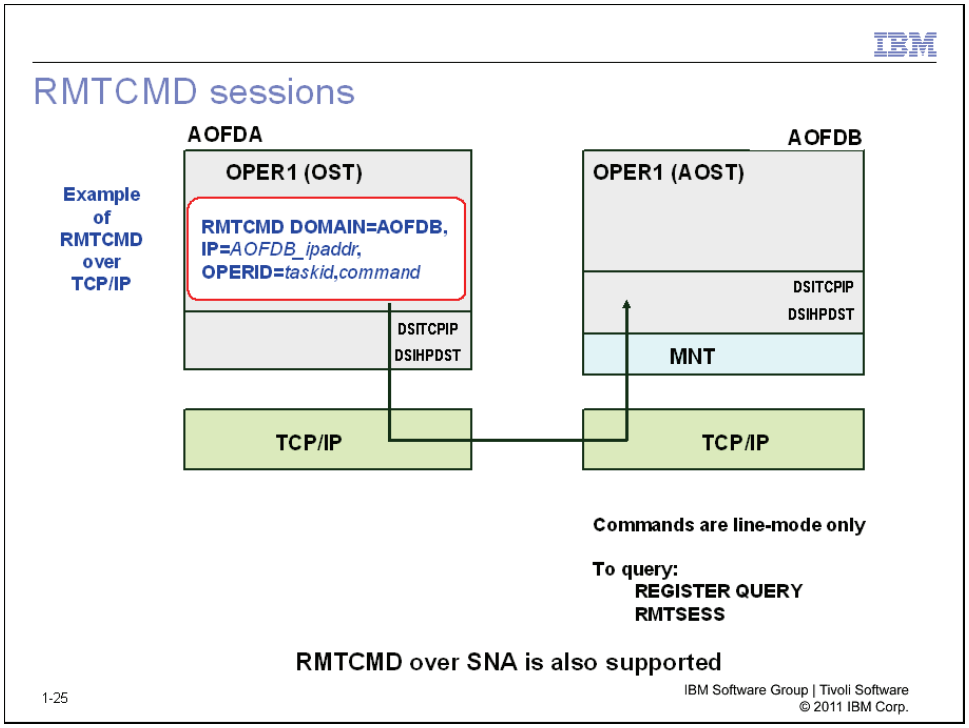
```
AUTOTASK OPID=AUTO1 CONSOLE=*ANY*
```

An autotask can start without requiring VTAM to be active. It remains active even after VTAM termination. Autotasks can start after a NetView CLOSE command has been issued, usable for automating shutdown. After VTAM is active, the autotask uses VTAM access control block (ACB). You define your VTAM APPL definitions to accommodate operators and autotasks.

If NetView commands are entered from an EMCS console, use the optional CONSOLE operand to indicate that those commands are to run in this autotask. (All commands or EXECs must execute in an OST task.) Specify the console name. Use console names *MASTER* and *ANY*. If a console does not have an associated autotask, NetView commands cannot be entered from it.

For more details on console names, see the online help for the AUTOTASK command or the *IBM Tivoli NetView for z/OS Administration Reference* manual.

RMTCMD sessions



Use the RMTCMD command to send a command that is to run in another NetView domain. The originating domain receives responses to the command.

The remote domain might be connected by SNA LU 6.2 or TCP/IP (IPv4 or IPv6). With RMTCMD, you can send more data than by using the older NNT support.

If no remote task is in the RMTCMD command, an existing session to that remote domain is used. If none exists, a *distributed autotask* starts in the remote domain. The remote domain uses the same name as the originating operator, and this session becomes an owned session.

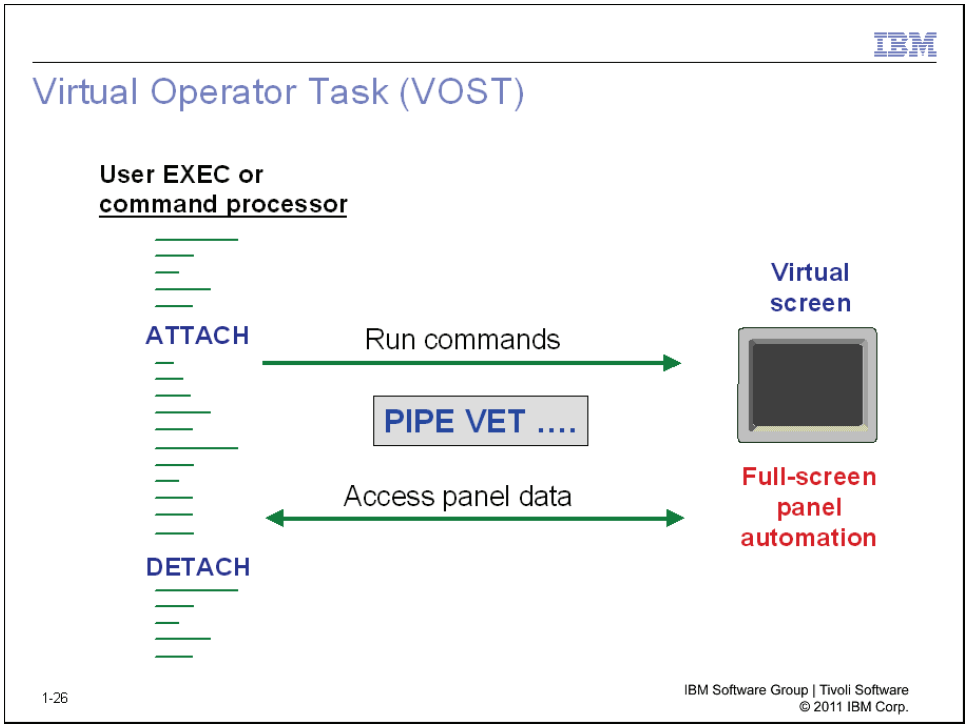
If a remote task is in the RMTCMD command, and the task already exists on the remote domain, the command routes to the remote autotask. The session does not become an owned session. If the desired task does not exist, a *distributed autotask* with the desired name starts in the remote domain, and this session becomes an owned session.

Security can be enhanced when using the RMTCMD command. (See LIST DSIUDST command and the *NetView Security Reference* manual).

The RMTSESS command reports existing RMTCMD owned sessions. The REGISTER QUERY command reports the state of owned RMTCMD sessions. You can end owned sessions by issuing the ENDTASK command.

Always use the RMTCMD command if using EMCS consoles, not NNT connections. The full set of message characteristics return with each message. Commands that display a panel are not supported.

Virtual Operator Task (VOST)



Use a Virtual Operator Station Task (VOST) to automate full-screen 3270 sessions, sometimes called *screen scraping* automation. A VOST is an *unattended* task.

Create a VOST by issuing an ATTACH command (usually from an EXEC). The VOST associates with the task (OST or AOST) that creates it.

The VOST logs on to 3270 applications, usually through a TAF session. The VOST maintains a virtual screen that reflects the current state of the session. This virtual screen can be made available to the creating task. The creating task can enter data on the virtual screen and request that the ENTER and PF keys be pressed. The creating task can also request the current contents of the screen to analyze the results of the commands entered.

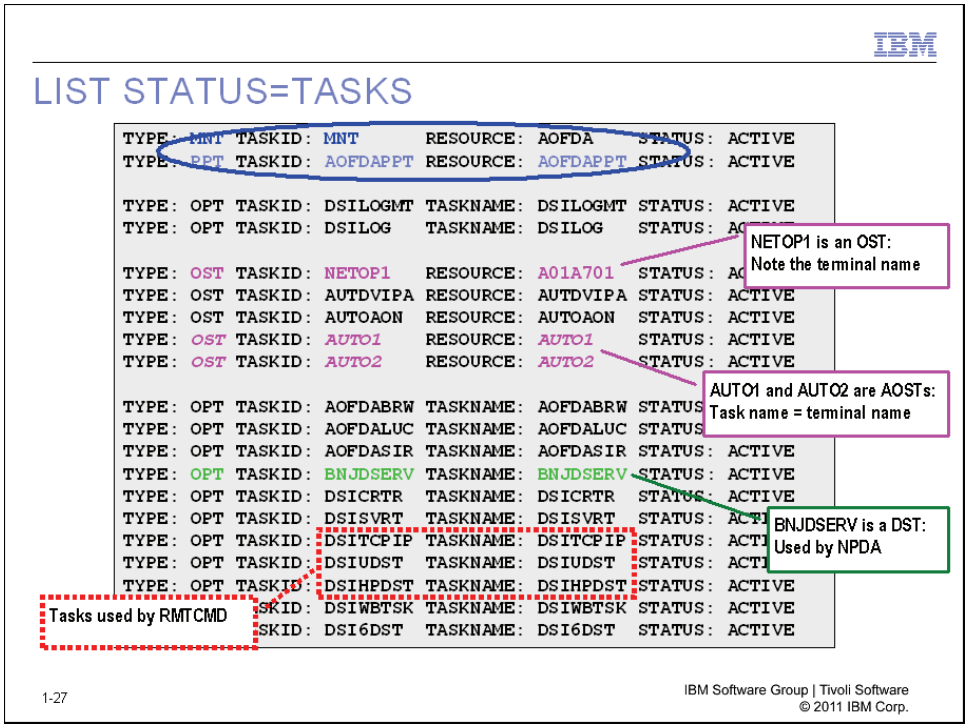
The DETACH command releases the VOST. Whether the VOST ends at OST logoff or not depends on how the VOST is invoked.

VOST tasks can be displayed with the following command:

```
LIST STATUS=VOST
```

See the *Programming: PIPES* manual for further details.

LIST STATUS=TASKS



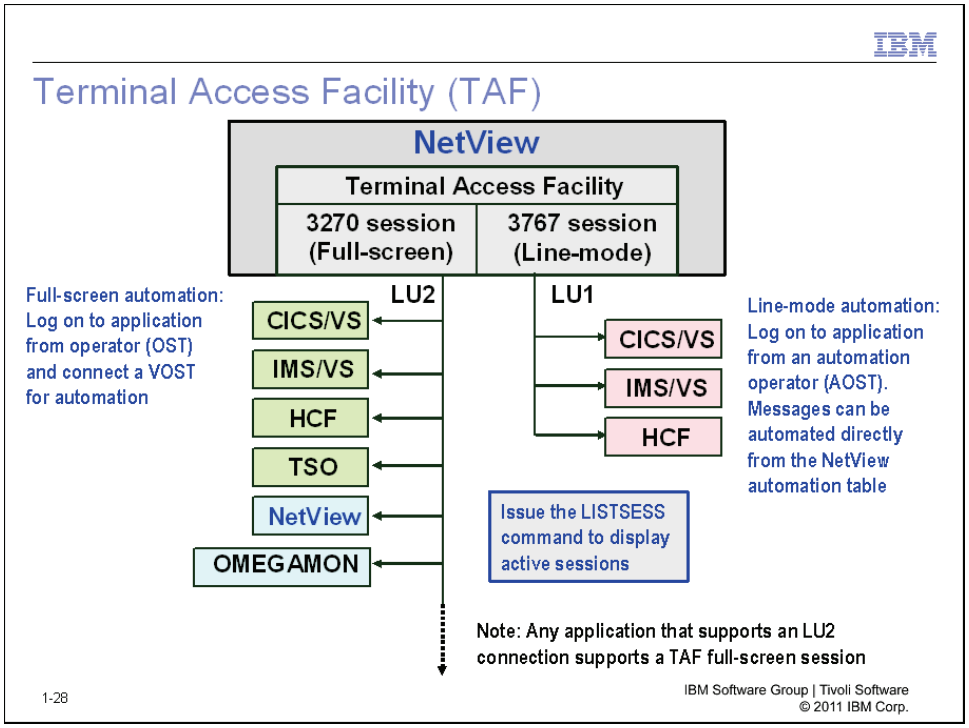
This slide shows an example of a LIST STATUS=TASKS command to display all tasks that are currently defined to NetView. The resource name for the main task indicates that the command was entered from NetView AOFDA domain. The PPT task name is the NetView domain ID concatenated with the PPT character string. In this case, it is AOFDAPPT.

Several tasks as follows are active on AOFDA domain:

- NETOP1 is an OST logged on from the A01A701 terminal.
- AUTO1 is an automation task (AOST). You can determine that because the resource (terminal) name is the same as the task name.
- AUTO2 is also an automation task.
- NPDA is active. BNJDSERV is listed as one of the optional tasks (OPT).
- DSITCPIP, DSIUDST, and DSIHPDST tasks are active. This means that the AOFDA domain is enabled for RMTCMD sessions over TCP/IP and SNA.

Note: LIST STATUS=TASKS might also display tasks that are defined to NetView but not currently active. The example shows only active tasks.

Terminal Access Facility



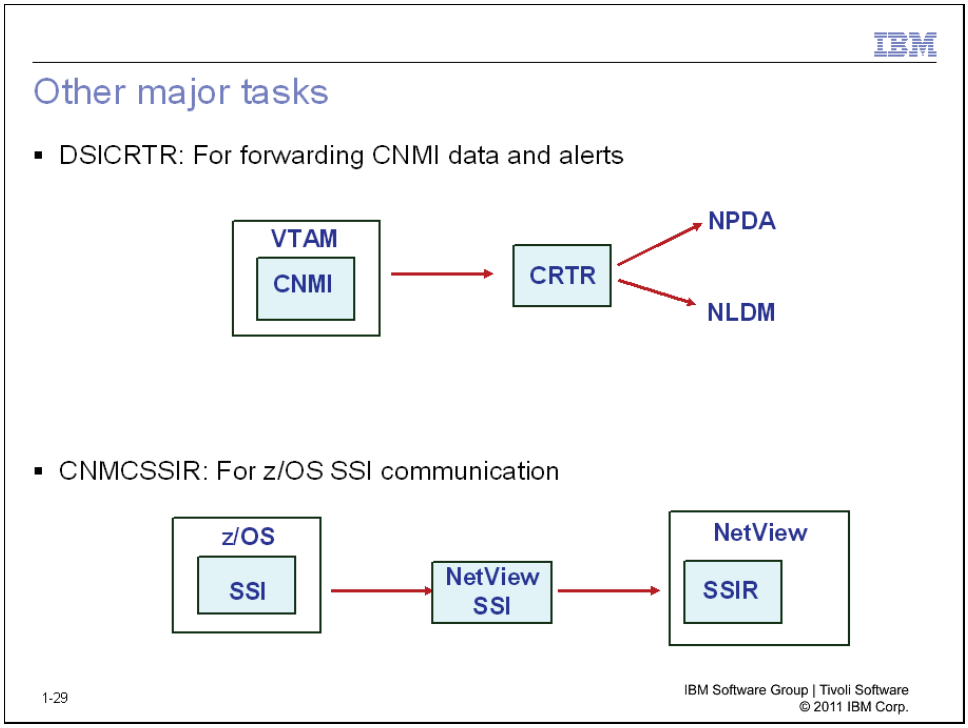
The Terminal Access Facility (TAF) of NetView simulates a terminal logon to a chosen VTAM application. The NetView operator is then logged on to both NetView and target application simultaneously. A target session might run in line-by-line mode (LU1 session type, simulating a 3767 terminal), or in full screen mode (LU2 session type, simulating a 3270 terminal). You can use a TAF to log on to any VTAM application that supports LU1 or LU2 sessions, including a remote NetView domain.

A TAF full-screen (FLSCN, 3270 LU2) session cannot be used for message automation. A TAF operator-control mode (OPCTL, 3767 LU1) session uses line-by-line mode. This type of session supports message exchange between NetView and the target application providing a mechanism for automation of messages from the application.

An autotask can use the TAF operator-control mechanism to manage CICS/ VS and IMS/VS sessions. You can also use the NetView PPI to manage CICS and IMS.

Tip: Use a VOST for full-screen automation of TAF LU2 sessions.

Other major tasks



This slide showsthe Communication Network Management Router Task (DSICRTR) and Subsystem Interface Router Task (CNMCSSIR)

The CRTR handles data and alerts that arrive from the VTAM CNMI. The CRTR forwards CMN data from VTAM to tasks, such as NPDA and NLDM, as necessary. Alerts are sent to the Hardware Monitor database.

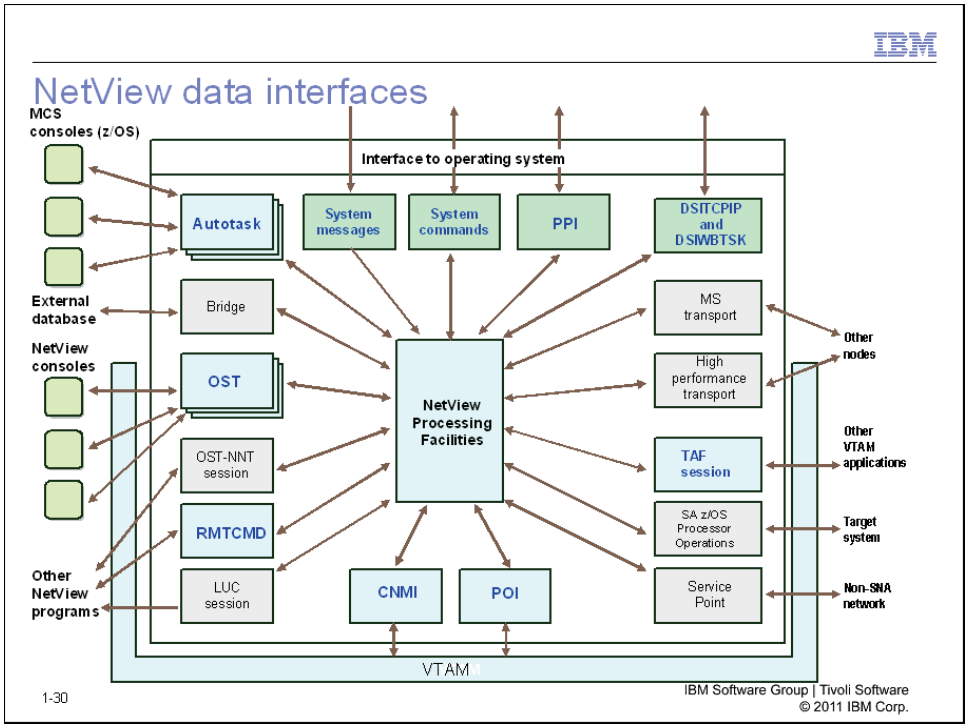
The CRTR task is optional. A NetView application can have only one CRTR task. If several NetViews are active within a system, each can have an active CRTR task. Only one can receive CNM data from VTAM, but each can be used for processing alerts.

The CRTR task is defined in CNMSTYLE and requires a corresponding VTAM APPL definition in order to forward CNMI data.

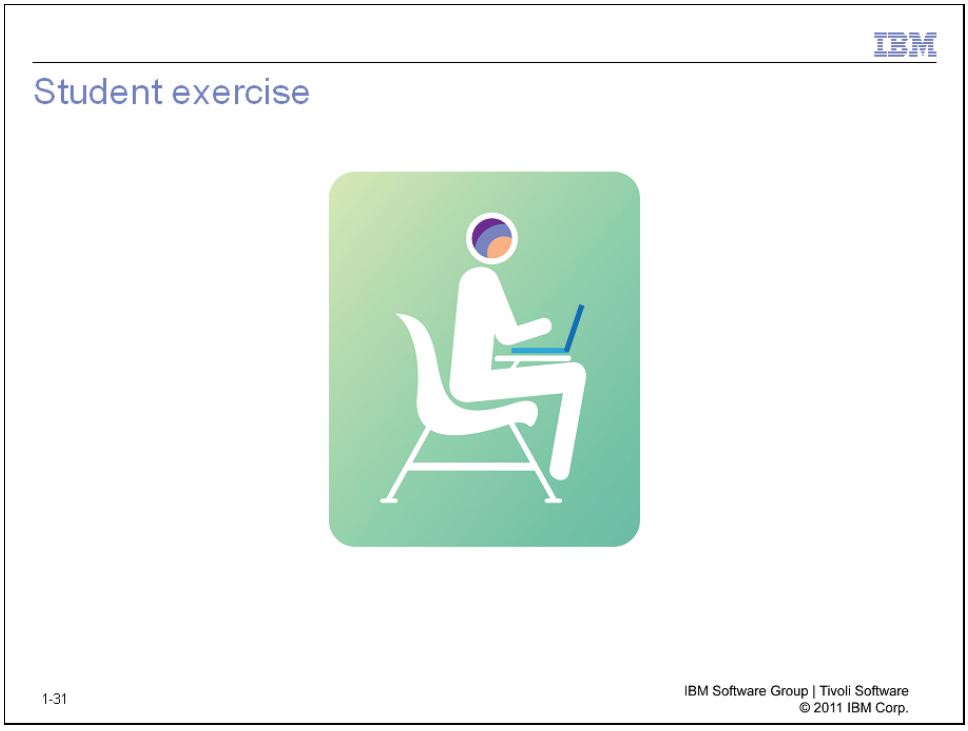
The SSIR task receives the flow of unsolicited messages from z/OS. NetView can be set up so that this flow comes through the z/OS SSI using the NetView subsystem address space. The flow comes directly from z/OS that uses the EMCS feature of z/OS. In effect, the SSIR task is defined to z/OS MCS as a console that receives messages with all routing codes.

Only one SSIR task is in each NetView application address space. The SSIR also handles commands that are entered at an EMCS console by using a command prefix. The SSIR then routes these commands to the autotask that is associated with that console (if any) for actual execution. Any output from such commands routes back to the originating EMCS console.

NetView data interfaces

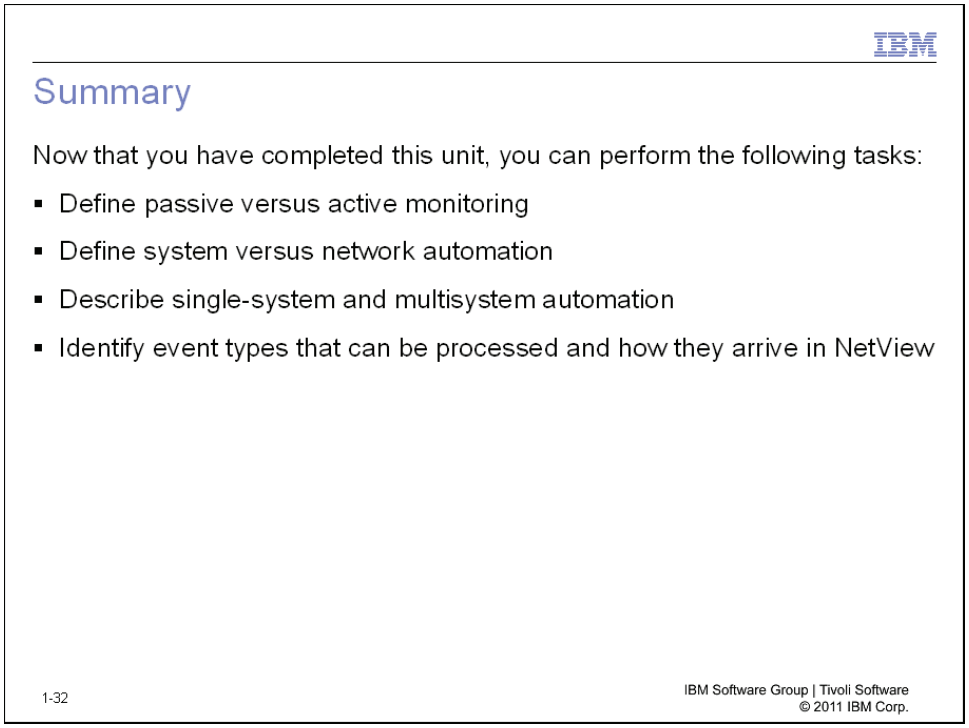


This slide summarizes several interfaces that can present data to NetView. See the *Automation Guide* manual for full descriptions of these interfaces.

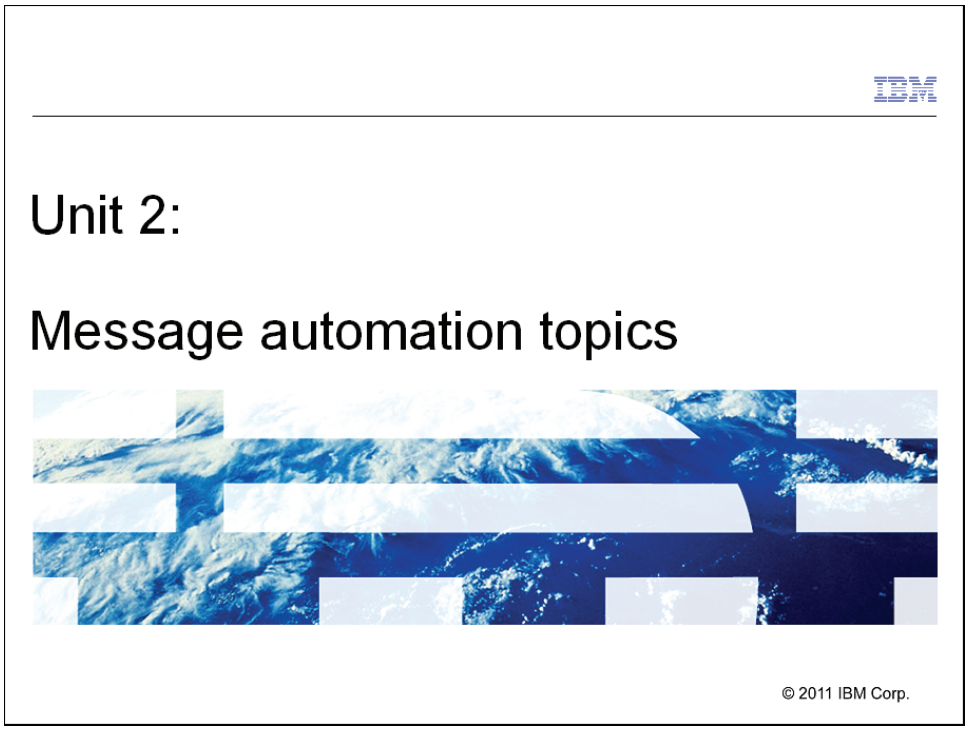


Open your Student Exercises book and perform the exercises for this unit.

Summary



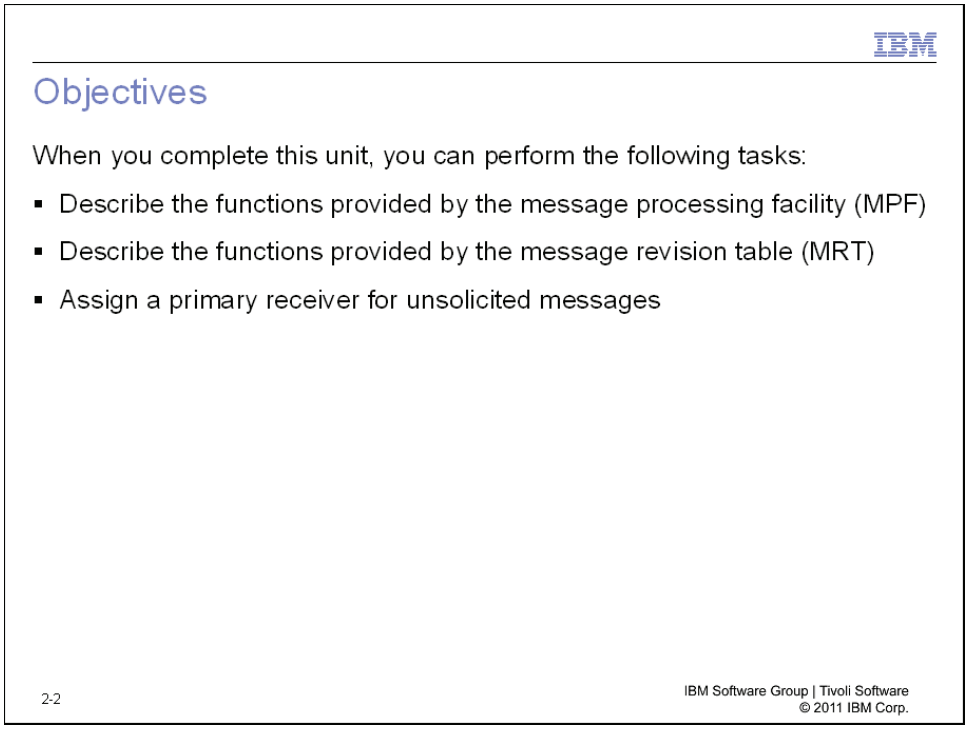
Unit 2: Message automation topics



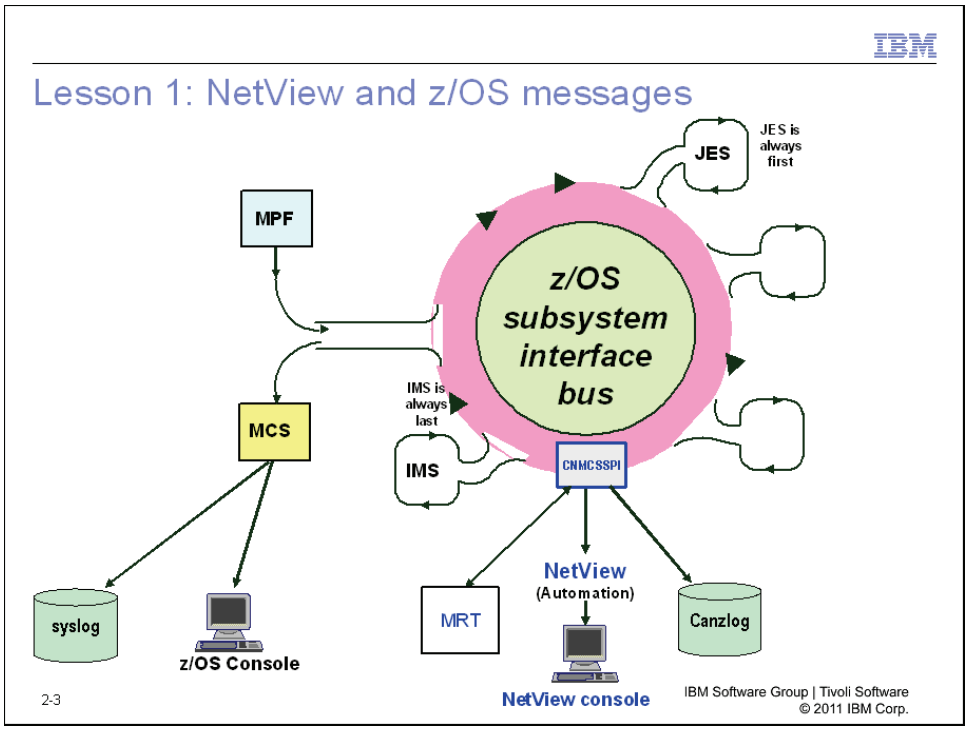
Introduction

This unit discusses the automation of messages with NetView. For example, using the message processing facility (MPF) of z/OS, you can suppress messages. The message revision table (MRT) of NetView provides functions similar to the MPF. This unit also provides information about the *primary receiver* for messages and how it pertains to message automation.

Objectives



Lesson 1: NetView and z/OS messages



NetView is the base component for mainframe-based management and automation of system and network resources in your enterprise.

Messages can arrive from operating systems, their applications and subsystems, and also from the network. NetView analyzes messages, which pertain to the status of tasks and resources, then it takes actions as assigned by the installation.

When implementing an automation project, it is necessary to understand how NetView receives the relevant messages. You must also know how to work with the appropriate NetView facility to achieve the desired automation process. The networking connections where messages flow vary, requiring careful implementation and operation.

The slide shows one possible flow for receiving z/OS messages, the *z/OS subsystem interface bus*. Messages flow through MPF, then to the SSI, where alterations can be made that affect their routing and presentation. As part of SSI processing, messages are written to the NetView Canzlog data space. After the SSI, messages are normally routed through MCS to one or more consoles. NetView is one of many application on the SSI bus. JES is usually the first application on the SSI bus, and IMS is usually the last. The order of the remaining applications is based on the order of their subsystem names that are defined in PARMLIB member, IEFSSN*xx*.

The NetView listener is named CNMCSSPI, actually in the LPA, but logically, in the SSI bus.

If your installation produces large volumes of messages for operators to monitor, use the action message retention facility (AMRF). If you want operators to be able to retrieve action messages and WTOR messages that are no longer on the console, use AMRF. AMRF keeps action messages so that the operator can view them later. WTOR messages are always available for operator retrieval regardless of the state of AMRF.

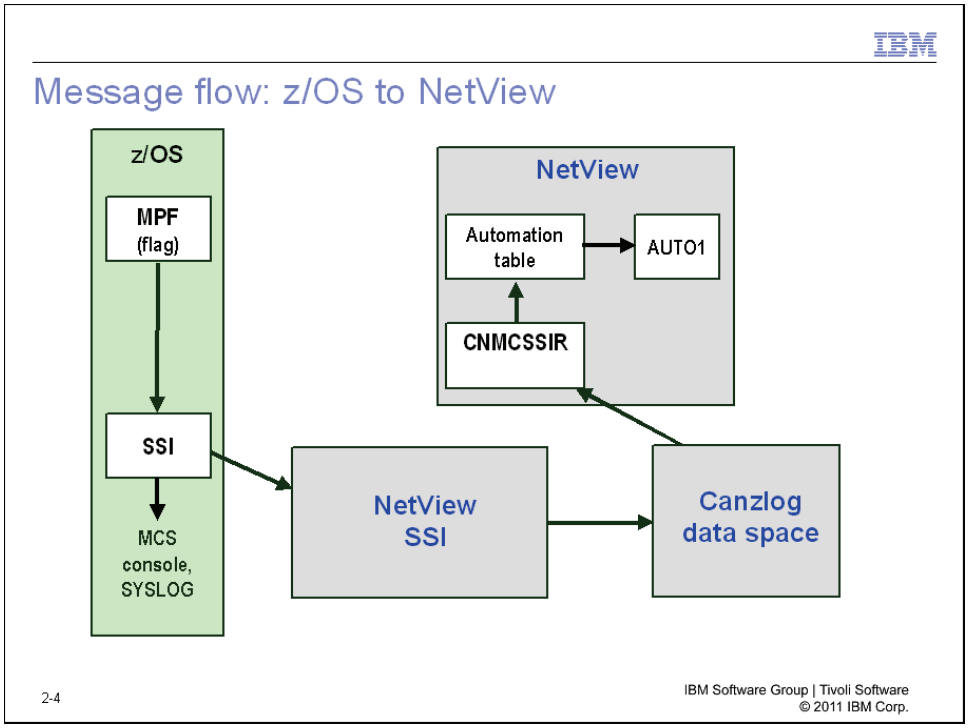
When the operator has performed the action required that a screen message displays, the system deletes the message, or the operator can use the CONTROL C command to delete the message. If AMRF is active, operators can remove action messages from the screen, then retrieve them in their entirety later by using the DISPLAY R command.

Use the MVS DISPLAY SSI command to display all of the SSI applications and their attributes, for example:

```
IEFJ100I  04.20.46  SSI DISPLAY 537
SUBSYS=JES2 (PRIMARY)
   DYNAMIC=YES     STATUS=ACTIVE       COMMANDS=REJECT
SUBSYS=MSTR
   DYNAMIC=NO      STATUS=ACTIVE       COMMANDS=N/A
SUBSYS=SMS
   DYNAMIC=YES     STATUS=ACTIVE       COMMANDS=REJECT
SUBSYS=AUTO
   DYNAMIC=YES     STATUS=ACTIVE       COMMANDS=ACCEPT
SUBSYS=EKGX
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=RACF
   DYNAMIC=YES     STATUS=ACTIVE       COMMANDS=REJECT
SUBSYS=DFRM
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=TNF
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=VMCF
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=BLX1
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=CNDL
   DYNAMIC=YES     STATUS=INACTIVE     COMMANDS=REJECT
SUBSYS=AXR
   DYNAMIC=YES     STATUS=ACTIVE       COMMANDS=REJECT
```

In this example, AUTO is the subsystem name for the NetView application (AUTONETV) and NetView SSI Assist Procedure (AUTOSSI). If you need the NetView SSI to be inserted into the z/OS SSI bus earlier or later, you move the subsystem name definition. Issue the MVS SETSSI command to dynamically update the subsystem name table.

Message flow: z/OS to NetView



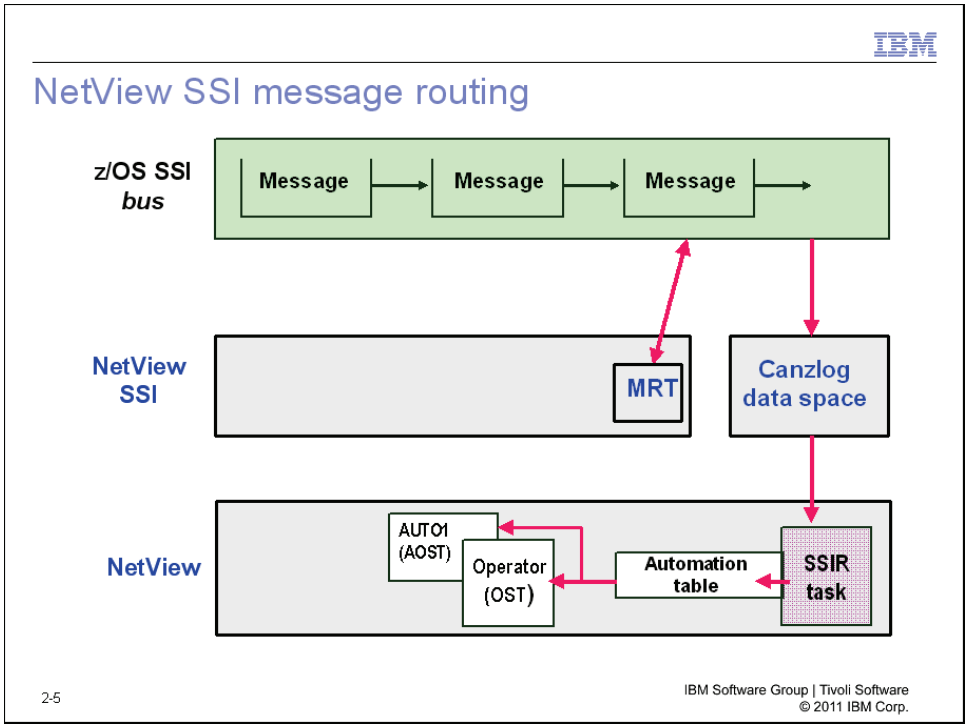
NetView can receive messages from the z/OS operating system. The messages pass from z/OS to NetView as follows:

1. The z/OS application issues a WTO macro, creating the z/OS message.
2. The z/OS message processing facility (MPF) examines the message, and sets the automation and suppress flags according to the MPF list specification.
3. The Multiple Console Support facility passes the message through the active SSI exits. The NetView exit (or exits) pass the message through MRT processing (if active), and copies the message with attributes into the Canzlog data space.
4. After MPF and MRT processing, the message might or might not be wanted for automation. If it is, the NetView SSI exit posts the NetView task CNMCSSIR to find new messages. The original message, with any changes made by the MRT, continues in parallel with z/OS consoles and syslog, as appropriate.
5. The CNMCSSIR task extracts the next automatable message from the Canzlog data space, and routes it through the automation table.

Note: Messages that are directed to an EMCS console that a NetView task owns are automated by that task. Other messages are automated at the CNMCSSIR task.

When the CNMCSSIR task is recycled by the RECYCLET command or is a result of RESTYLE MVSPARM command, no messages are bypassed by automation. However, when NetView itself is recycled or if the CNMCSSIR task is down, the new style statement, MVSPARM.Msg.Automation.MaxAge, controls how old messages can be and still be automated. The MaxAge value can be set as high as 86400, one full day, or as low as zero.

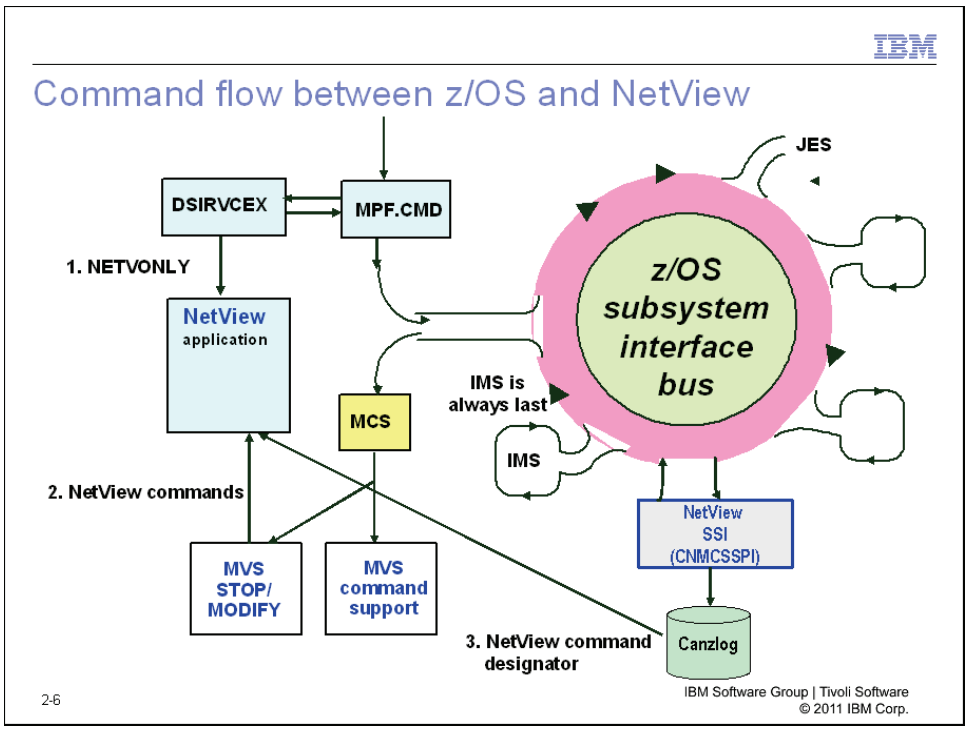
NetView SSI message routing



NetView SSI message routing occurs as follows:

1. The CNMCSSIR SSI router task, which runs in the NetView application address space, routes messages and commands between the NetView SSI and the NetView application.
2. The SSIR task inspects the buffers in the Canzlog data space, and copies (and queues) selected messages to process in the NetView application address space.
3. When the messages arrive, they route to the various NetView automation facilities, including the automation table. These, in turn, might cause the messages to route to one, some, or none of the different OSTs in NetView.

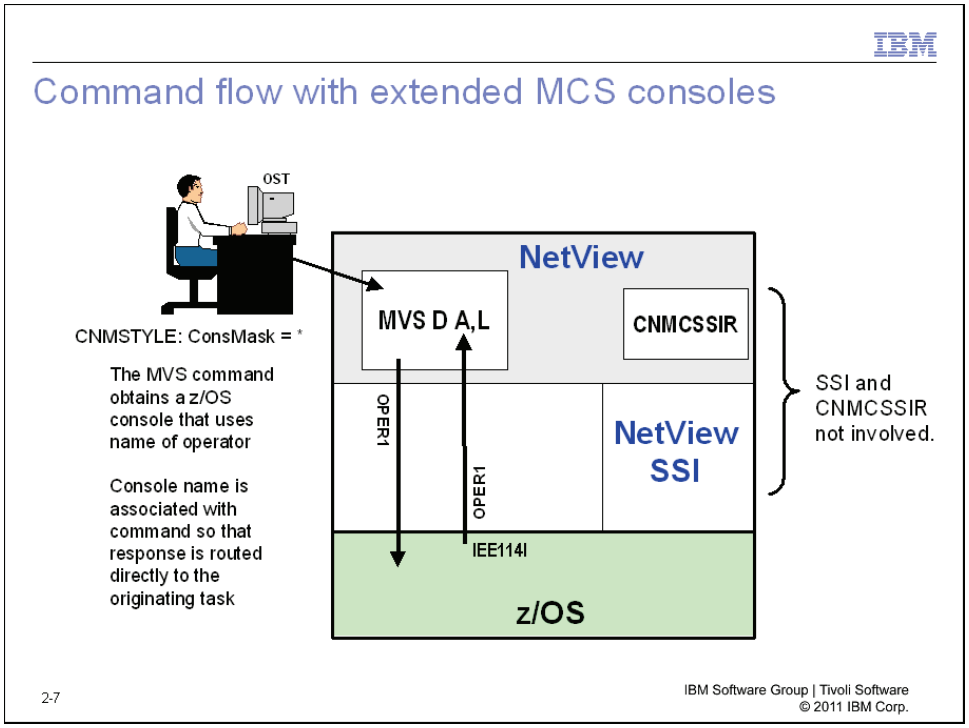
Command flow between z/OS and NetView



This slide shows the flow of commands between z/OS and NetView, as summarized as follows:.

1. Command Revision Table might revise commands. Commands that are flagged with NETVONLY are sent to the NetView application.
2. Commands that are sent to NetView through a MODIFY command are retrieved directly from z/OS.
3. Commands with the NetView command designator are routed through the Canzlog data space.

Command flow with extended MCS consoles



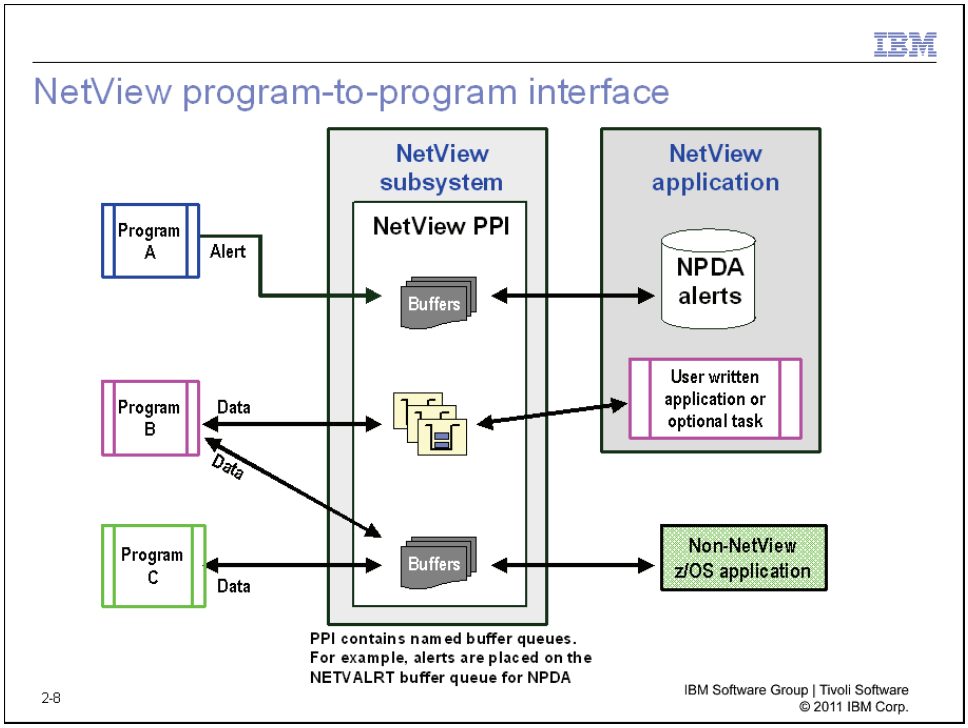
In this slide example, the MVS command obtains an Extended MCS console if the name matches the task name. This console name is sent with the command to z/OS. The command response is created using the WTO macro, but in this case, the MCS directly passes the response back to the calling task. The NetView subsystem and the SSIR task are not involved.

Note: Console names within a sysplex must be unique. You can use the *ConsMask* statement in CNMSTYLE to create unique console names.

In this example, **ConsMask = *** was defined to use the operator ID as the console name.

You can issue the GETCONTD command to obtain the console before any MVS command runs. This action helps because a sysplex might have several NetView applications, each with similar task names. Use GETCOND to create unique task names for each application. If you use a SETCONID command, you can set a name without actually creating an EMCS console. This is useful when some NetView tasks do not issue MVS commands.

NetView program-to-program interface

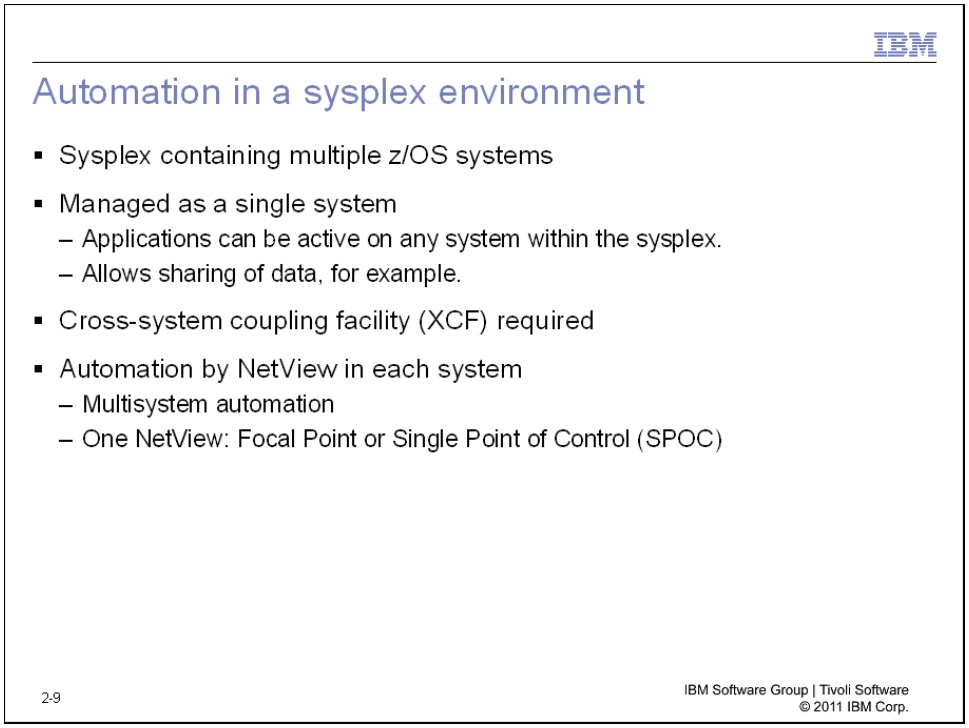


The NetView SSI address space supports the NetView program-to-program interface (PPI) for transfer of data between applications. NetView can be one of the applications.

Any application program that runs under z/OS can send data to the NetView PPI by using z/OS *cross memory services*. The data goes into named buffer queues. With PPI, programs can pass data directly to any other program in the same system. NetView applications can be one, both, or none of the programs involved in such transmissions. Note examples as follows:

- Use of the PPI to send messages from one batch job to another
- Use of the NetView PPI buffer called NETVALRT for sending alerts to NetView
- Applications that use the NetView PPI to share data. The applications can be in the NetView address space or in separate z/OS address spaces. NetView macros are used for sending and receiving data. For more information, see the *NetView Application Programming Guide*.

Automation in a sysplex environment



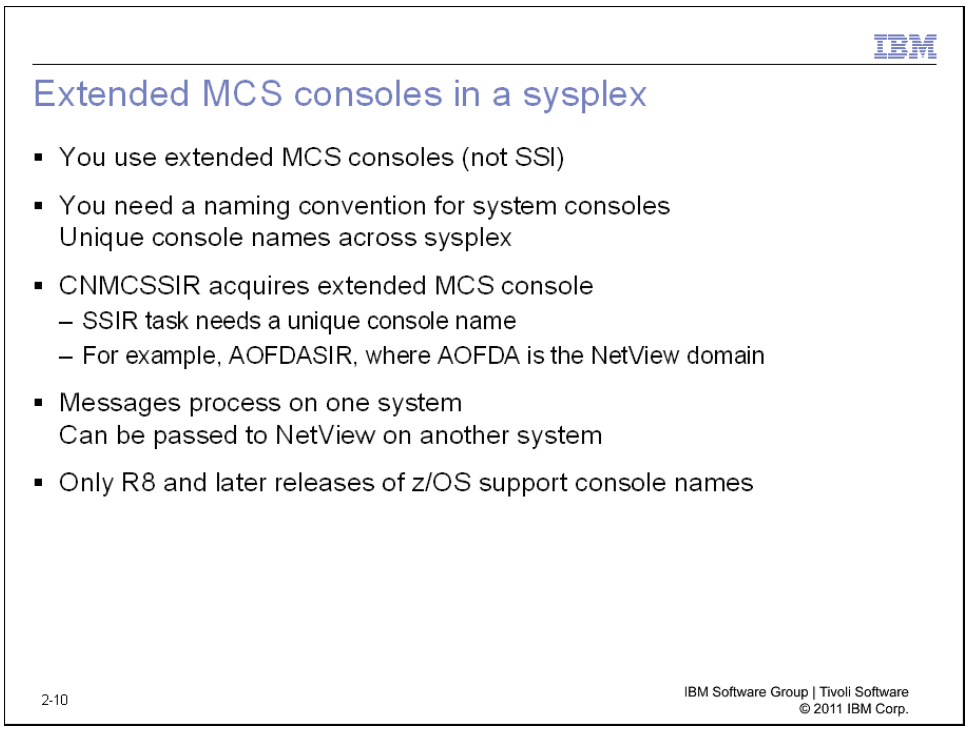
A z/OS sysplex is a configuration of multiple z/OS operating systems that work as a single system by sharing functions and programs. A key component is the cross-system coupling facility (XCF). When using NetView automation in a sysplex environment, NetView can use the standard z/OS commands. These commands include routing commands to other systems in the sysplex and receiving their responses. The following considerations are significant:

- Using Extended MCS consoles
- Coordinating z/OS message processing options with Extended MCS
- Deciding whether to centralize system automation on one system or not

For more information pertaining to sysplex automation, see the NetView *Automation Guide*.

Tip: SA z/OS provides you with extensive automation and management capabilities for your sysplex.

Extended MCS consoles in a sysplex



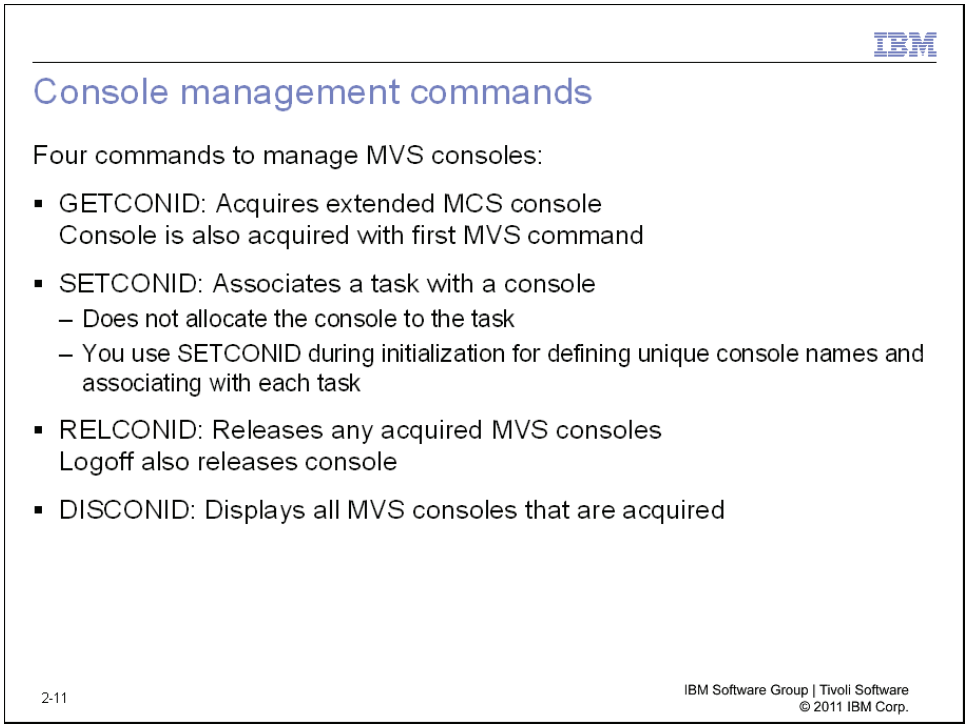
Use Extended MCS consoles for message delivery instead of using the SSI when using Netview for automation in a sysplex. A strategy is necessary for naming consoles.

EMCS console names must be unique across the sysplex. Use the *ConsMask* statement in CNMSTYLE to create unique console names. If you assign console names to match the task names, and the CNMCSSIR task acquires an EMCS console, ensure a unique name for each CNMCSSIR task. The convention is to create a name of the form *xxxxx*SIR, where *xxxxx* is the NetView domain name. An example is AOFDASIR.

Operators might be logged on to several NetView applications in the same sysplex while also using TSO SDSF to enter z/OS commands. If so, they need to obtain consoles with unique names across the sysplex.

The MPF table in the system that originated a message processes the message. However, the message might pass to other systems where a NetView facility might perform further processing.

Console management commands



By default, when you issue an MVS command (for example, MVS D A,L), you acquire an EMCS console name as the task name that issues the command. This situation can cause problems in a sysplex environment because the console names must be unique across a sysplex.

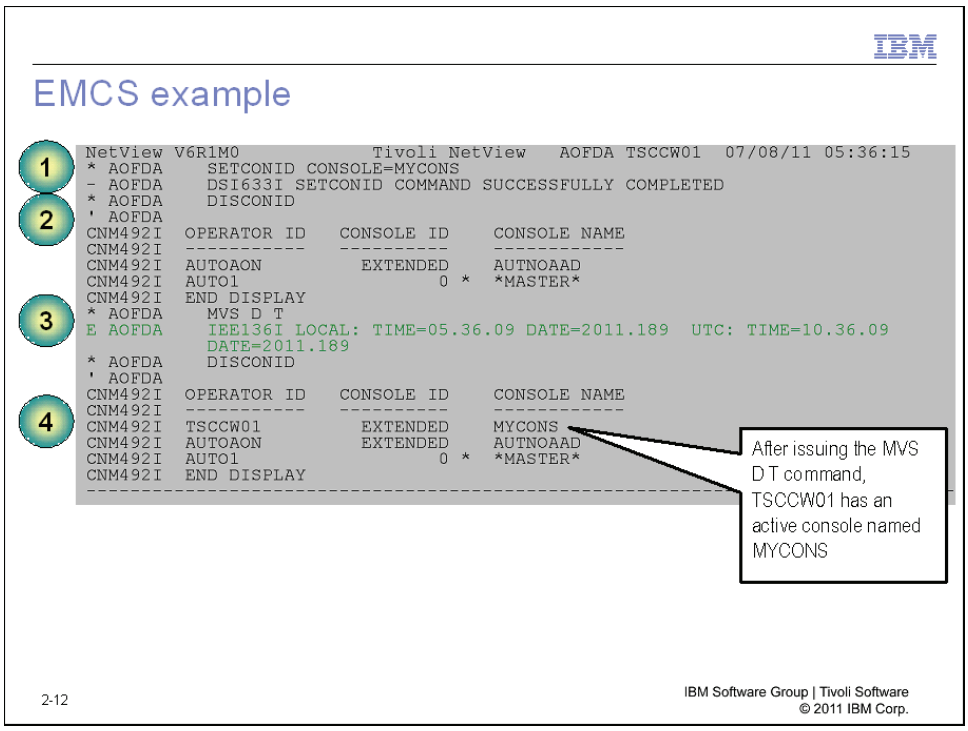
Using a naming strategy, you can achieve unique console names with GETCONID. Acquire a unique EMCS console (or issue a SETCONID command to associate a unique EMCS console) in an operator initial command list (IC). The initial command list is run when the operator logs on to NetView.

The operator initial command list is defined in the operator profile. For example, suppose you use profile DSIPROFB and it looks like the following text:

```
DSIPROFB     PROFILE  IC=LOGPROF1
             AUTH     MSGRECVR=YES,CTL=GLOBAL,NGMFADMN=YES
             END
```

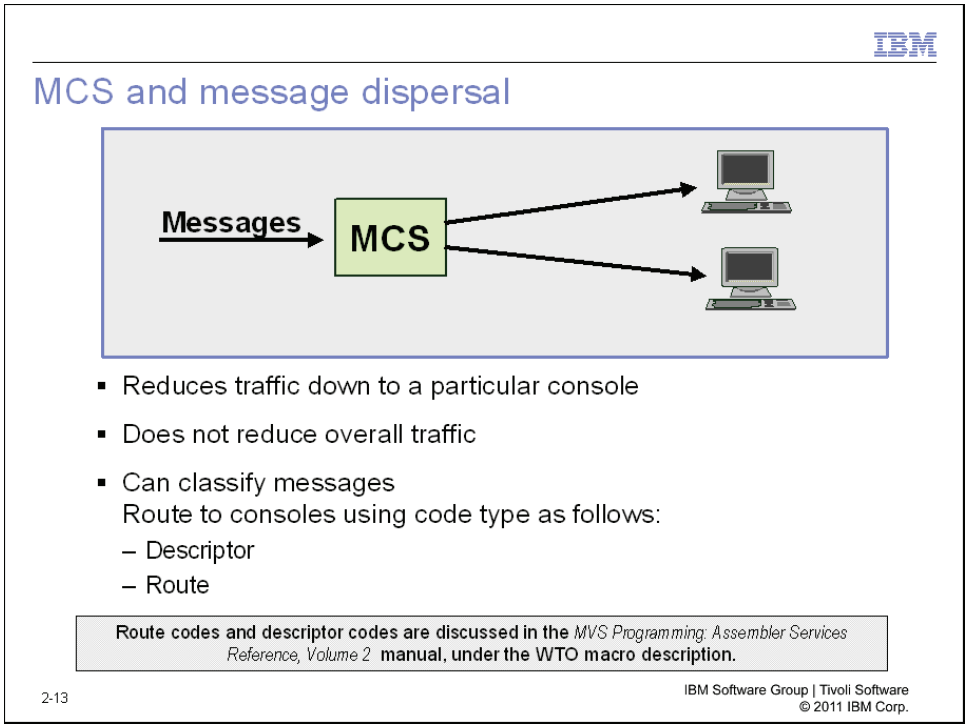
In this case, the operator initial command list is LOGPROF1. Each NetView task can have only one EMCS console associated with it.

EMCS example



In this slide example, the console named MYCONS associates with operator (OST) TCCW01 from the use of the SETCONID command. The DISCONID command shows three consoles have been allocated. MYCONS is not allocated because no commands have been issued yet. The MVS D T command is issued. This causes the MYCONS console to be acquired by TSCCW01, as shown in the second DISCONID command.

MCS and message dispersal



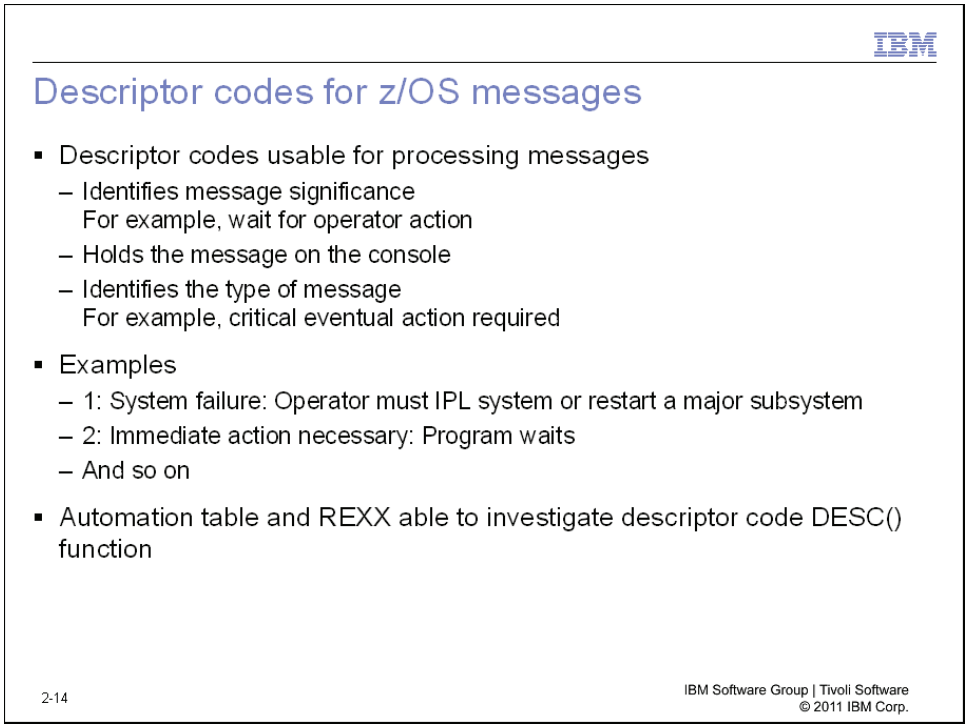
You can reduce message traffic to a particular console. MCS enables specific consoles to receive particular *classes of messages*. Total traffic does not reduce, but message dispersion improves.

Note: Within NetView, the ASSIGN command and automation table also perform a similar routing function, to be discussed in a later unit.

Each z/OS system message has two codes so that you can define the message flow, enabling message processing by the operators as follows:

- Descriptor code: Provides a description of the significance of a message, used by MCS for determining how the message is to be displayed.
- Route code: Provides the ability to group messages by function, used for routing the message to the appropriate MCS consoles.

Descriptor codes for z/OS messages



A z/OS message might have a descriptor code (numbered from one to 16) associated with it. The descriptor code describes the significance of the message (for example, immediate action required). The code designates how the system displays the message, or if it deletes the message.

The last character of a message ID, by convention, matches the descriptor code. For example, a last character of A (as in DSI802A) indicates an immediate-action message.

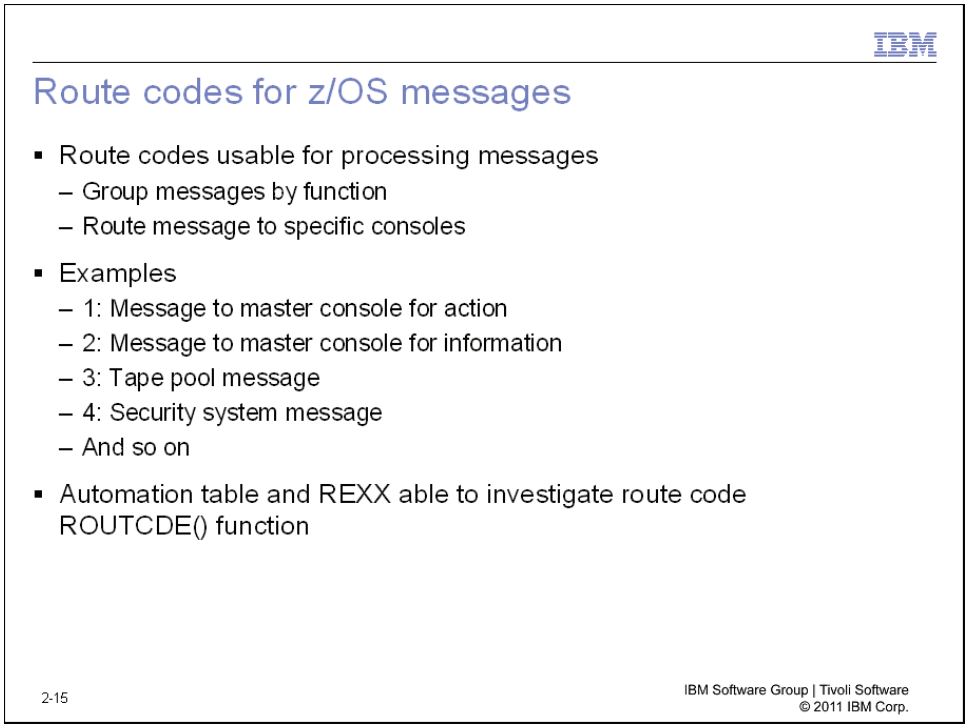
Examples of descriptor code values are as follows:

1 = System failure

2 = Immediate action required

Descriptor codes of 1, 2, 3, and 11 denote *action* messages. Action messages stay on the MCS console display until removal, either by an operator or because of programming. Action messages can optionally be retained by the Action Message Retention Function (AMRF) for later reviewing. The exception is the active Message Processing Facility specification that indicates otherwise. Action messages also stay on NetView consoles.

Route codes for z/OS messages



Route codes (numbered from one to 128) are the key to using MCS for a z/OS environment. With route codes, you can group messages by function, and send messages to specific consoles. Examples of route code values are as follows:

1 = Message to master console for action

2 = Message to master console for information

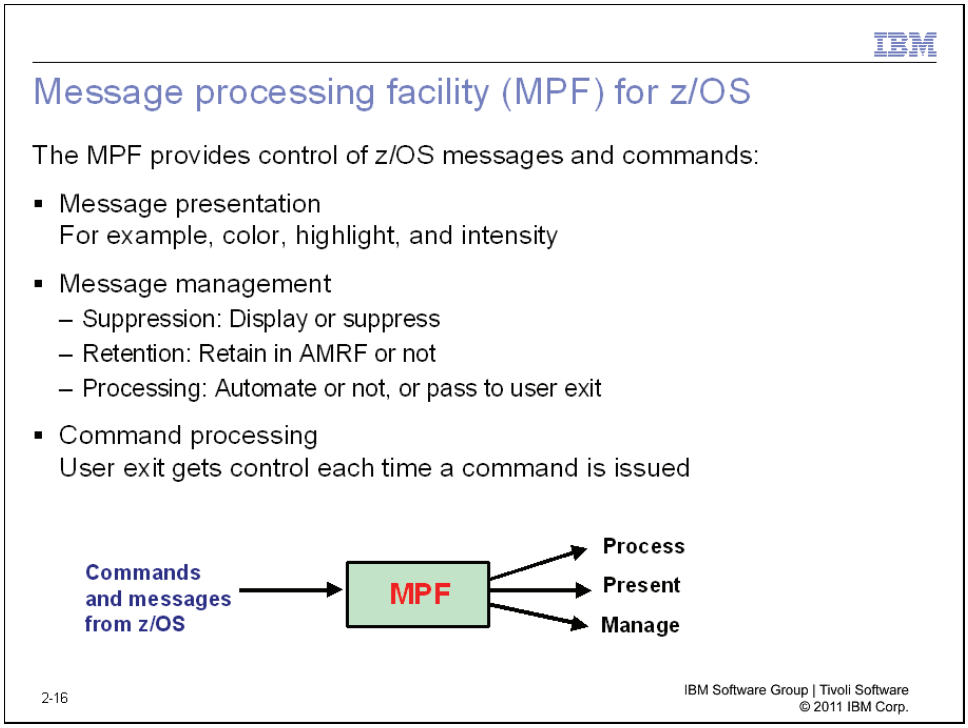
3 = Tape pool message

9 = System security message

For example, an MCS Console could be defined to receive only messages of route codes one, two, and nine, but not messages with other routing codes. A message might have more than one route code.

NetView automation table processing for messages depend on the descriptor codes, routing codes of messages, or both.

Message processing facility (MPF) for z/OS



In a z/OS environment, the message processing facility (MPF) controls as follows:

- Command processing

  Provides control to user each time a command is issued. This feature is available if the installation defines user exits that receive control when a command is entered. The NetView Command Revision Table (CRT) uses this z/OS feature to screen commands issued by all operators. NetView Command Revision Table discussion is in Unit 3.

- Message presentation

  Determines color and intensity of messages on operator consoles. Highlighting indicates that the messages are in color.

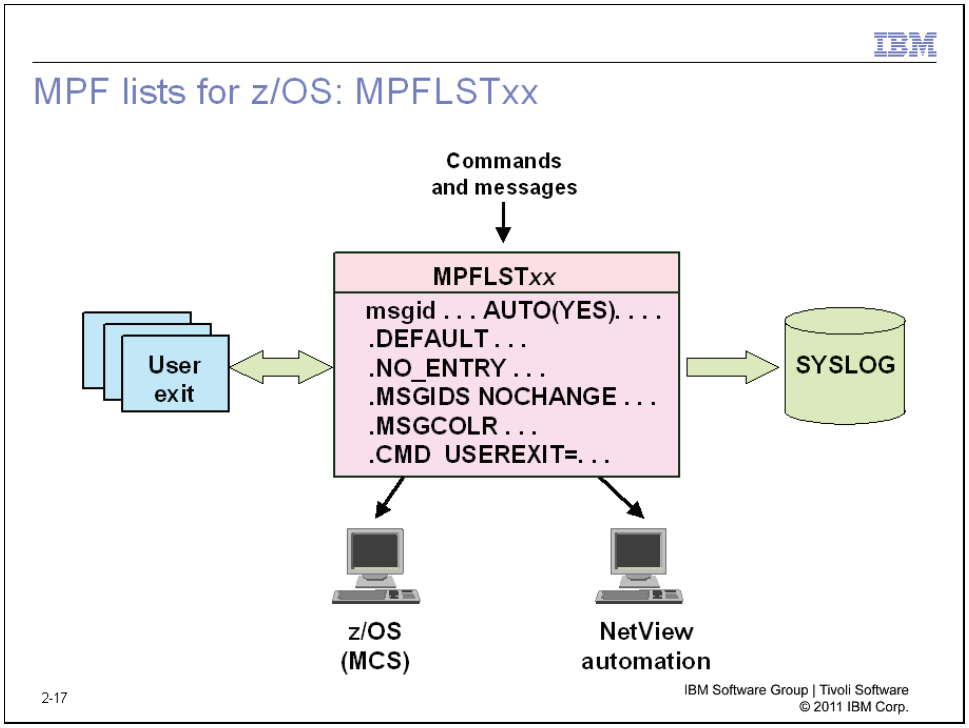
- Message management

  – Message suppression: Indicates if messages are to be displayed on an MCS console where they are directed.WTO(R) messages are always written to the syslog.

  – Message retention: Indicates if messages are retained by automation message retention facility (AMRF).

  – Message processing: Indicates messages that are to be processed by a message automation subsystem (for example, NetView), or by an MPF user exit.

MPF lists for z/OS: MPFLSTxx

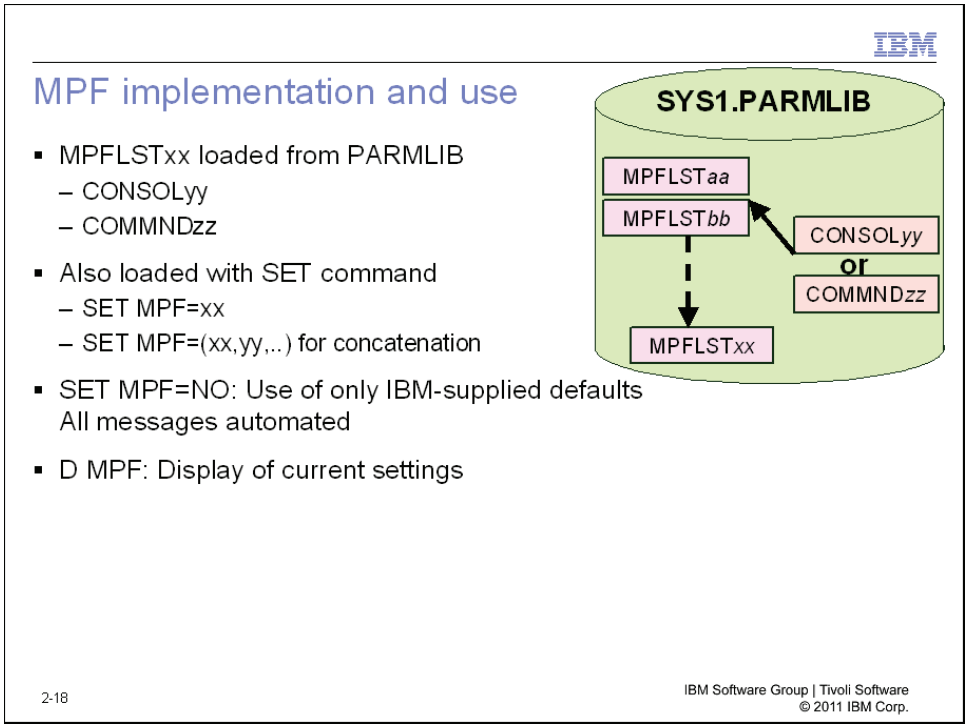


Message processing facility lists define how MPF processes commands and messages. The lists are MPFLST*xx* members in your PARMLIB. You can create as many MPF lists as you want. The most common use of MPF lists is for *message management,* with some *message presentation*, and sometimes with *command processing*.

Note: IBM does not supply any MPFLST*xx* members with the z/OS products. SA z/OS generates an MPF list (MPFLSTSA) when you build the automation policy.

See the *z/OS Installation And Tuning Reference* manual for more detailed information.

MPF implementation and use



Various MPF members might be in the z/OS PARMLIB. Each member has a name of MPFLST*xx* and is activated by one of these following methods:

- The specification of an MPFLST*xx* member. The member is in the INIT statement of the CONSOL*yy* member of PARMLIB that is used at system initialization.
- Issuing of the SET MPF=*xx* command. The command comes from the COMMND*zz* member of PARMLIB that is used at system initialization.

An authorized user can issue the z/OS SET MPF=*xx* command. To properly concatenate several MPF lists, you must use the SET MPF=(xx,xx,..) command.

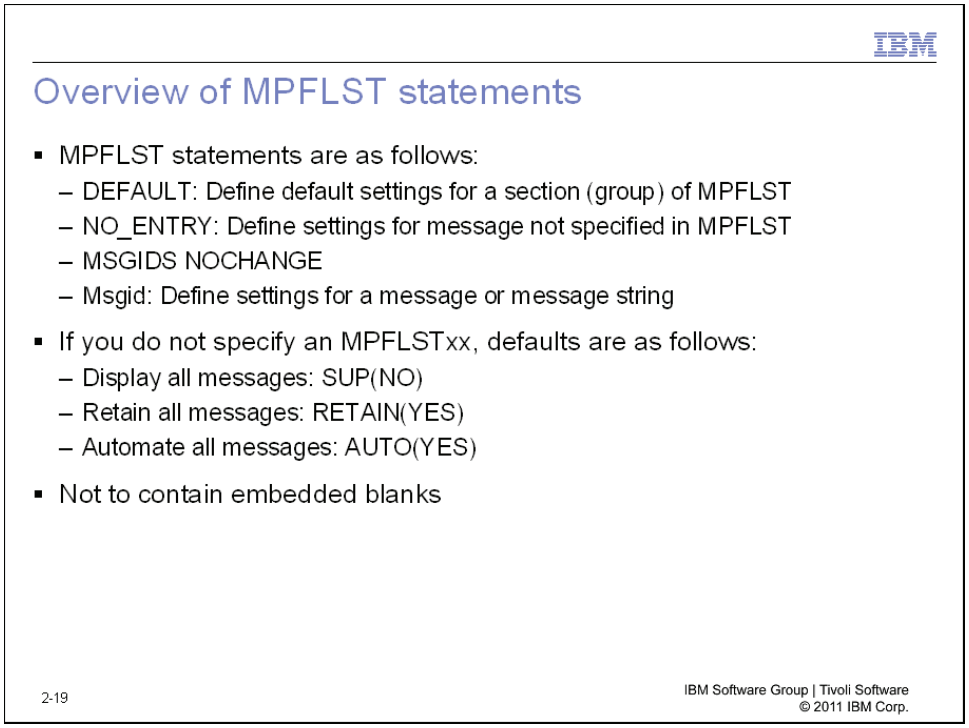
Using the SET MPF command to set a list overrides any current active list, unless the NOCHANGE operand is used within the MPF list definition. In such cases, some of the definitions are left unchanged.

The default is SET MPF=NO. If the operator issues the following command, default values are used for message presentation and message management of all messages:

```
Display all messages, retain all action messages, and send all
messages to automation.
```

Display the active MPF lists by using the z/OS D MPF command.

Overview of MPFLST statements



The most important of the MPF statements for message processing are as follows:

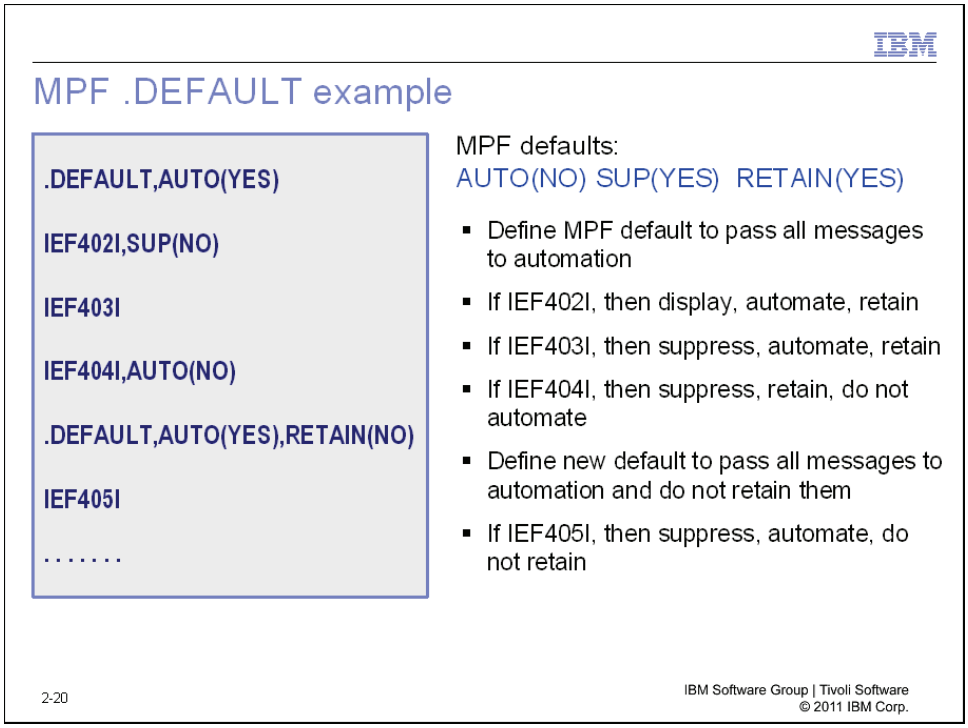
```
.DEFAULT
.NO_ENTRY
.MSGIDS NOCHANGE
msgid
```

If no message-processing statements or parameters occur in the active MPF list, default actions are as follows:

```
Display all messages
Retain all action messages
Automate all messages
```

These defaults are the same as when no MPF list is active.

MPF .DEFAULT example

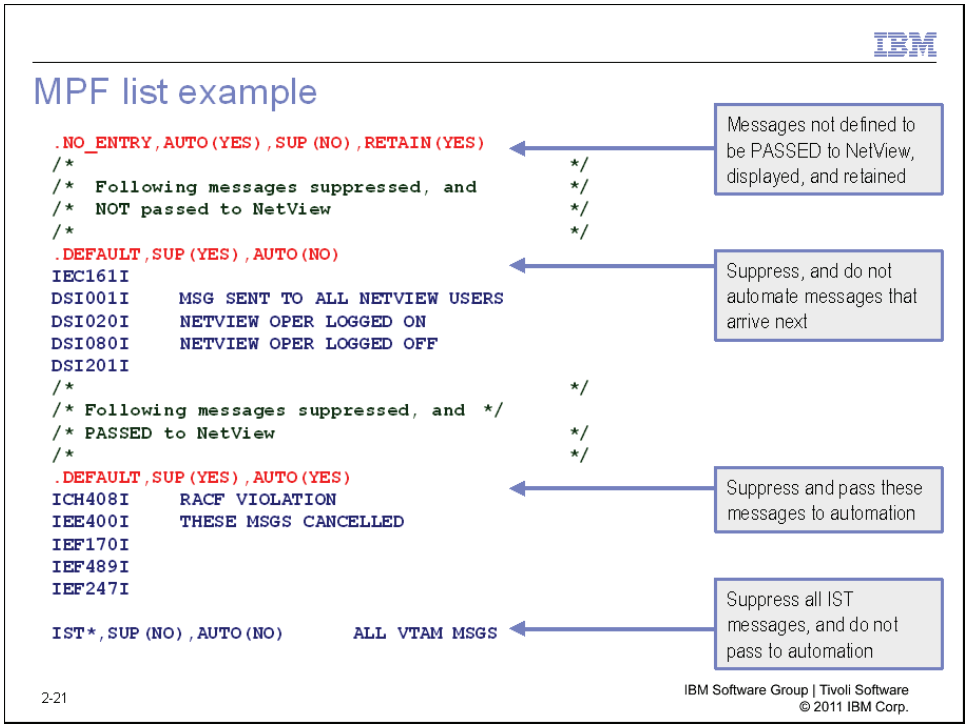


Any number of .DEFAULT statements can be coded. Any number of *msgid* statements that follow a .DEFAULT statement can override the values set on that .DEFAULT statement.

Four *msgid* statements are on the example slide: IEF402I, IEF403I, IEF404I, and IEF405I.

- The first .DEFAULT statement sets a default of AUTO(YES) for all messages that follow, and inherits the MPF default settings for SUP(YES) and RETAIN(YES). The example is IEF403I.
- Message IEF402I is displayed based on its SUP(NO), eligible for automation based on the .DEFAULT,AUTO(YES) statement, and retained based on the default MPF settings.
- Message IEF403I is eligible for automation based on the .DEFAULT,AUTO(YES) statement, and suppressed and retained based on the default MPF settings.
- Message IEF404I is not eligible for automation based on its AUTO(NO), and to be suppressed and retained based on the default MPF settings.
- The second .DEFAULT statement sets a default of AUTO(YES) and RETAIN(NO) for all messages that follow. The example is IEF405I.

MPF list example



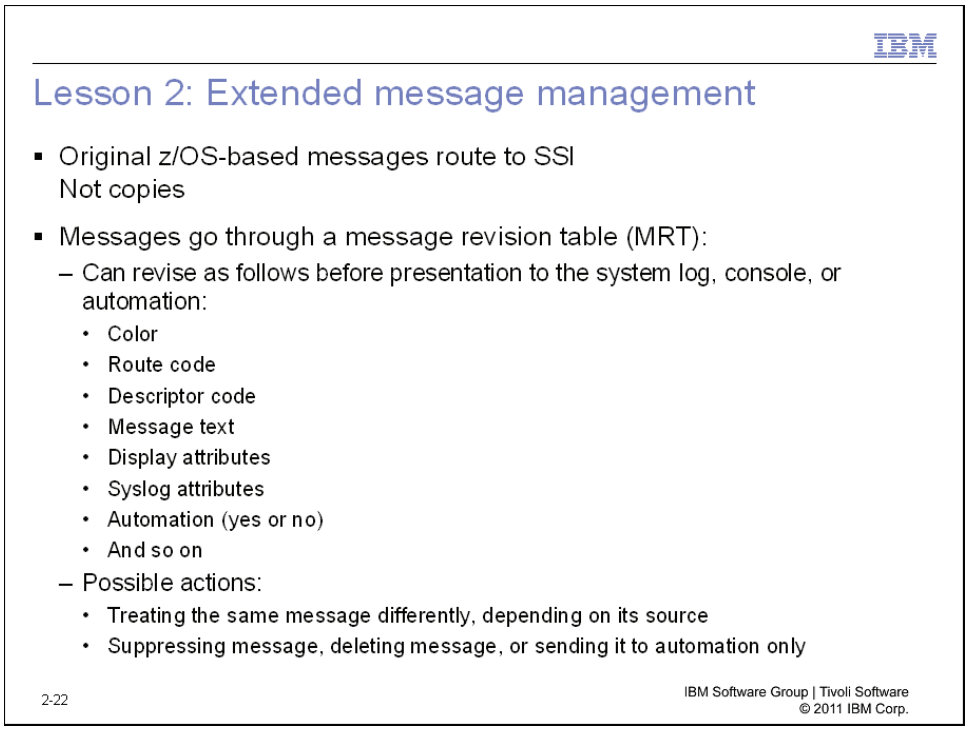
This slide illustrates an example of an MPF list. The .NO_ENTRY statement specifies that messages that are not defined in the MPF list are to be sent to NetView for automation, shown, and retained. Messages that are not in the list are to be handled with the .NO_ENTRY statement.

The first .DEFAULT statement suppresses messages, and does not send them to NetView for automation. This applies to all of the messages that follow this .DEFAULT statement: IEC161I, DSI001I, DSI020I, DSI080I, and DSI201I.

The second .DEFAULT statement suppresses messages and sends them to NetView for automation. This action applies to all of the messages that follow this .DEFAULT statement: ICH408I, IEE400I, IEF170I, IEF489I, and IEF2471I. The statement does not apply for all IST prefix (VTAM) messages. The IST* msgid statement overrides the second .DEFAULT statement by displaying the messages and not passing them to automation.

The .NO_ENTRY statement placed first. It is good practice to code one, even if it specifies the default values.

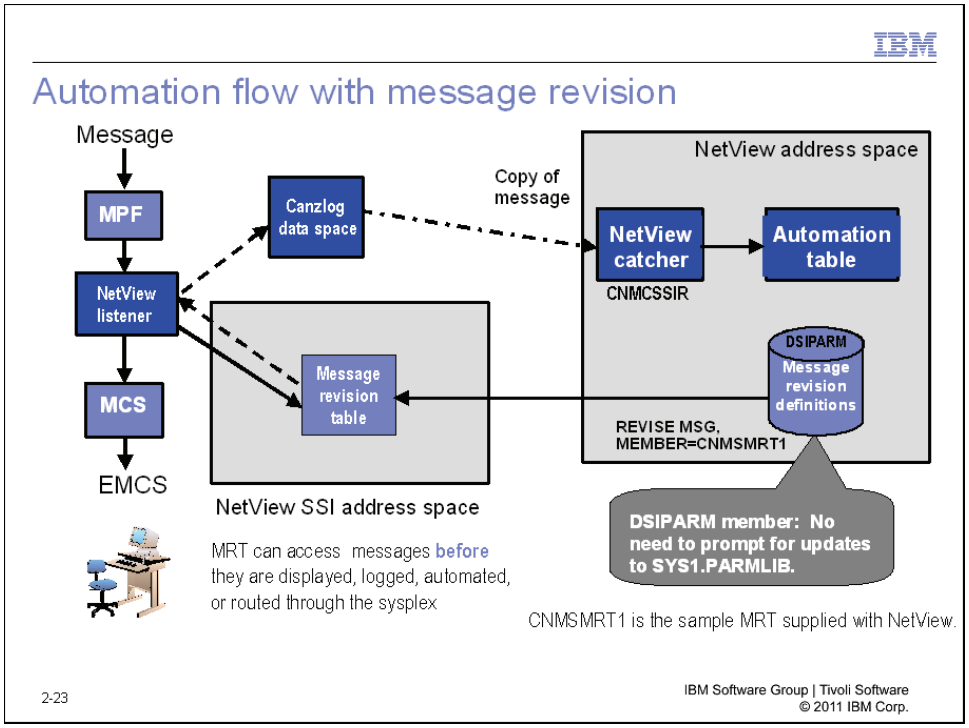
Lesson 2: Extended message management



This lesson discusses the functionality of the NetView message revision table (MRT). As messages come from the z/OS console into the NetView SSI, they can process through the NetView MRT. The MRT is in storage in the NetView SSI address space. You can use the MRT to suppress a message, change its attributes, and control whether it should be sent to NetView for automation or not.

The MRT can process any message from the z/OS console.They can be z/OS system messages, VTAM messages, application messages, and so on. Messages are received by the MRT after MPF processing but before being sent to a console. The MRT does not perform any automated functions. Automation of the message is still performed by the NetView automation table.

Automation flow with message revision



The MRT is a set of statements in DSIPARM. (The example member name is CNMSMRT1.) The NetView REVISE command loads the MRT in storage within the NetView SSI address space. Only one MRT for each NetView can be active. If you have three active NetView procedures (with different SSI names), each procedure can have its own MRT. Take care that the action of one MRT does not interfere with requirements of another. You can include members with %INCLUDE statements. Data REXX is also supported.

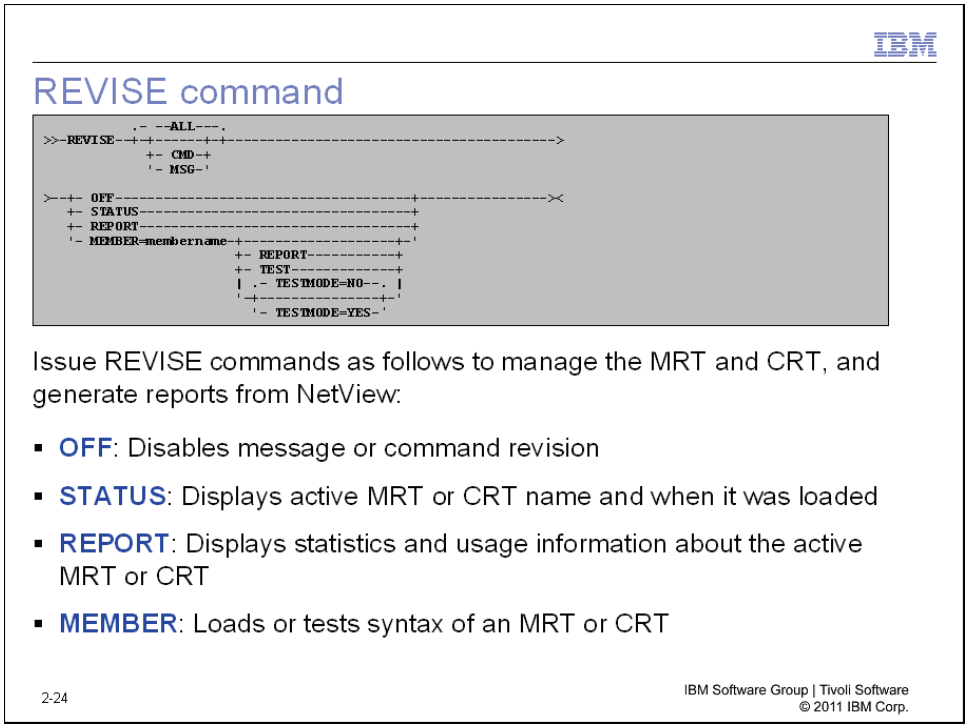
The *original z/OS messages* (z/OS system messages, VTAM messages, application messages, and so on) go to the NetView SSI, after processing by the message processing facility (MPF). If the message is suppressed with MPF, it goes to the SSI bus. Each application on the SSI bus can change the message attributes. For example, you can suppress a message in MPF, and use the MRT to display it.

The NetView SSI checks the loaded MRT definitions to designate the action to be taken. For example, the message can be suppressed, attributes of the message can be changed, or the message can be sent to NetView for automation. Attributes include color, route codes, descriptor codes, and more.

The message goes to the MRT processing before it is displayed on the system console. If you modify the message in the MRT, you modify the message that is displayed on the system console.

Important: Be careful when modifying or suppressing messages. The modified message routes to the remaining applications on the SSI bus and also the system console. This can adversely impact other applications.

REVISE command



This slide illustrates the syntax of the REVISE command, which replaces the old REVISMSG command that handled only MRT.

REVISE MSG,MEMBER= *membername* loads an MRT where *membername* is a set of MRT statements in DSIPARM. The MRT can contain other members with use of %INCLUDE. Data REXX is also supported.

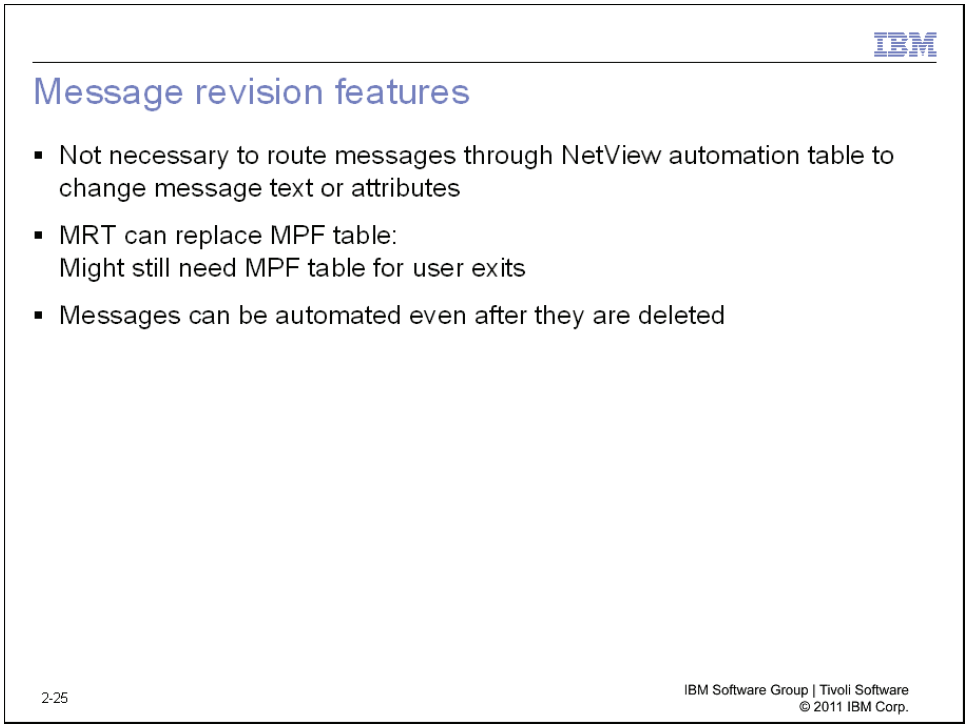
Note: The statements load in the order given, but the SSI processes the statements in a different order than the load order.

REVISE MSG,REPORT generates a report that includes the following information:

- Number of messages that have gone through the MRT
- Number of messages suppressed
- MRT activity based on statement within the MRT

The REVISE command supports other parameters. See the *Command Reference* manual for more details or the online help, such as “HELP REVISE”.

Message revision features

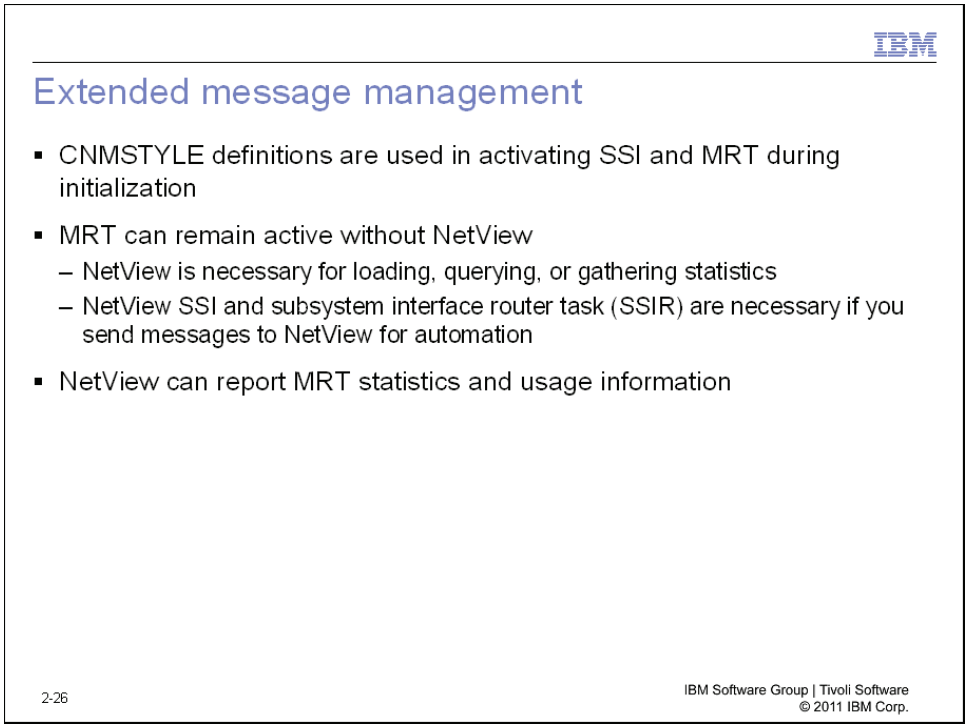


With the MRT, you can change the color of a message. This change affects the original message to be displayed on the system console and also the copy that was sent to NetView. Without the MRT, you can use the automation table to modify the message for only the copy that goes to NetView.

In most cases, you can replace the MPF settings with MRT statements. The MRT provides functions to parse and process messages.

Note: If you call user exits from an MPF list member, you still need the MPF to call the exit. Some products might also send user exits to handle message overflows at the system console.

Extended message management



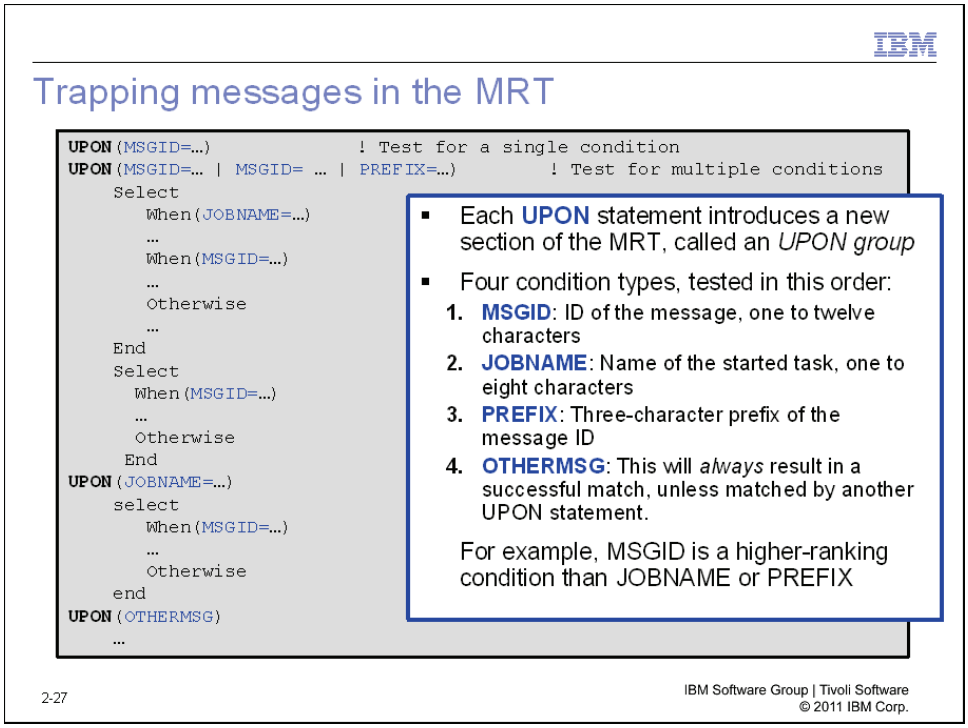
CNMSTYLE definitions are provided fr performing the following tasks:

- Start the NetView SSI address space when NetView initializes.
- Load a Message Revision Table (MRT).

Tip: After loading, the MRT has no dependency on NetView being active. For example, if you start NetView before JES, you can start the NetView SSI, load the MRT, and stop NetView. As your system IPL continues, your automation NetView can be started to support your automation needs.

Using the REVISE MSG,REPORT command, you can display statistics and detailed usage information. You can correlate the detailed usage information to specific statements within the MRT source in DSIPARM.

Trapping messages in the MRT



MRT statements fall into the two following categories:

- Statements that trap messages. This slide shows the statements that are used for trapping messages.
- Statements that take actions on trapped messages. Discussion about MRT actions occurs in a few slides.

Note: Mixed-case text is supported.

Within the MRT, the UPON statement identifies groupings of conditions that trap one or more messages. These conditions are collectively called an *UPON group*. Each UPON group defines a new section of the MRT.

UPON conditions can be one or more combinations of the following conditions:

- MSGID: Traps a specific message ID or substring of a message ID if an asterisk (*) is used.
- JOBNAME: Traps all messages from the specified job. An asterisk (*) is supported as a wildcard.
- PREFIX: Traps all messages that match the specified three-character prefix.
- OTHERMSG: Defines default actions for messages that are not already processed (trapped) in the MRT. The OTHERMSG condition is not allowed with other UPON conditions. There can be only one UPON(OTHERMSG) condition in an MRT.

The example that this slide shows contains four UPON groups, including one UPON(OTHERMSG). The UPON conditions are ranked in the order shown. MSGID is the *highest ranking* UPON group, and UPON(OTHERMSG) is the *lowest ranking* UPON group.

Multiple conditions can be specified within an UPON group by using an OR symbol (|). The AND symbol (&) is not supported.

An UPON group ends when the next UPON group is encountered or the end of the MRT is reached. Within an UPON group, you can code one or more SELECT statements as follows for parsing the message:

- SELECT: Introduces a series of WHEN statements.
- WHEN: Specifies one or more conditions to take action on.

  WHEN statements can test other conditions of the current message. Example conditions include console name, work queue element (WQE) bits, descriptor code, route code, and checking of syslog being on or off.
- OTHERWISE: Processes all *unmatched* conditions for a particular SELECT statement.
- END: Identifies the end of a SELECT statement.

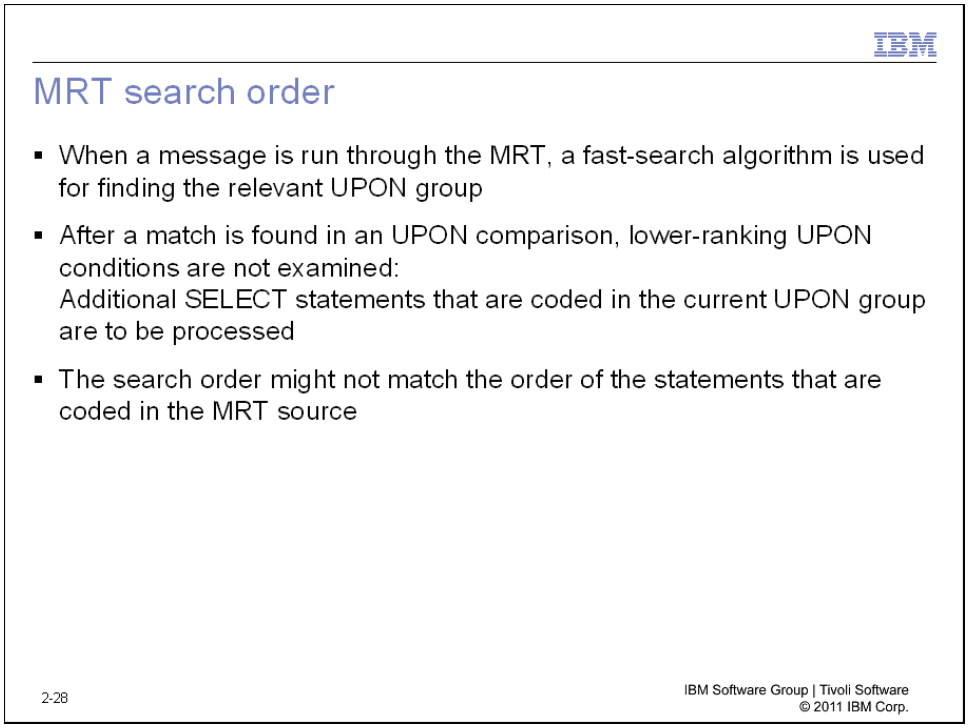
Note: Only the first line of a multiline message is examined. Actions taken, such as changing the color of a multiline message, apply to all lines of the multiline message.

you can code MRT comments in two ways:

- Starting the first column with an asterisk (*).
- Within MRT statements, following an exclamation point (!), as the following example shows:

```
UPON(MSGID='IEF403I') ! Test for a single condition
```

MRT search order



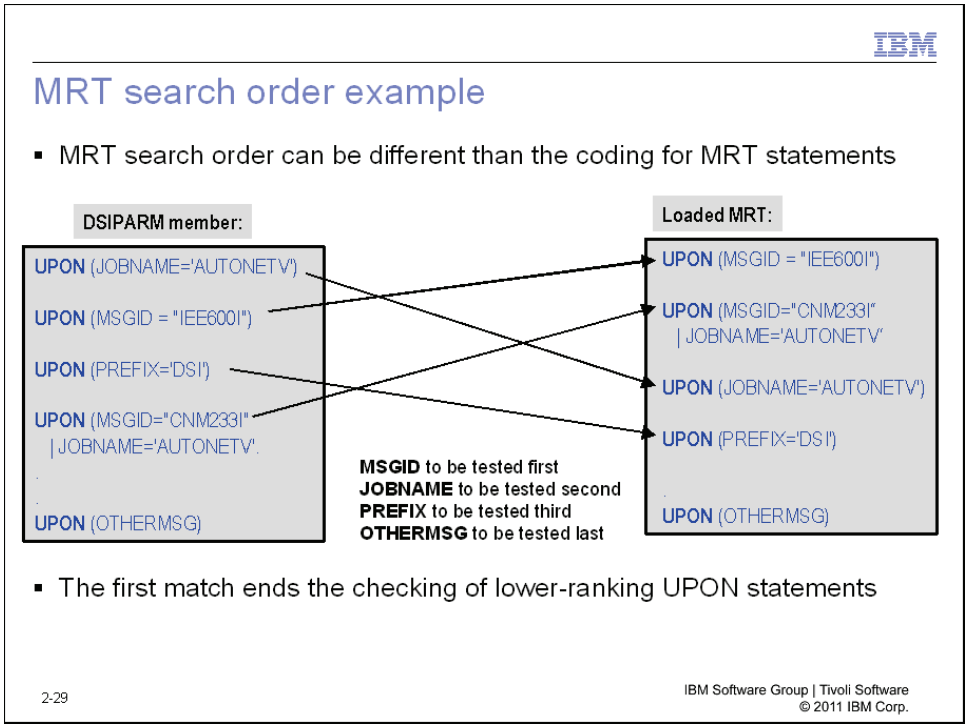
UPON statements are used fr trapping messages that run through the MRT. If a match is found, subsequent (*lower ranking)* UPON conditions are not examined. You can code multiple SELECT statements to take multiple actions for the message.

Each *UPON group* can test for a message ID, z/OS job name, or message prefix. Each of the condition types is assigned a priority with MSGID condition checks that have the highest priority and PREFIX condition checks that have the lowest priority.

Important: Because of the priority assigned to the condition types, the MRT that is loaded into storage in the NetView SSI might not match the exact order of the MRT statements as coded in DSIPARM. This can cause confusion when you attempt to debug the reason that your MRT outcome did not meet your expectations.

An UPON group that tests a message prefix (PREFIX) is a lower ranking UPON when compared to one that tests for a message ID (MSGID). UPON(OTHERMSG) is the lowest ranking UPON condition of an MRT.

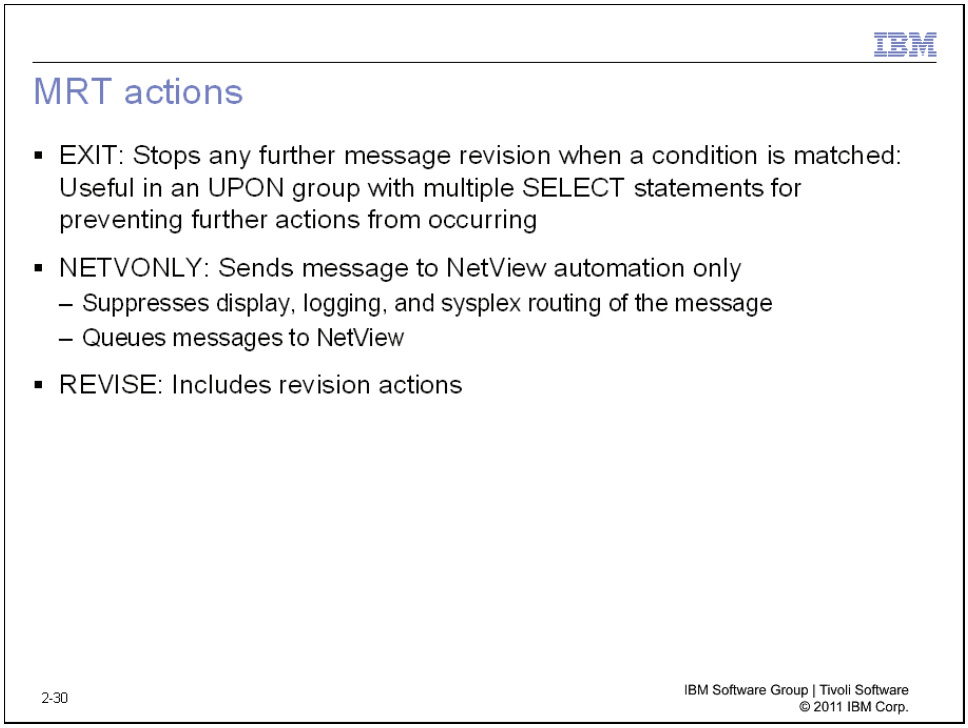
MRT search order example



This slide shows an example MRT source in DSIPARM and the order of the statements in the actual MRT that is loaded in storage in the NetView SSI. Key points to note are as follows:

1. The second statement in the MRT source is loaded as the *highest ranking* statement because it contains a test for MSGID. In this example, the test is for only MSGID.
2. The fourth statement in the MRT source is loaded as the second highest ranking statement because it also contains a MSGID test. It is loaded after the test for only MSGID because the check for JOBNAME reduces its priority.
3. The first statement in the MRT source is loaded next because JOBNAME is a lower priority than MSGID but higher priority than PREFIX.
4. The third statement in the MRT source is loaded as the *lowest ranking* statement because of the check for PREFIX only.
5. UPON(OTHERMSG) is always the last statement in the MRT to load.

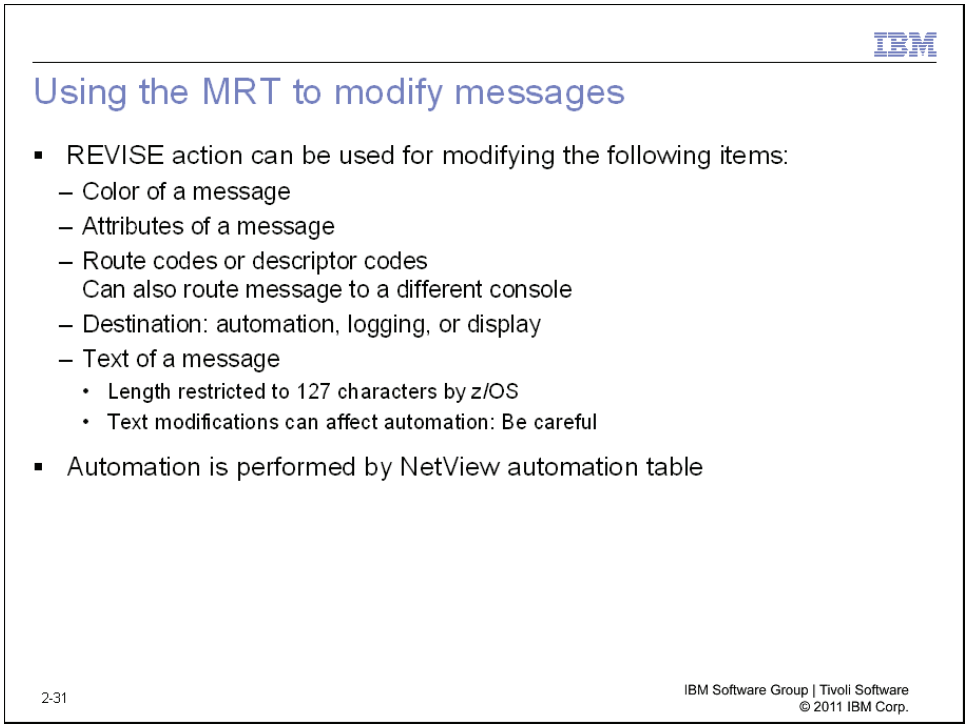
MRT actions



Three actions that are available are EXIT, NETVONLY, and REVISE. You can specify more than one action.

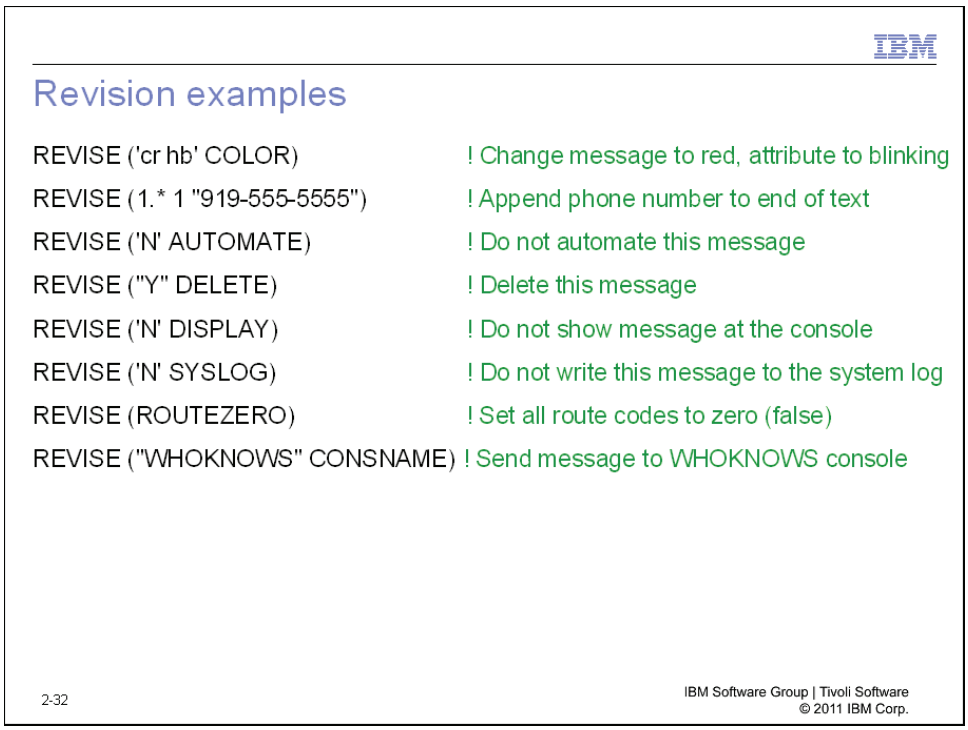
- EXIT: Ends with the current condition that matched. No further tests occur. This is analogous to a CONTINUE(STOP) in the automation table.
- NETVONLY: Message goes to NetView for automation only. The message is not displayed, not logged, and not routed.
- REVISE: You can modify the message text, attributes, descriptor code, route code, and more.

Using the MRT to modify messages

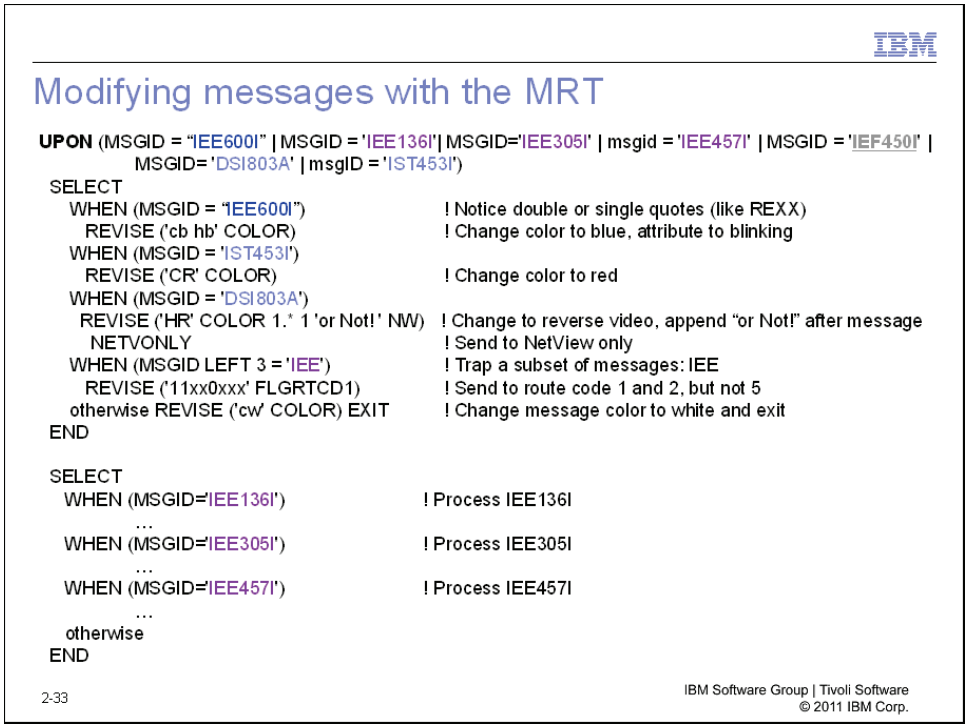


The MRT does not perform automation. Messages that run through the MRT must be passed to NetView and the automation table for automation to occur. You can use the MRT REVISE action to modify attributes of a message, such as its color, route codes, descriptor codes, and so on.

Revision examples



Modifying messages with the MRT



In the slide example, this UPON group traps seven messages and processes all of them in two single SELECT statements. The second SELECT statement processes the IEE messages individually.

The IEE600I message is trapped by an exact match for its message ID and is not passed to the remaining WHEN statements. If there were another SELECT statement, the IEE600I would be passed to it. It is to be displayed as blue and blinking on the system console.

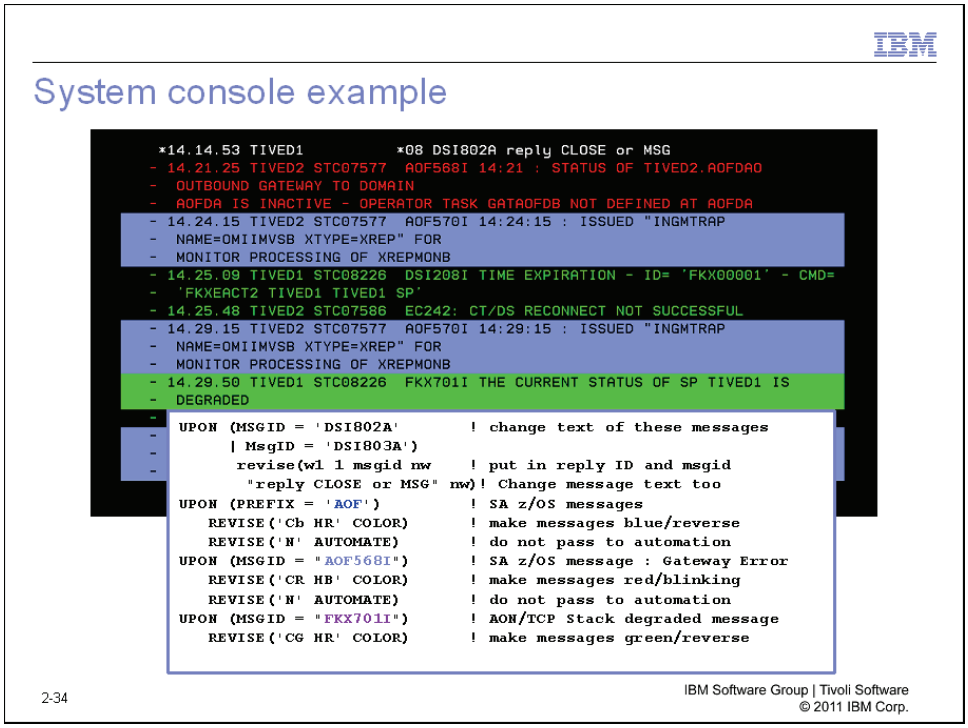
Three of the messages that the UPON statement (IEE136I, IEE305I, and IEE457I) trapped match the WHEN (MSGID LEFT 3 = 'IEE') statement in the first SELECT. The messages for their route codes undergo modification. The second SELECT statement processes each individual IEE message.

Because the IEF4501 message has no WHEN statement, the OTHERWISE statement traps and processes it. The IEF4501 message is to be displayed on the system console in white.

The REVISE action is very similar to the NetView PIPE EDIT stages. For more information on other possible revision actions, see the *Programming: NetView PIPEs* manual.

Note: Multiple REVISE statements are supported or combinable into one statement.

System console example



This example shows the MRT source from DSIPARM and the results on the system console. The MRT source contains four UPON groups. Several of the messages shown here have been revised with the MRT.

The text of the DSI802A message was altered completely. Without message revision, the DSI802A would look similar to the following text:

DSI802A *domainid* REPLY WITH VALID NCCF SYSTEM OPERATOR COMMAND

Important: Be careful when modifying the text of messages. The modified message text can be passed to automation. The automation might not function properly because of the text of the message.

The AOF568I message is red, yet an UPON group is coded for all AOF messages (PREFIX=AOF) to change the messages to reverse video blue. The UPON group for AOF568I uses a MSGID condition, but the UPON group for AOF messages uses the PREFIX condition. Because the MSGID condition is a *higher ranking* condition, it is tested and processed before the PREFIX condition.

Revision variables

IBM

Revision variables

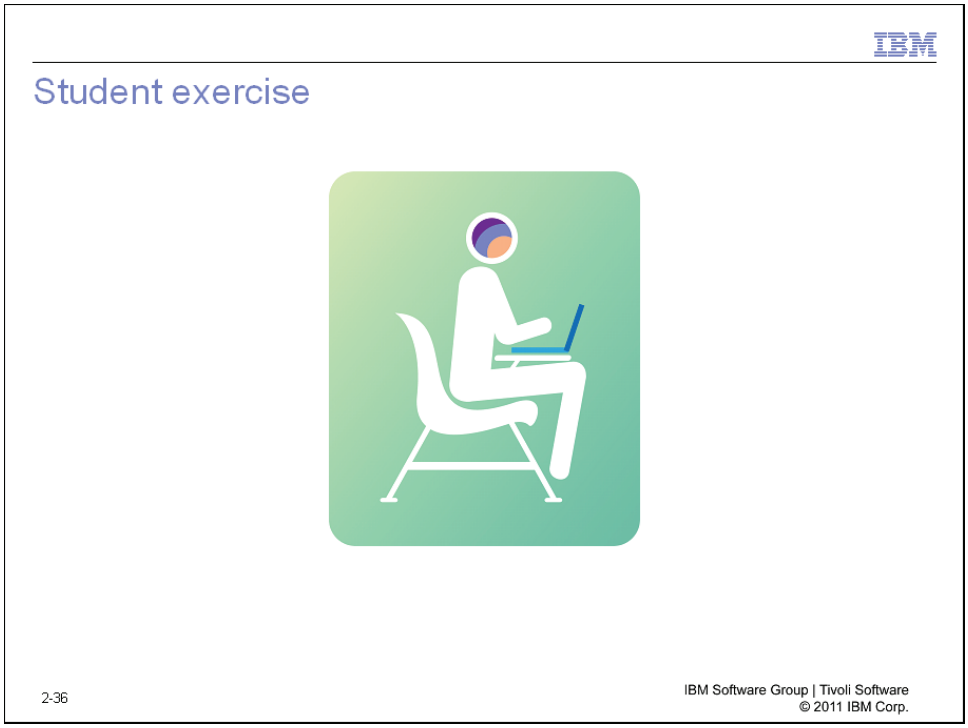
- Command SETRVAR and sample CNMSRVAR can be used for setting revision variables
- Revision variables can be resolved by MRT and CRT order, RVAR
- Allows to change the behavior of the revision tables without loading a different table
- CNMSMRT1 example shows CHRON can be used for setting variables that indicate shift or holiday

```
SELECT
WHEN (W2 ¬= /STATCOMP/) ! starting special proc? see next WHEN
WHEN ("SHIFT" RVAR = "NORMAL") !
NETVONLY=CNMSRVMC ! double use of sample clist!
WHEN ("SHIFT" RVAR = "HOLIDAY")
! cmd is allowed, no action here
```

2-35

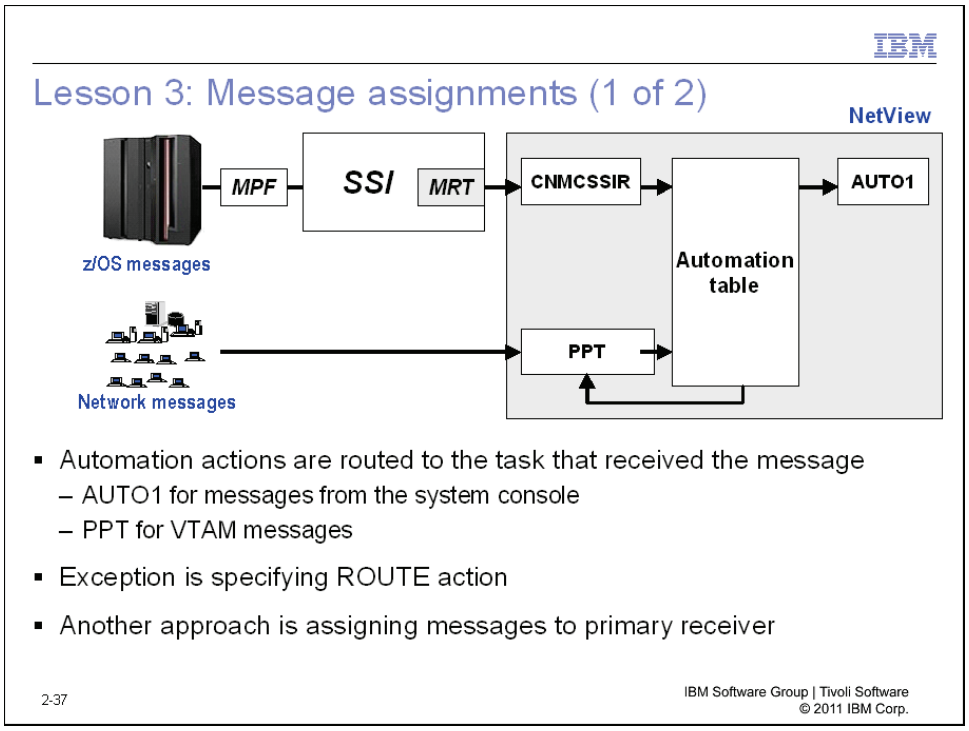
IBM Software Group | Tivoli Software

Student exercise



Open your *Student Exercises* book and perform Exercise 2-1.

Lesson 3: Message assignments



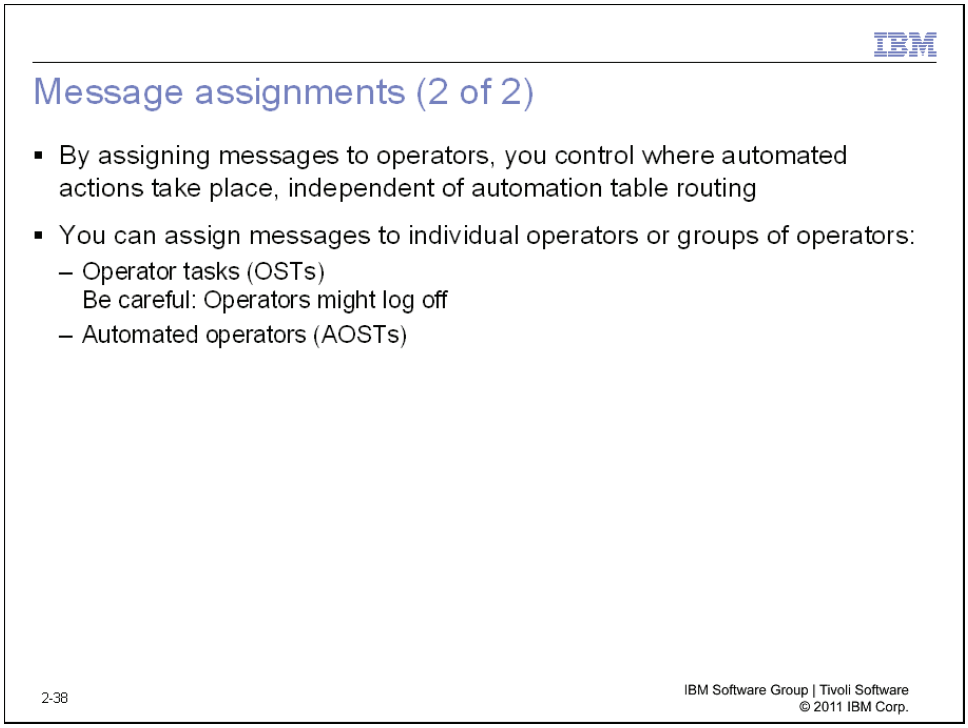
Messages that are matched in the automation table can result in commands being issued to take corrective actions. By default, the commands run on the task that receives the message. AUTO1 receives z/OS messages (because AUTO1 is the task that started CNMCSSIR) and the PPT task receives VTAM network messages. This approach can cause several problems:

- The task might not be active.
- The task might be active but busy, potentially creating a bottleneck.
- Not all tasks support all NetView commands. For example, the PPT task does not support commands that cause the task to wait.
- And so on

You can direct the action (and message) to one or more tasks with the ROUTE specification in the automation table. Or you can assign the message to one or more tasks. Tasks that are assigned a message are called *primary receivers* of the message.

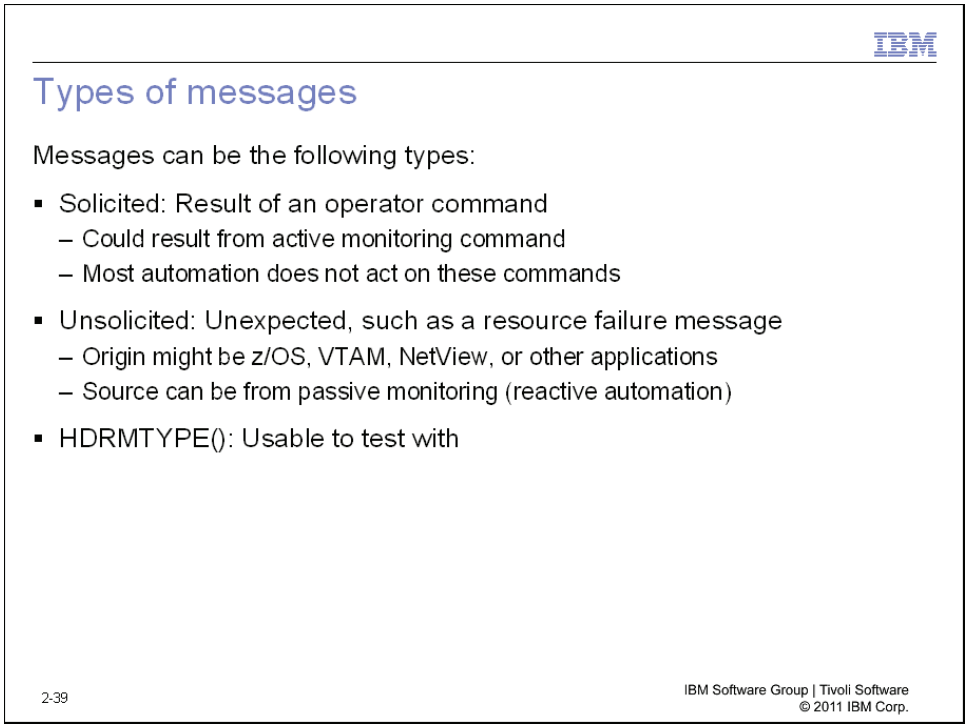
This lesson discusses assigning messages to NetView tasks. Customers can use a combination of the ASSIGN command and automation table ROUTE specification to control where to route messages and the actions to take. Discussion of the automation table ROUTE specification comes at the time of more detailed discussion of the automation table.

Message assignments (2 of 2)



The ASSIGN command is used for assigning messages to NetView operators and automation operators. With the ASSIGN command, you can define a default task for receiving the message (or group of messages). The primary receiver can be the default task for automation table actions, increasing the likelihood that the action routes to an active task and runs.

Types of messages



NetView distinguishes two basic types of message as follows:

- Solicited message

  A solicited message is a response to a command. Solicited messages route to the task that issues the command.

- Unsolicited message

  – An unsolicited message has no known destination. An unsolicited message typically indicates a problem. An examples might be a job start, a line problem, a transaction abend, or security violation.

  – An unsolicited message needs to be routed for review and action.

  – An unsolicited message is the source for reactive automation.

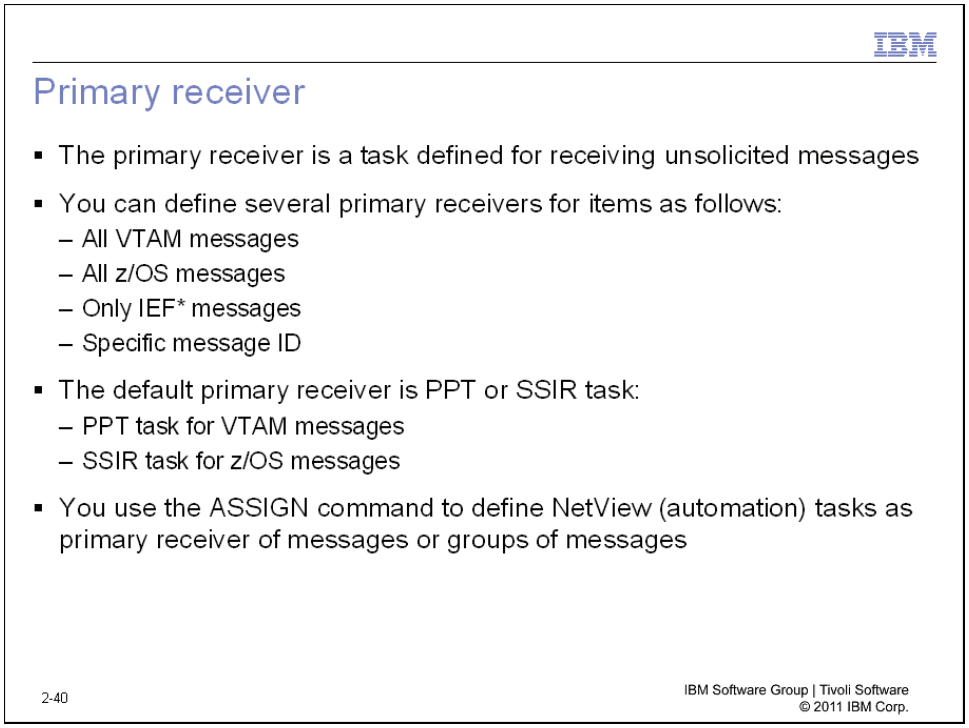
Unsolicited messages might come from z/OS while others come from VTAM and NetView itself. Be aware that unsolicited messages from z/OS might require different treatment than those that arrive from VTAM or NetView.

For a complete list of all possible HDRMTYPE values, see “Appendix G: NetView Message Type (HDRMTYPE) Descriptions”, in the *NetView for z/OS Automation Guide*.

Because NetView uses the concept of a known destination, some messages that seem unsolicited are treated as solicited. For example, if one NetView operator sends a message to another NetView operator, the message is treated as solicited because the destination is known.

Similarly, if a NetView user specifies the MVS VARY CONSOLE(*),ROUTE=1 command to receive unsolicited MVS messages of route code one, the treatment is different. NetView treats replies to this user as solicited messages.

Primary receiver

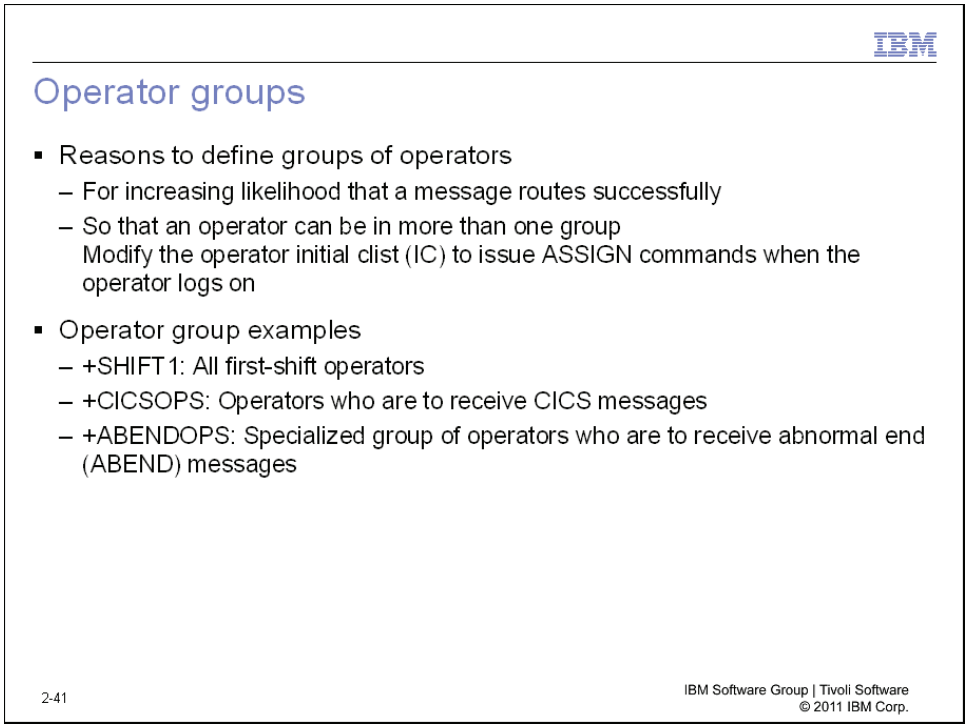


NetView uses the concept of a *primary receiver* to receive one or more unsolicited messages. You can define more than one primary receiver.

By default, the PPT task is the primary receiver of unsolicited VTAM messages, and the SSIR task is the primary receiver for unsolicited z/OS messages. The PPT task causes an error if EXECs try to wait for a response to a command.

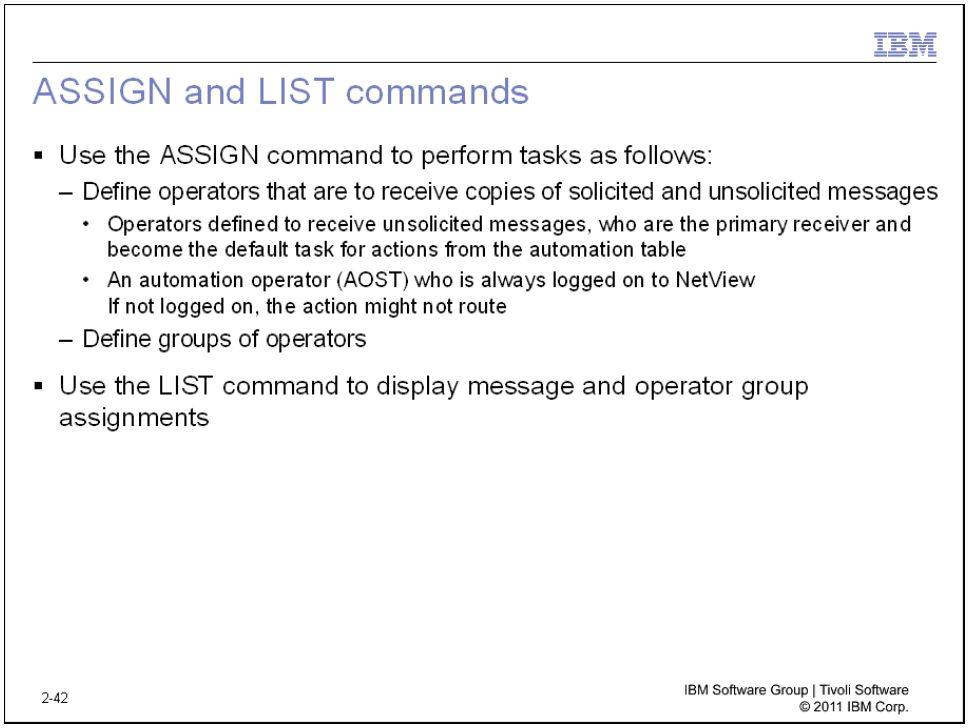
The ASSIGN command enables the definition of automation tasks as primary receivers to ensure that automated actions occur. You can also use the ASSIGN command for defining groups of operators as primary receivers. Because unsolicited messages are used for reactive automation, you define automation tasks as primary receivers.

Operator groups



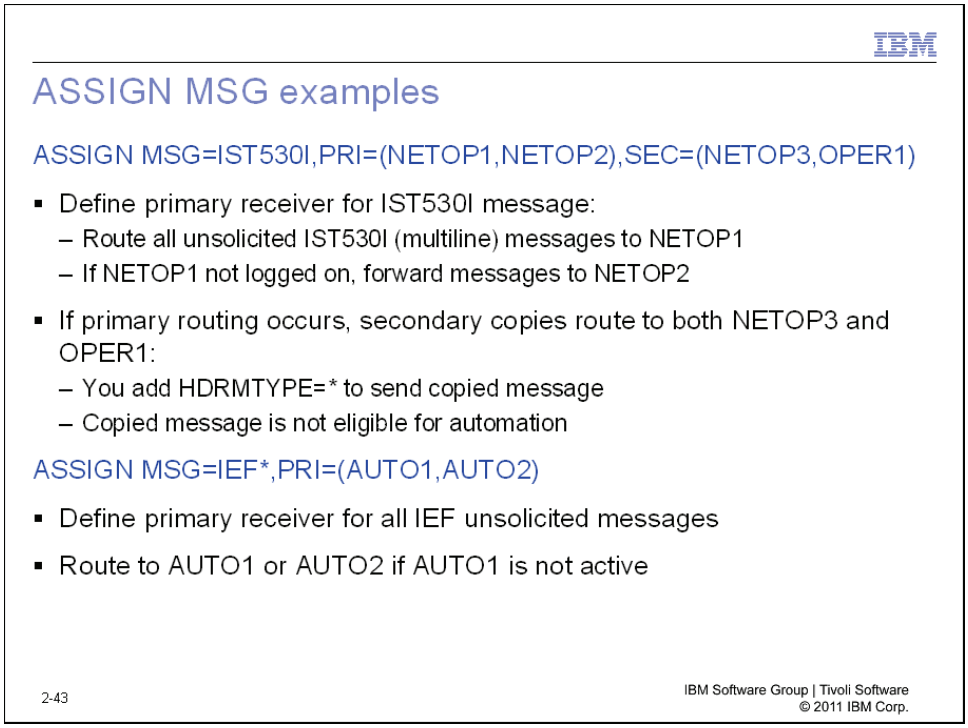
You can route automation actions to a group of operators. The operators in the group might change, but you do not need to change the automation table. For example, you can define a group of operators to process all CICS events. As NetView operators log on, they can be dynamically added to the +CICSOPS group. When their shift changes and they log off, they can be dynamically deleted from the group.

ASSIGN and LIST commands



You can use the NetView ASSIGN command to define primary receivers, operators who should receive copies of messages, and groups of operators.

ASSIGN MSG examples



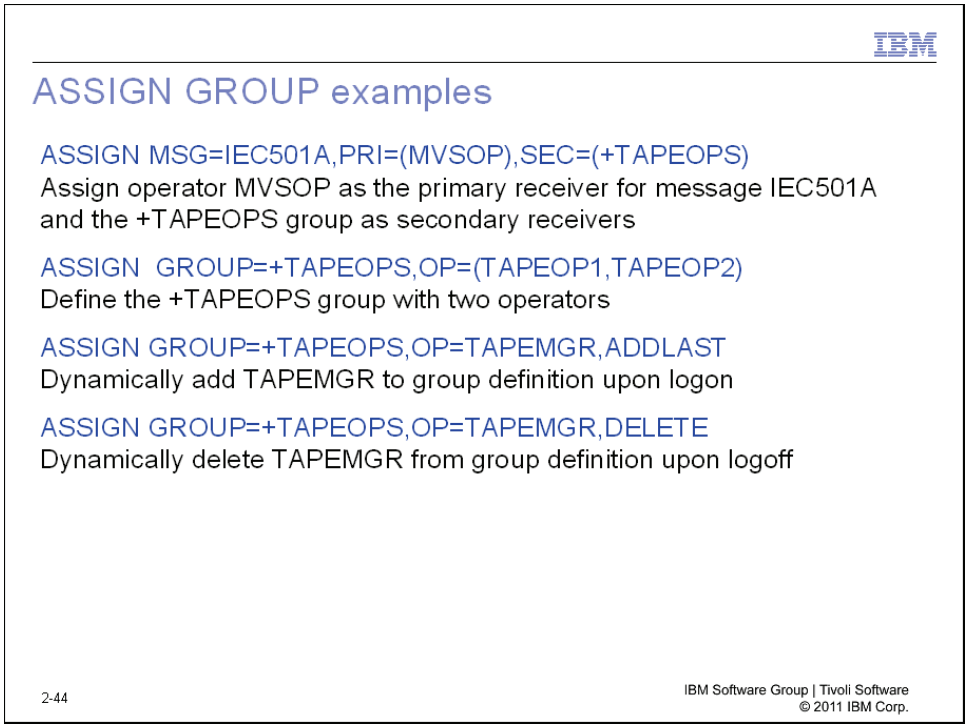
Use ASSIGN MSG to define primary receivers (PRI), secondary receivers (SEC), and also operators who should receive a COPY of the message.

The first example explains how to use PRI and SEC. NETOP1 and NETOP2 are defined as the primary receiver for IST530I. If NETOP1 is not logged on, NETOP2 receives the IST530I message. Secondary copies of the message route to NETOP3 and OPER1. If the IST530I does not route to a primary receiver, the secondary routing does not occur.

The second example defines AUTO1 and AUTO2 as the primary receiver for all IEF messages. The receivers are as follows:

- PRI: Defines a list of operators to receive unsolicited messages. (Defines the primary receiver for the message.)
  - Message routes to the first logged-on operator in the list.
  - Message is to be displayed with a percent sign (%) to identify the primary receiver.
- SEC: Defines a list of operators to receive copies of unsolicited messages.
  - Message routes to all logged-on operators if it was sent to the primary receiver.
  - Message is not automated.
  - Message is to be displayed with an asterisk (*) to identify it is a copy of an authorized message.
- COPY: Defines a list of operators to receive copies of solicited messages.
  - Message is not automated.
  - Message is to be displayed with a plus sign (+) to identify it is a copy.

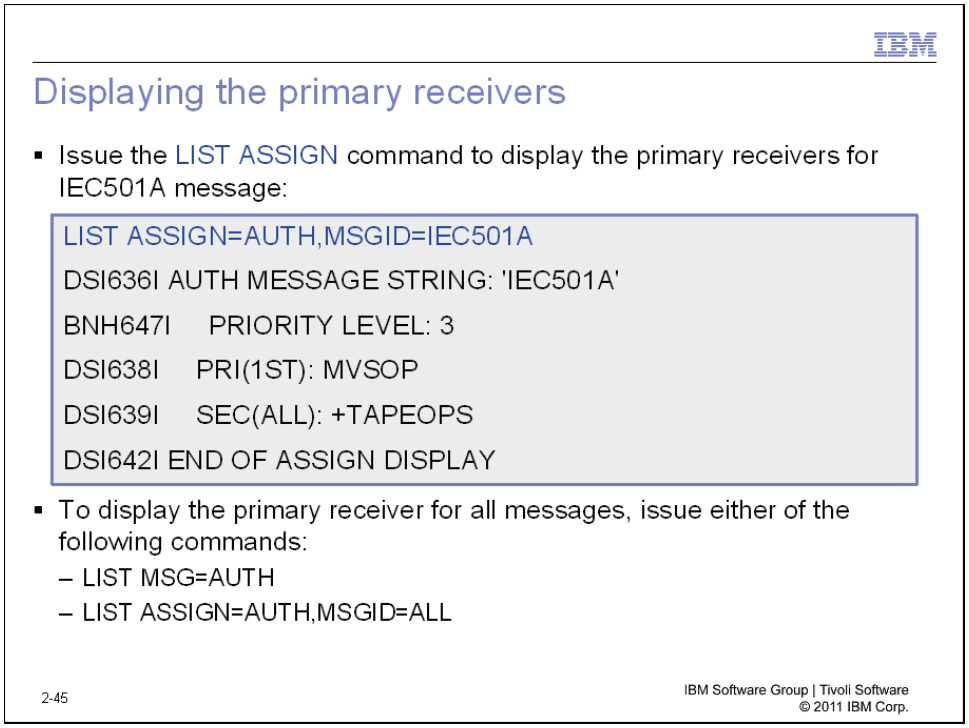
ASSIGN GROUP examples



Use ASSIGN GROUP to define operator groups. You can add operators to a group dynamically. The slide examples show how to route a message (IEC501A) to MVSOP as the primary receiver and an operator group, +TAPEOPS, as secondary receivers. The operator group can be defined before or after the ASSIGN MSG is issued.

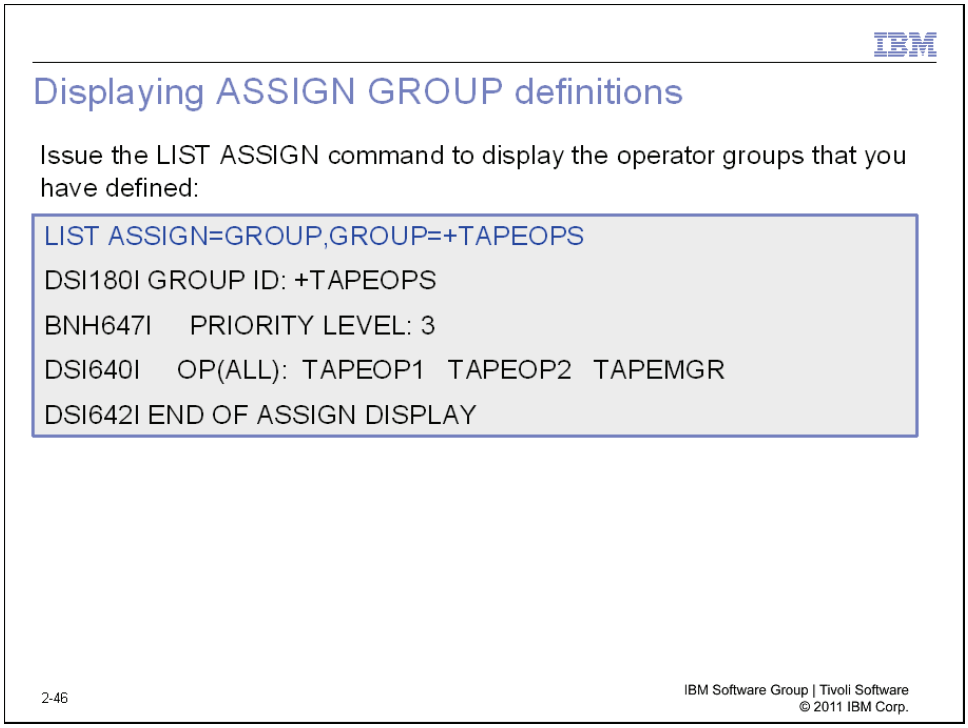
Initially, +TAPEOPS is defined with two operators: TAPEOP1 and TAPEOP2. When the manager, TAPEMGR, logs on, it is dynamically added to the +TAPEOPS group as the last task in the list. When TAPEMGR logs off, it is dynamically removed from the +TAPEOPS group.

Displaying the primary receiver



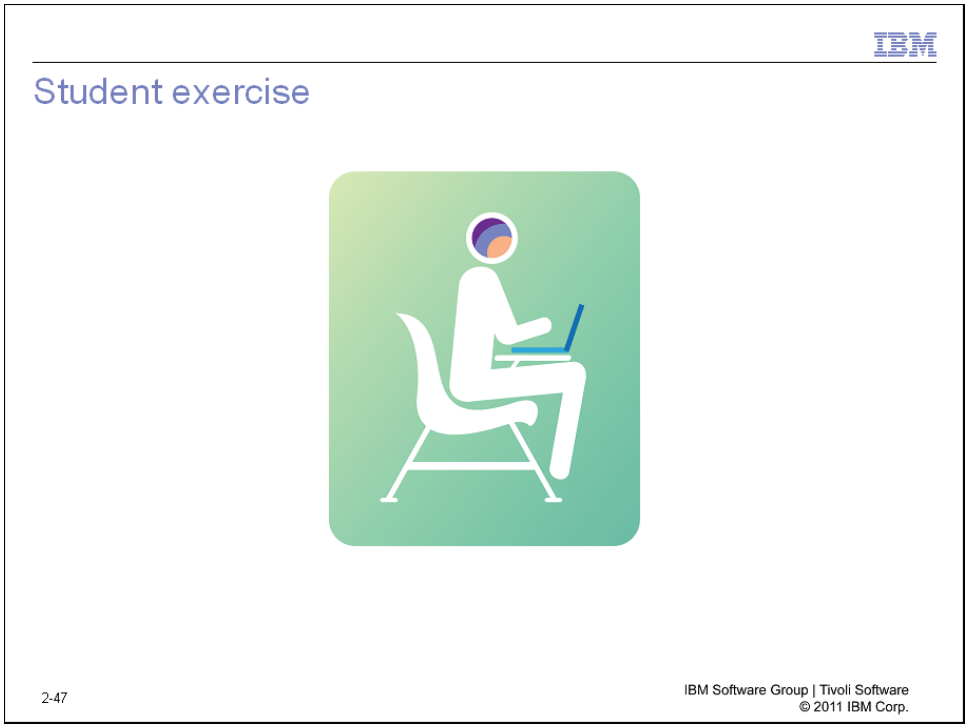
This slide example displays the message assignment for the IEC510A message that the previous slide use. The +TAPEOPS operator group is listed, but none of the operators.

Displaying ASSIGN GROUP definitions



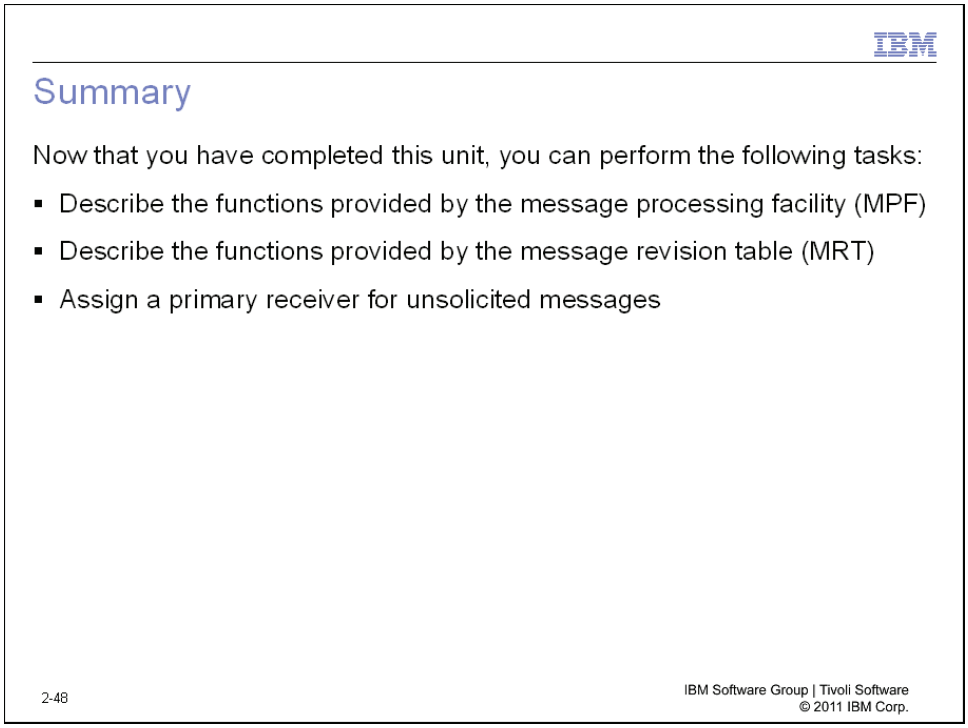
This slide example shows how to display the members of the +TAPEOPS operator group.

Student exercise

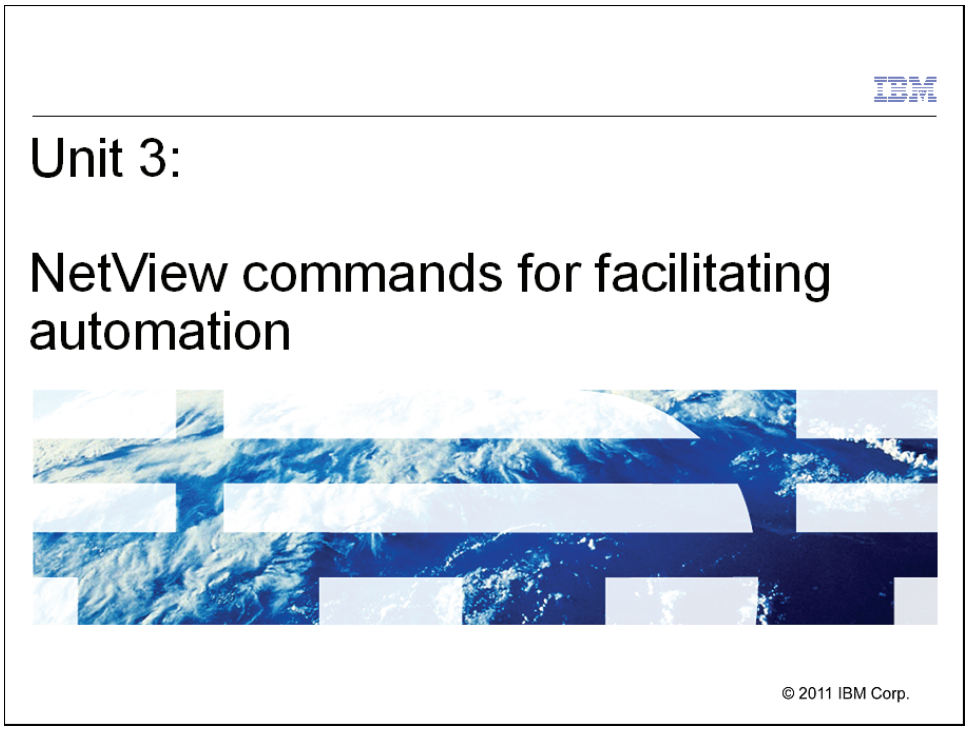


Open your *Student Exercises* book and perform Exercises 2-2 and 2-3.

Summary



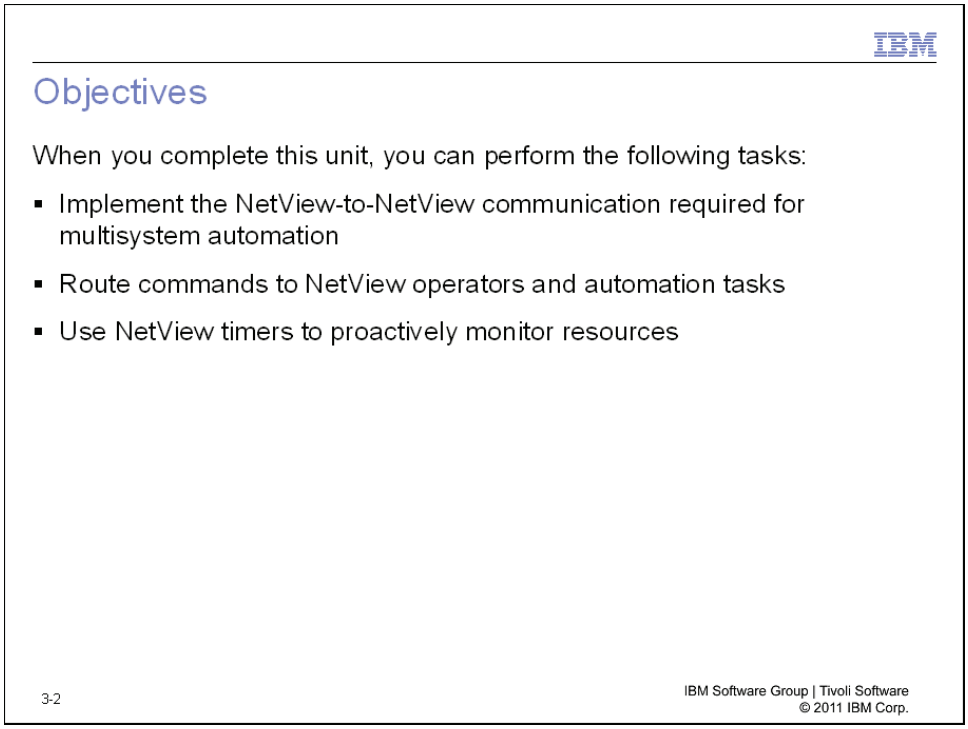
Unit 3: NetView commands for facilitating automation



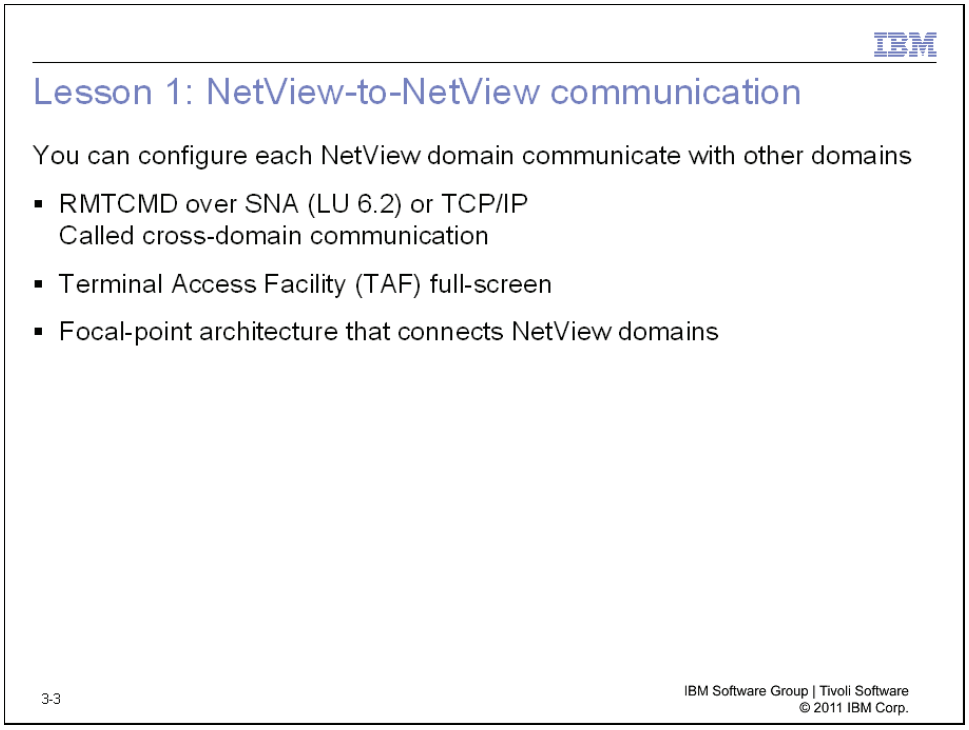
Introduction

This unit discusses the NetView commands that you can use for facilitating automation. Commands include those for establishing communication to other NetView domains and other applications, and commands for scheduling and managing timers. Using timers is very important when monitoring resources.

Objectives

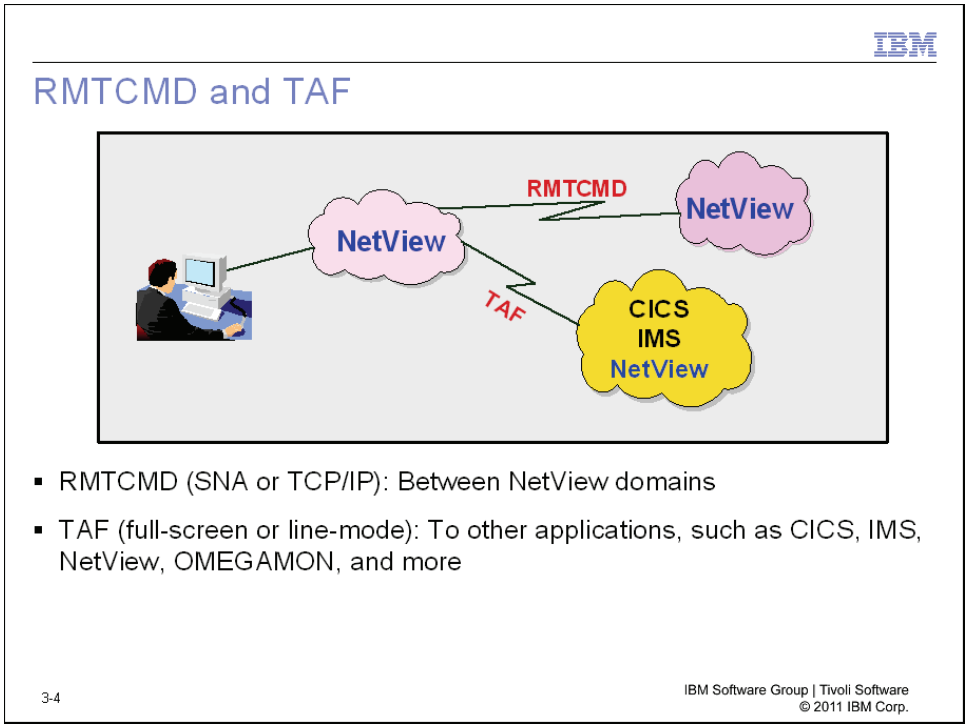


Lesson 1: NetView-to-NetView communication



This lesson focuses on communication between NetView domains, commonly called *cross-domain* communication, and other applications. These connections are key in implementing multisystem automation. This lesson also includes discussion about RMTCMD, TAF, and focal point architecture discussions.

RMTCMD and TAF

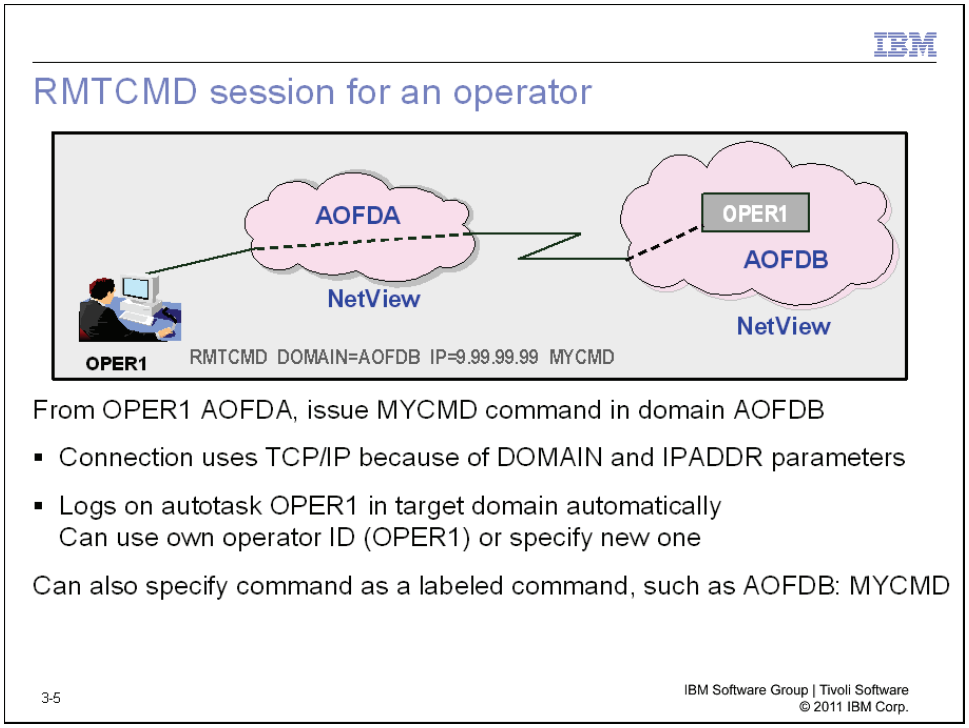


RMTCMD and TAF connections to another system provide the basis for multisystem automation. RMTCMD establishes connections to remote NetView domains. You can use TAF sessions for establishing connections to applications, such as IMS, CICS, and OMEGAMON.

TAF supports both full-screen and line-mode sessions. You can use line-mode sessions for receiving messages and issuing commands directly to the target application. *Screen scraping* automation uses full-screen sessions for displaying panels on VOSTs and interrogate fields on the panels. The TAF commands are as follows:

- BGNSESS: Start a session (full-screen or line-mode).
- SENDSESS: Route commands over a line-mode session.
- LISTSESS: Display the active TAF sessions.

RMTCMD session for an operator



The RMTCMD command sends commands to another NetView domain by using either a SNA LU6.2 or TCP/IP session. Both IPv4 and IPv6 are supported.

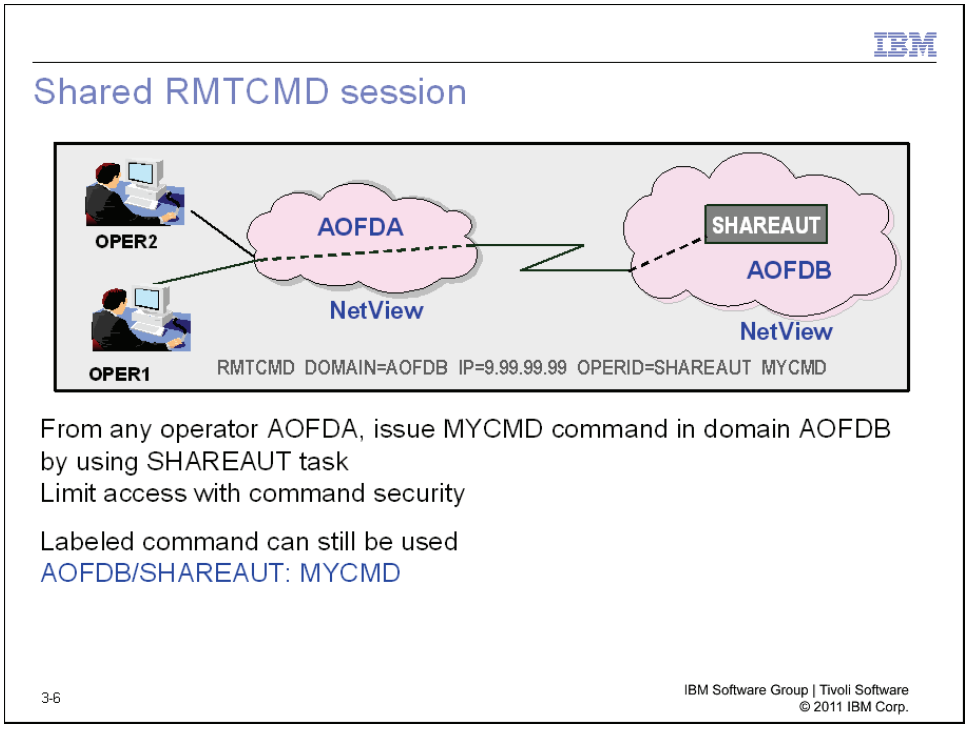
If a specific operator task is specified and the task is active, that task is used for the command. (The operator task might be logged on at the remote domain, or the task might be an autotask,) If the operator is not active in the target domain, it is logged on as an autotask before the command runs. Whenever a RMTCMD command is issued to a new domain and autotask combination, a remote session begins for that user.

If a RMTCMD is issued to a NetView domain but no operator ID is specified, an existing session is used. If the caller does not have an existing session, a new session starts, using the operator ID of the caller.

Note: The caller must be defined in the DSIOPF member at the target NetView.

Labeled commands are an abbreviated format for issuing RMTCMD commands. To route the MYCMD command to NetView AOFDB domain, issue the AOFDB: MYCMD command. You use an existing session or start a new session by using the operator ID of the caller.

Shared RMTCMD session

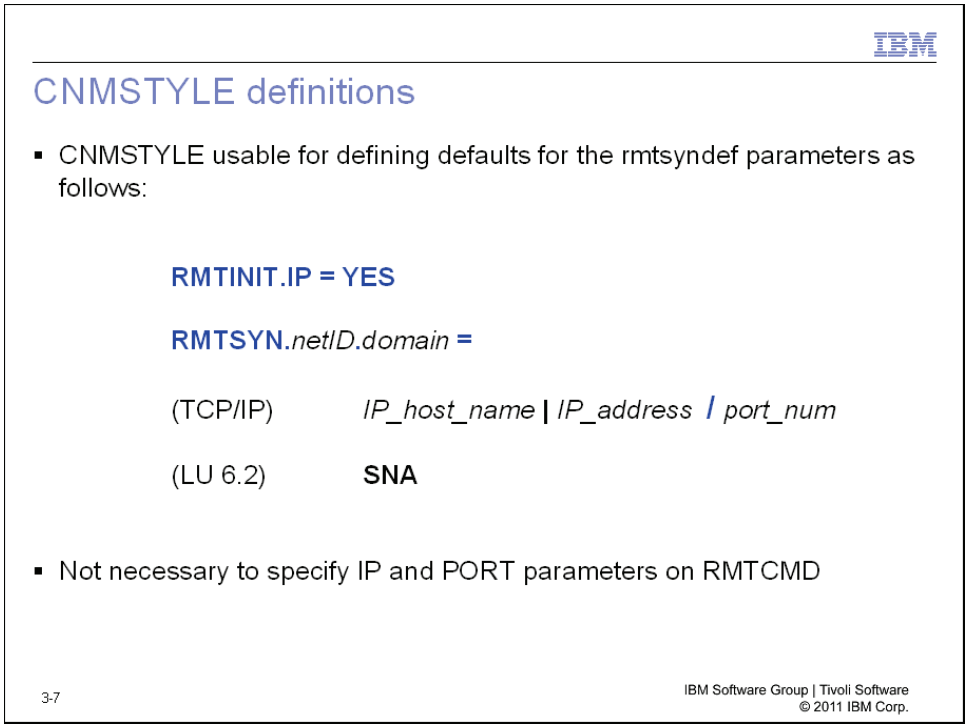


Several users from several source domains might route requests to a single domain and autotask combination. Such requests queue and run serially at the target domain and might not run immediately. Or the requests might not run at all. Results from such executions return to the caller as a single block of messages when the command has finished.

In the example, correlated responses return to the originating operator (OPER1 or OPER2). All operators at source systems (for example, AOFDA domain) can use the SHAREAUT autotask for their RMTCMD session to the AOFDB domain.

Labeled commands are supported. To run the MYCMD command on the SHAREAUT task in the NetView AOFDB domain, issue the AOFDB/SHAREAUT: MYCMD command.

CNMSTYLE definitions



CNMSTYLE contains two statements for defining RMTCMD communication as follows:

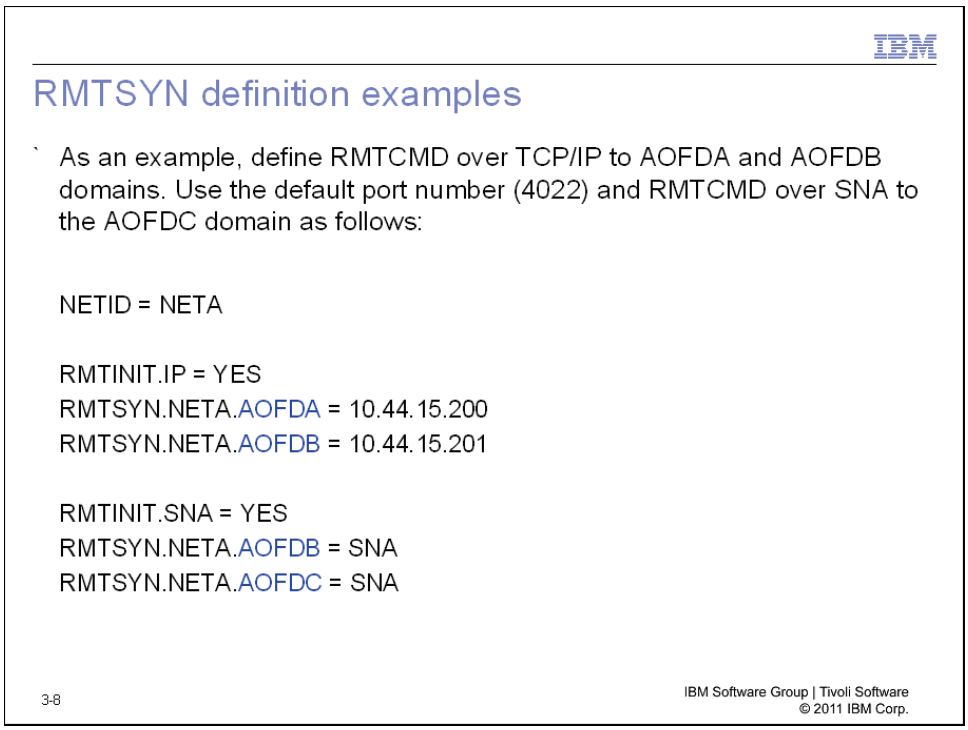
- The RMTINIT statement enables IP or SNA (not shown on slide) over RMTCMD.
- The RMTSYN statement defines RMTCMD connectivity (SNA LU 6.2 or TCP/IP) between NetView domains.

  The RMTSYN statement simplifies use of RMTCMD over TCP/IP. You do not need to specify an IP address on your RMTCMD command if it is defined on a RMTSYN statement. For example, RMTCMD DOMAIN=*domain*,*command* uses the IP address and port parameters from a RMTSYN statement.

Note: RMTCMD requires NetView tasks as follows:

- DSIHPDST and DSIUDST for SNA LU 6.2
- DSITCPIP for TCP/IP communication

RMTSYN definition examples



This slide shows examples of using RMTSYN to define RMTCMD sessions:

- RMTCMD over TCP/IP is enabled. The target IP address for two domains (AOFDA and AOFDB) is defined.

Note: You can start RMTCMD sessions to other domains that are not in the RMTSYN definitions. RMTSYN is not required. If you do not define a RMTSYN statement, you must specify all parameters on your RMTCMD when starting the session.

- RMTCMD over SNA LU 6.2 is enabled to two domains (AOFDB and AOFDC).

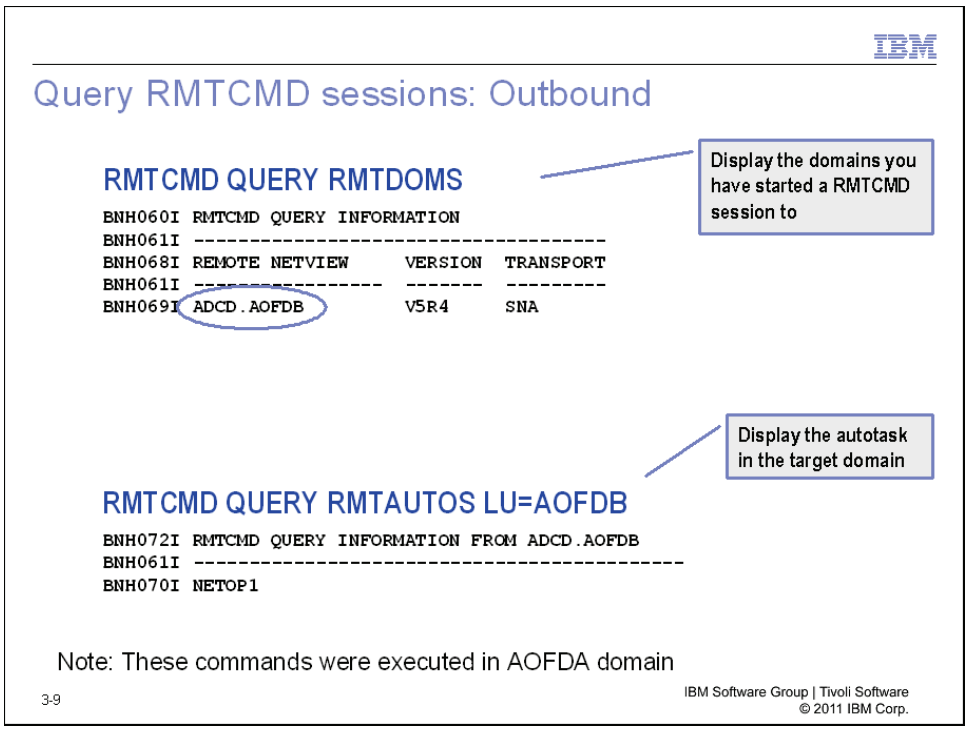
To start a RMTCMD session to domain AOFDB by using the same operator ID, enter **RMTCMD** DOMAIN=AOFDB,*command*.

If not specified, RMTCMD over TCP/IP connections use the port that is defined for the DSIUDST NetView task (default of 4022). An example follows:

```
LIST DSIUDST
TYPE: OPT TASKID: DSIUDST  TASKNAME: DSIUDST  STATUS: ACTIVE
MEMBER: DSIUINIT SECURITY: NONE  AUTH: SENDER
HOSTNAME=TIVED1                       HOSTID=10.44.15.200
PORT=4022
SOCKETTYPE=AF_INET
LOADMOD: DSIZDST
Task Serial:  721
Messages  Pending:  0  Held:  0
WLM Service Class:  Not Available
END OF STATUS DISPLAY
```

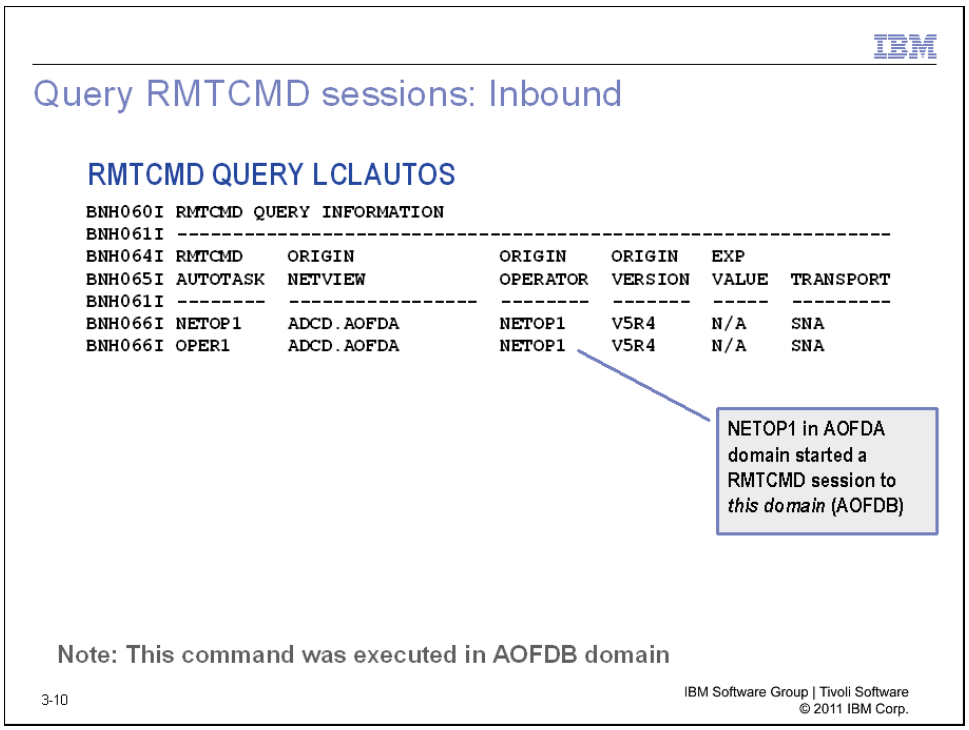
See the *Administration Reference* manual for more details.

Query RMTCMD sessions: Outbound



RMTCMD QUERY displays session data for both SNA LU 6.2 and TCP/IP sessions.

Query RMTCMD sessions: Inbound

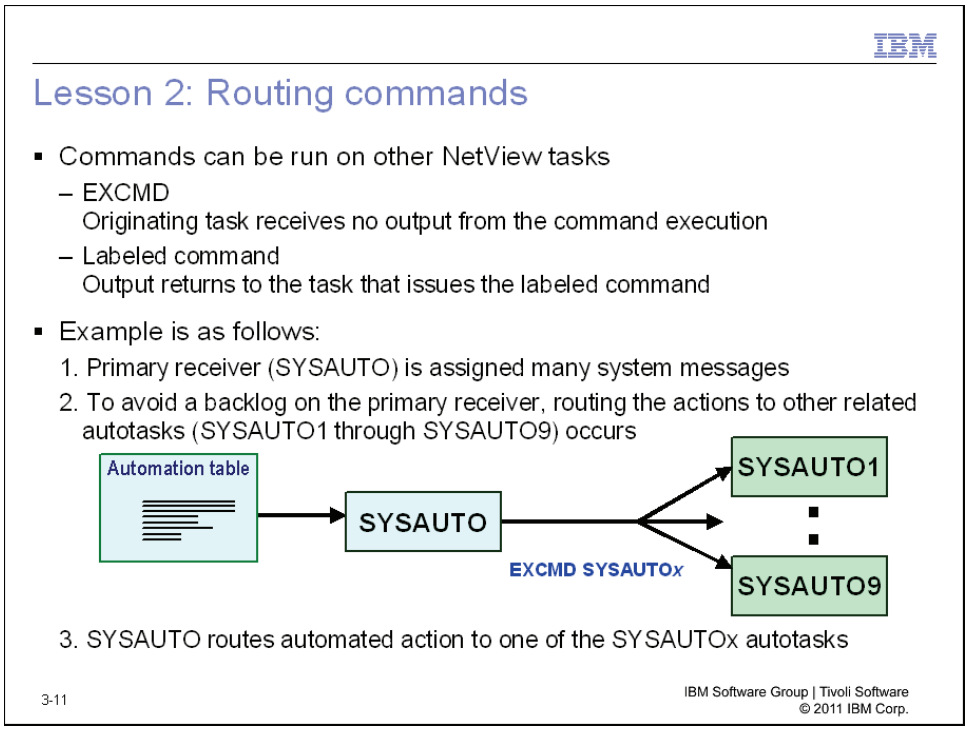


The slide shows two RMT sessions, both started by NETOP1 in AOFDA. The first one uses NETOP1 as user in AOFDB, while the second one uses OPER1.

```
RMTCMD DOMAIN=AOFDB,OPERID=XXXX,MYCMD
```

If you do not specify an OPERID, the operator ID of the caller is used.

Lesson 2: Routing commands



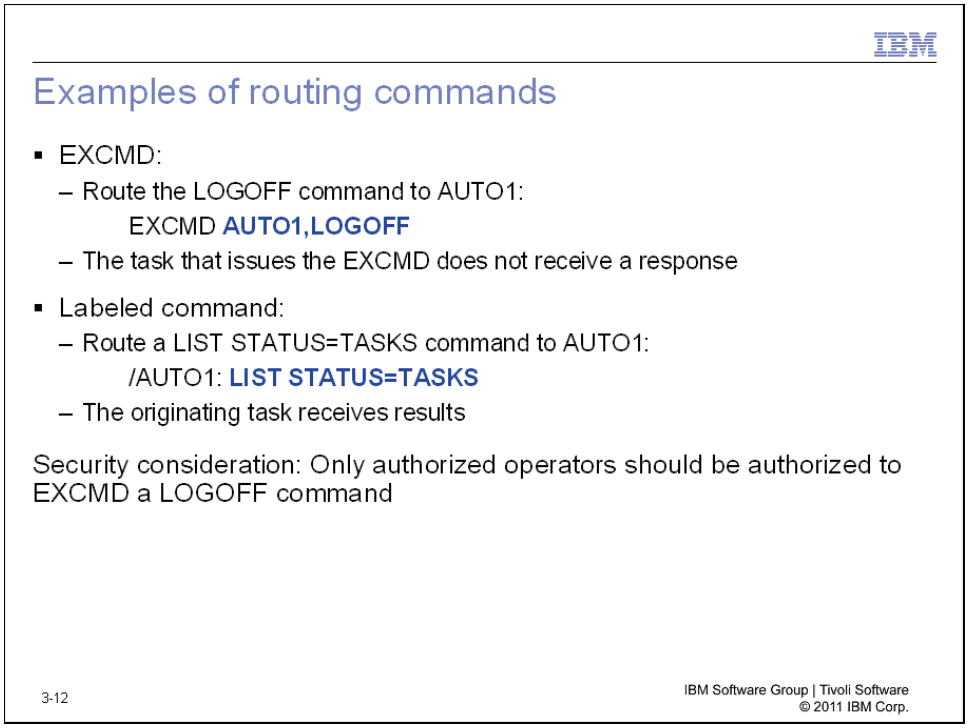
NetView provides the EXCMD command, usable to route a command to another task. The specified command runs if the task is authorized.

The slide shows an example of EXCMD use. Suppose you have a SYSAUTO automation task that is defined as the primary receiver of a group of messages. The messages do not have to be related. Because SYSAUTO is assigned to receive the messages, the actions that are defined in the automation table run under SYSAUTO, by default. You can use SYSAUTO to act a *router task* by routing the actions to a pool of automation tasks: SYSAUTO1 through SYSAUTO9 in this case.

This approach is usable for reasons as follows:

- The logic in the automation table is simplified to route the messages and actions to the default automation task, SYSAUTO.
- SYSAUTO checks for the next available SYSAUTO*xx* automation task and routes the actions. Distributing the automation across multiple automation tasks affects overall throughput.
- By separating the routing function from the actions being routed, you can avoid congestions on the automation tasks that perform the automation.

Example of routing commands



As an example, OPER1 issues the following command:

EXCMD AUTO1,LOGOFF

This command causes the AUTO1 task to log itself off by running the LOGOFF command. This example also points out that the use of EXCMD can be a security issue. You should control the tasks that have access to issue an EXCMD and also the commands that route with EXCMD. You can also use AUTHCHK=SOURCEID to have the command checked under the origin task. See the NetView *Security Reference* manual for more details on command security.

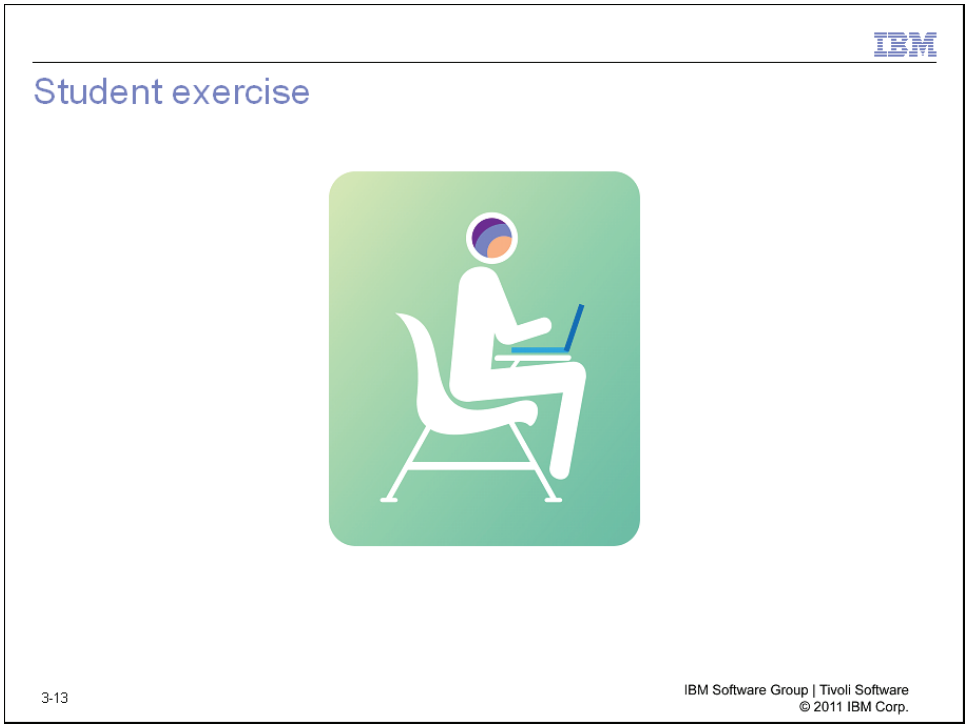
When EXCMD is used, no output from the routed command returns to the task that routes the request (OPER1, in this example). You can use *labeled commands* to see the output.

As an example, OPER1 issues the following labeled command:

/AUTO1: LIST STATUS=TASKS

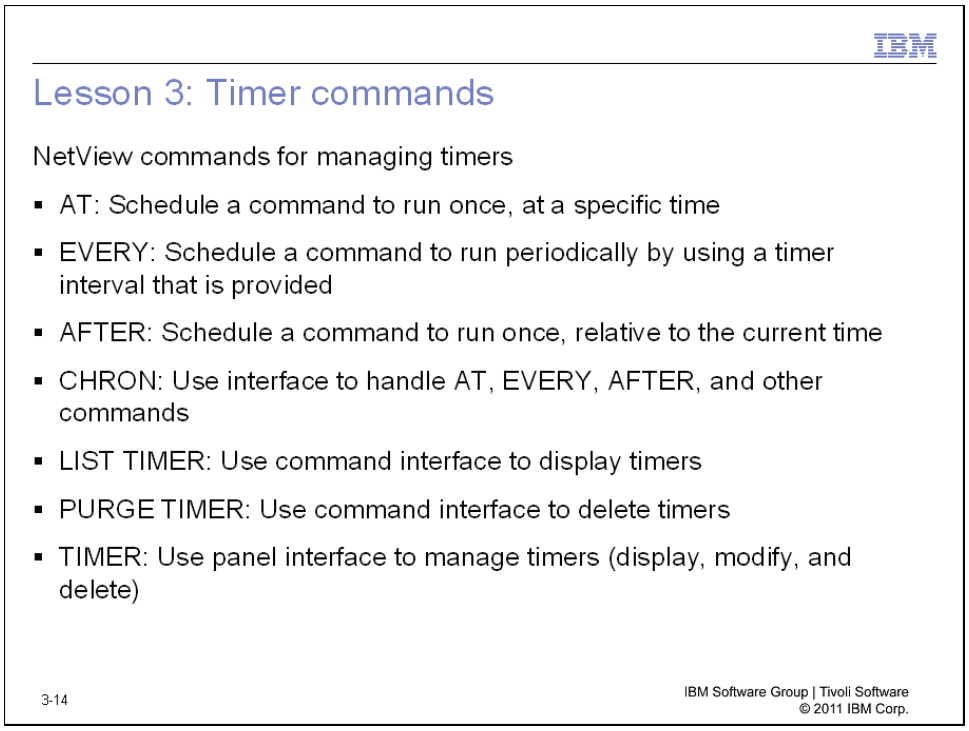
The LIST command runs on the AUTO1 task, but the command responses are displayed to OPER1.

Student exercise



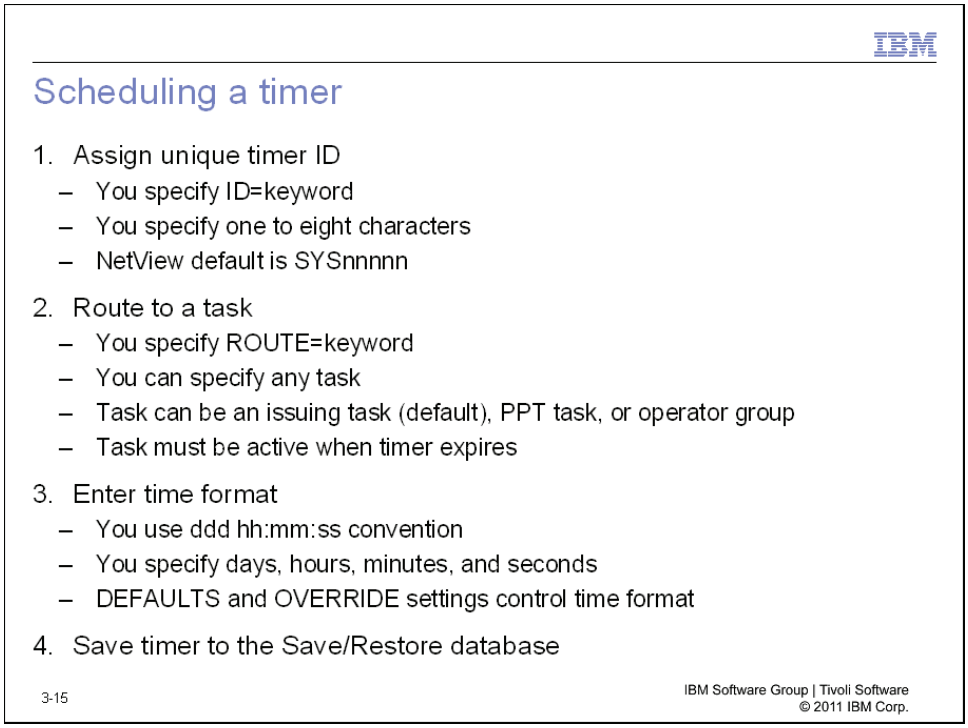
Open your *Student Exercises* book and perform Exercise 3-2.

Lesson 3: Timer commands



This lesson focuses on NetView commands for scheduling actions, commonly referred to as *timer commands*. They are the basis for proactive automation. You use timers to schedule commands for collecting resource information.

Scheduling a timer

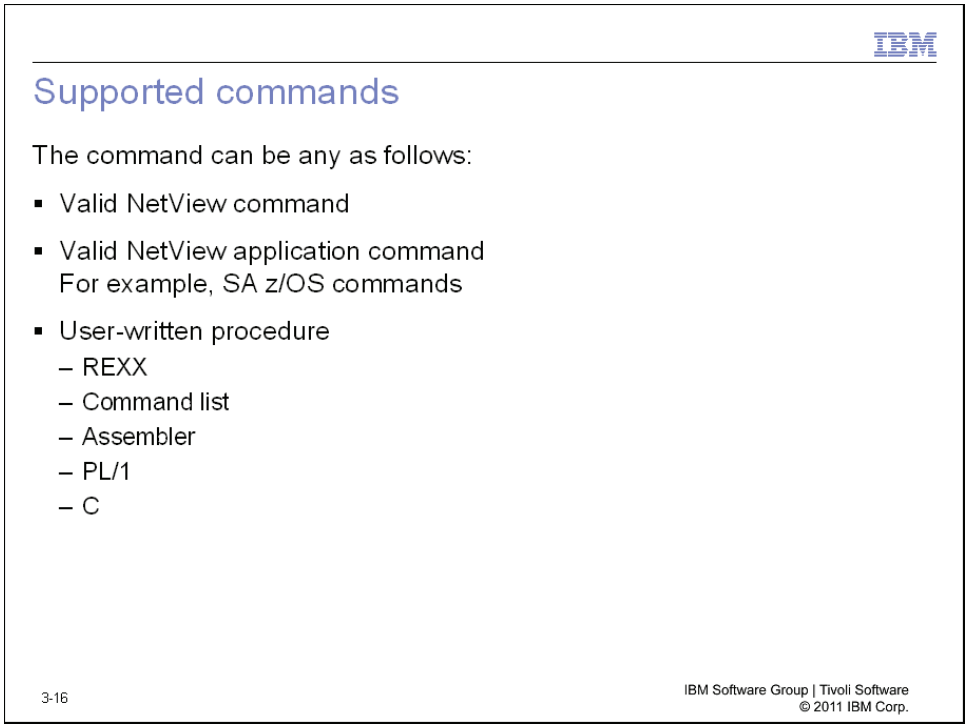


Each timer has an associated timer ID that can be up to eight characters long. The timer ID must be unique. If you do not specify a timer ID, NetView generates one that uses a prefix of SYS. Operators cannot specify a timer ID that uses SYS for a prefix.

You can schedule timers on the task that issues the timer command or on another NetView task by specifying the ROUTE operand. The task does not need to be active when you schedule the timer (AT, EVERY, AFTER, CHRON). However, the task must be active to run the command when the timer expires. Security can be performed on the origin or target task.

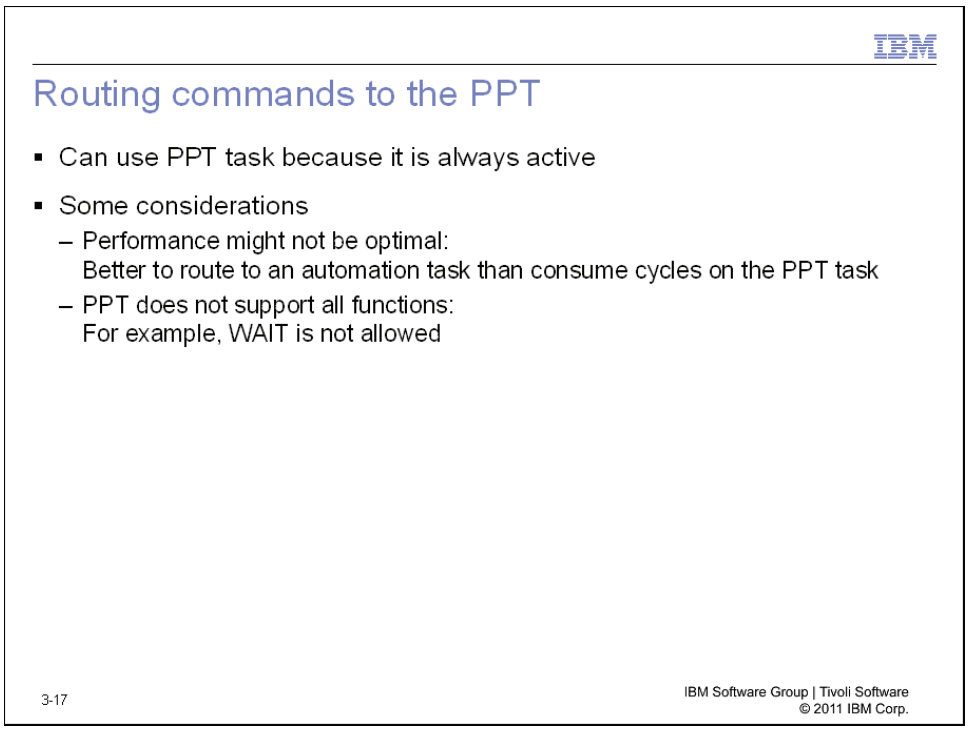
Timers can be saved to the NetView Save/Restore database (using task DSISVRT) and restored when NetView re-initializes. This action prevents having to set the timer every time NetView starts.

Supported commands



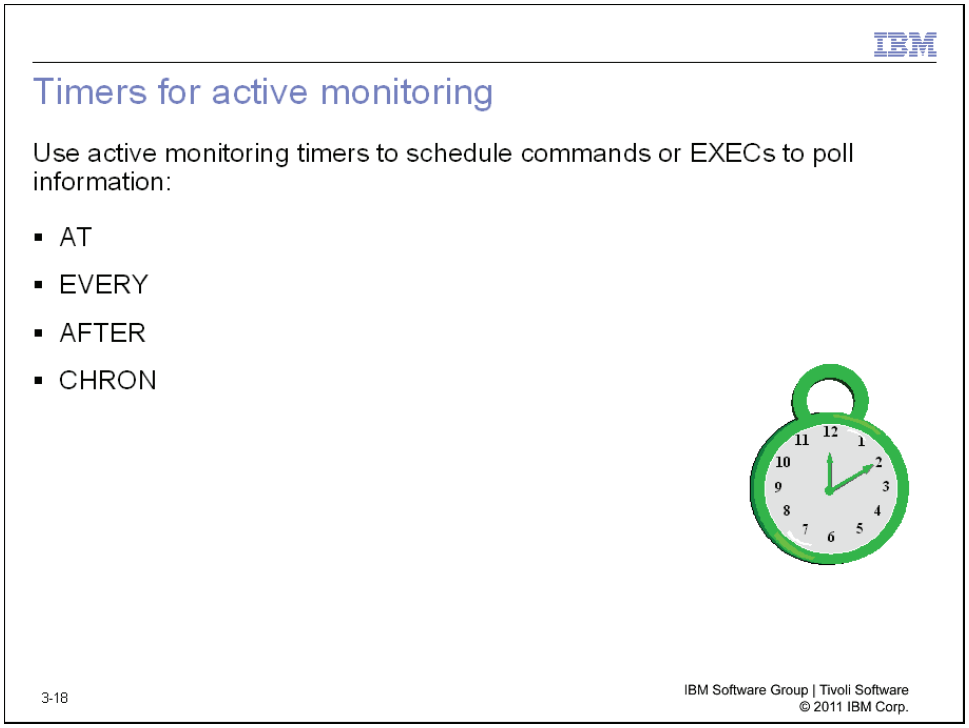
You can schedule timers to issue any valid NetView command, including commands from products such as SA z/OS and user-written commands.

Routing commands to the PPT



You can schedule timer commands to run under the NetView PPT task. Limit the amount of activity on the PPT task because the task is vital for several NetView functions.

Timers for active monitoring



Active monitoring, also known as proactive monitoring, uses NetView timers to schedule the polling of resource information. Characteristics of polling timers are as follows:

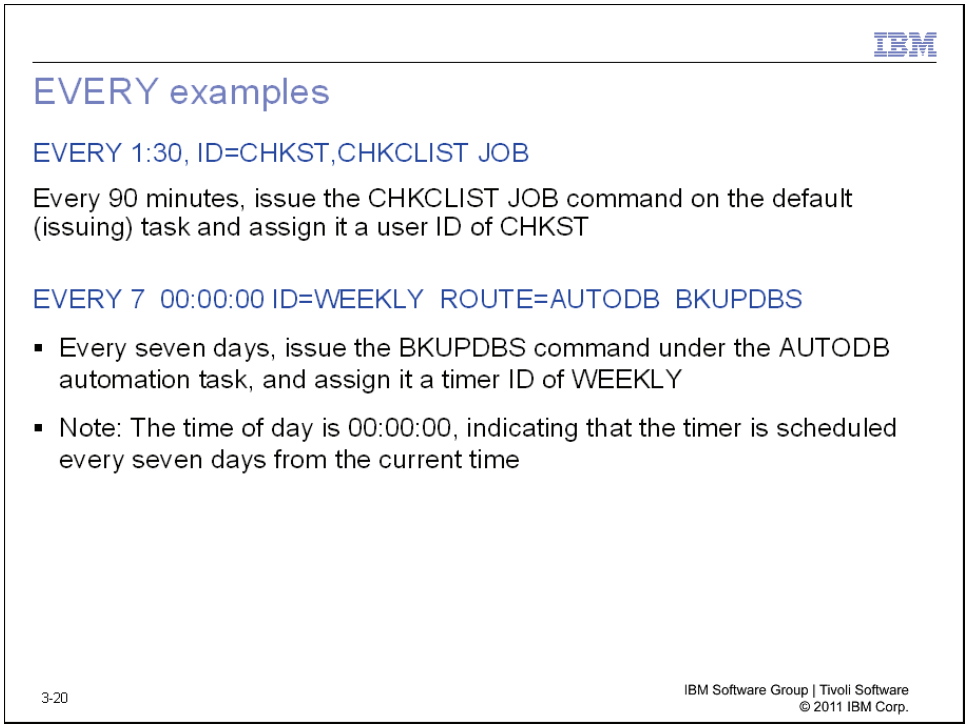
- The EVERY timer is the most common timer for active monitoring. In some cases where it might take a long time to retrieve and process the polling information, an AFTER timer is also used.
- The AFTER timer is scheduled after each polling interval completes.
- An AT timer is used for scheduling a command to run once daily. An example is performing a database backup and archive at midnight.
- A CHRON timer provides the same functions as AT, EVERY, and AFTER, but with additional features.

AT examples

AT timers run a command one time at a specific time. Examples are as follows:

- At midnight
- At 6 p.m.

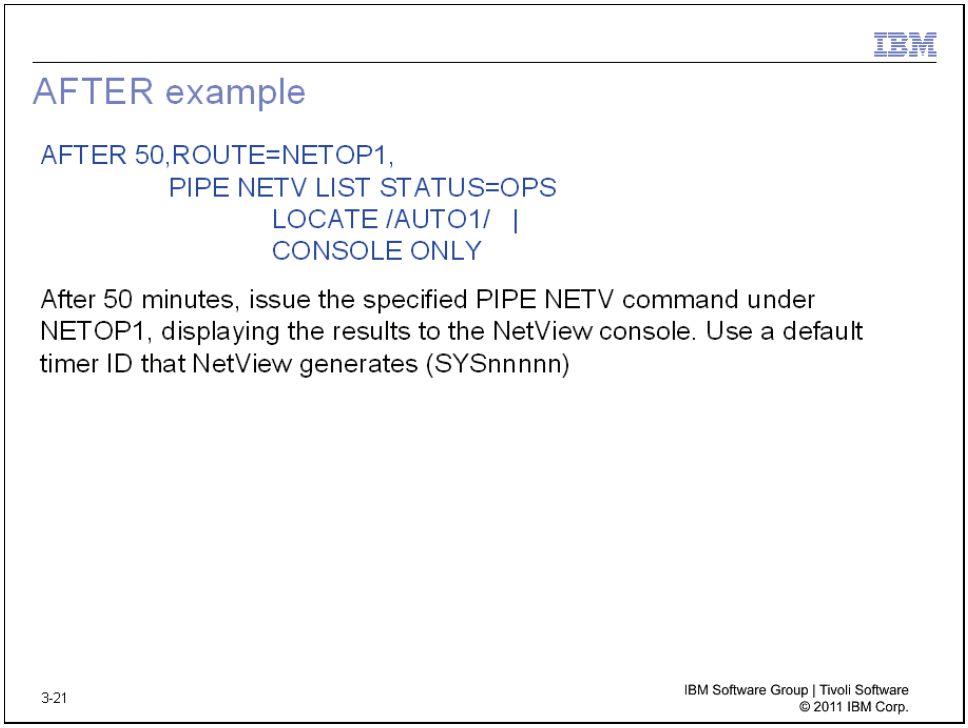
EVERY examples



EVERY timers run a command more than one time, using the time interval provided until the timer is purged or NetView shuts down. Examples are as follows:

- Every 5 minutes
- Every 24 hours

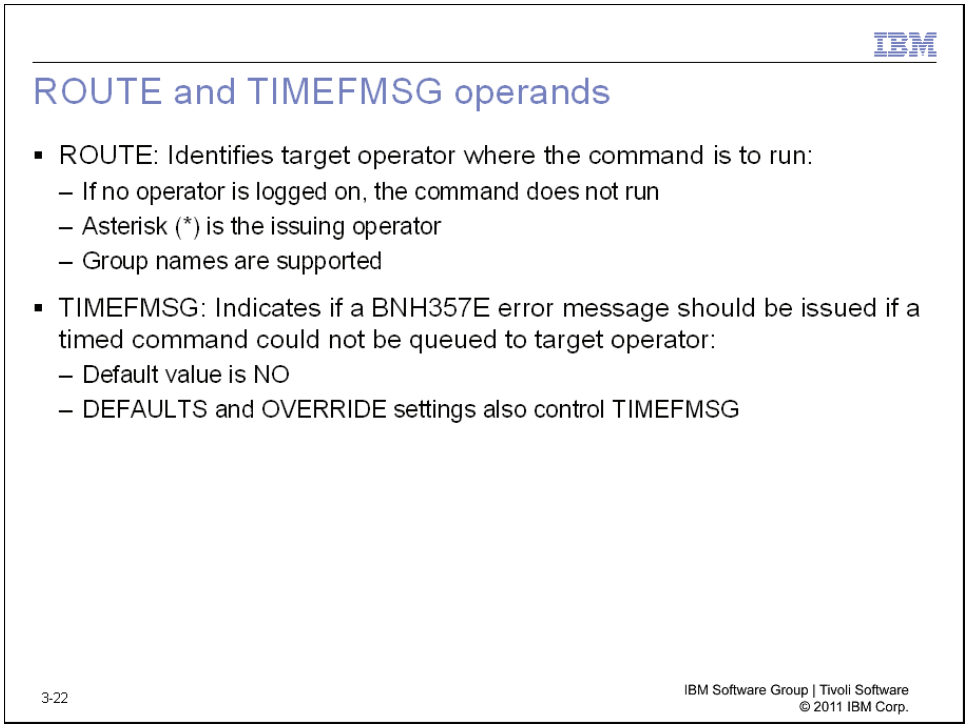
AFTER example



AFTER timers run a command one time relative to the current time. Examples are as follows:

- After five seconds
- After ten minutes

ROUTE and TIMEFMSG operands



The ROUTE parameter of a timer command (AT, AFTER, EVERY) specifies the operator task where the timed command is to run. For example, the following command runs on the OPER1 task (if OPER1 is logged on at the specified time):

```
AT 12/31 23:59:59,ROUTE=OPER1,MSG ALL,'Happy New Year'
```

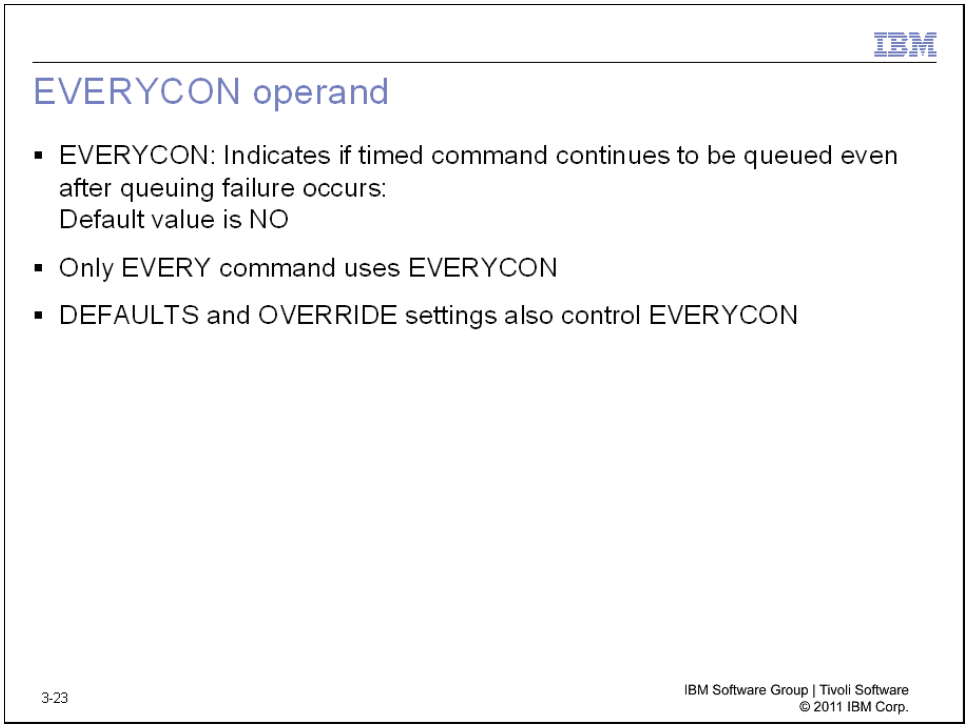
A single operator or a group of operators can be specified. ROUTE=* indicates the issuing operator and is the default. A group operand indicates that the first logged-on operator in the list of group members runs the command. If no valid operator is logged on, the command is not issued.

Security can be performed on the origin or target task.

If the command is unable to be queued to the target task, specify the TIMEFMSG parameter to display a BNH357E message that the timer command failed to start. Details are in the message to identify why the command failed.

```
BNH357E  TIMER COMMAND FAILED TO START. TYPE: type ID: id TASK:
task MQSRC: mqsrc INTERVAL: int1 int2 CONTINUE: cont CMD: cmd
```

EVERYCON operand



If the timed command fails for an EVERY timer, specify the EVERYCON parameter to continue scheduling the EVERY timer (EVERYCON=YES). An example follows:

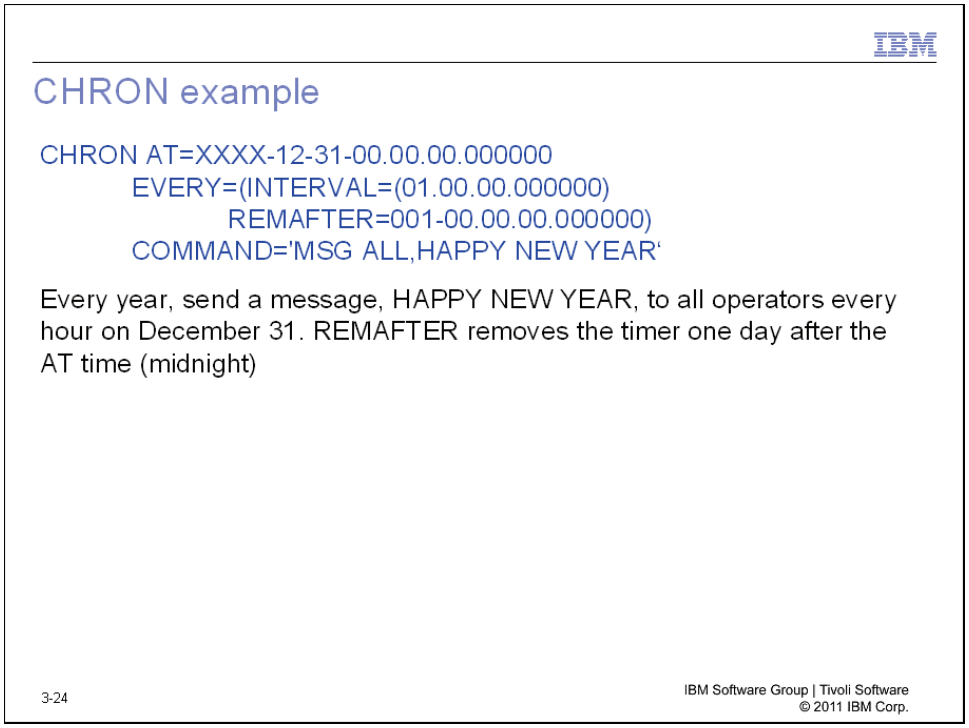
```
EVERY 1:30,EVERYCON=YES,ID=CHKST,CHKCLIST JOBS
```

The default value for the EVERYCON parameter is defined in CNMSTYLE as follows:

```
DEFAULTS.EVERYCON = yes
```

You can change the default value with the DEFAULTS and OVERRIDE commands.

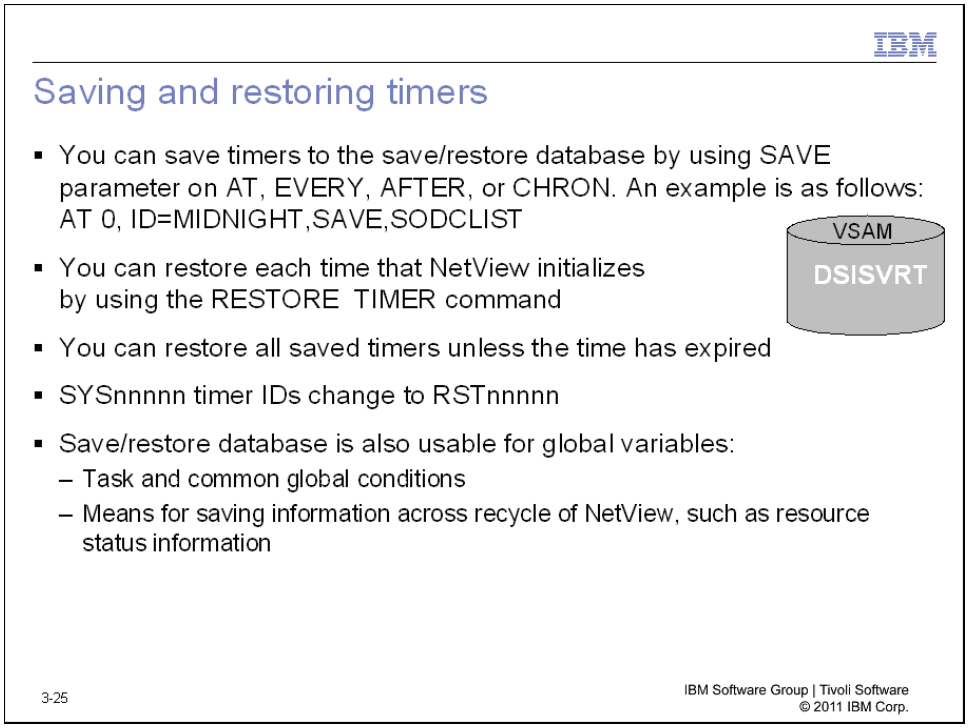
CHRON example



You can use the CHRON command to schedule more complex timers. In this example, December 31 is a day that is defined in the DSISCHED database:

```
XXXX-12-31 NEW_YEARS_EVE,HOLIDAY
```

Saving and restoring timers



When NetView ends, all timers that were created by earlier AT, EVERY, and AFTER commands are lost unless they were saved to the Save/Restore database. (This database is a VSAM file.) When NetView restarts, any saved timers can be restored, removing the need to re-issue the original commands.

To save a timer, use the SAVE operand on the timer command, which must precede the command to be issued. An example follows:

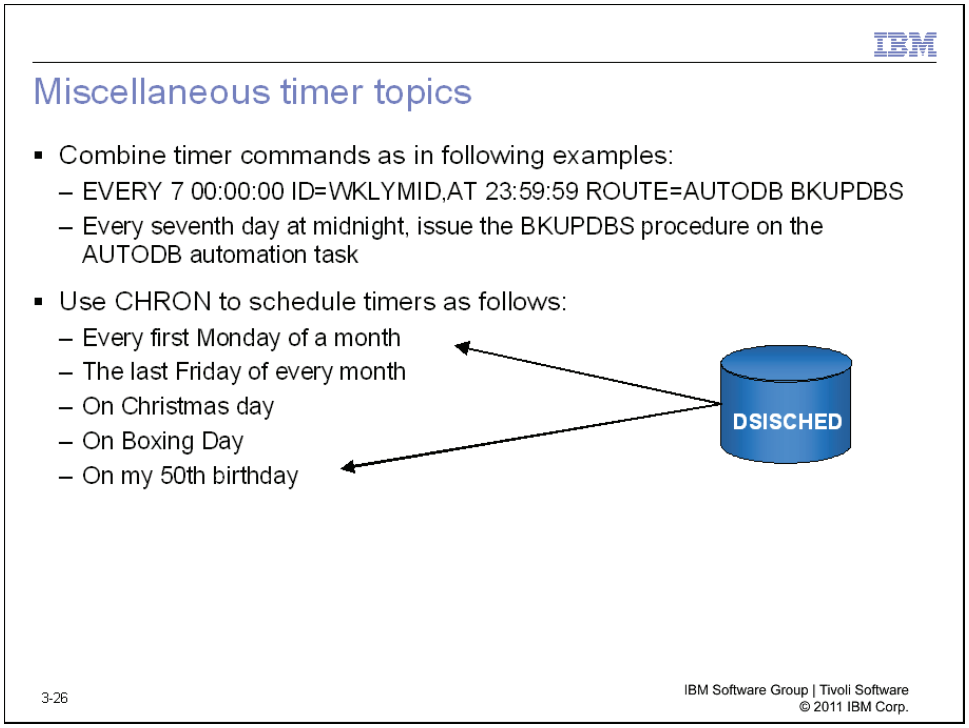
```
AT 0,ID=MIDNIGHT,SAVE,SODCLIST
```

Use a meaningful timer ID for timers to be saved.

Note: The Save/Restore database is also used for NetView common and task global variables.

The RESTORE TIMER command retrieves all timers in the Save/Restore database. Timers are restored when NetView initializes with the INIT.TIMER=Yes statement in CNMSTYLE. If a timer has expired, it is not restored, but a message is provided to indicate the expiration.

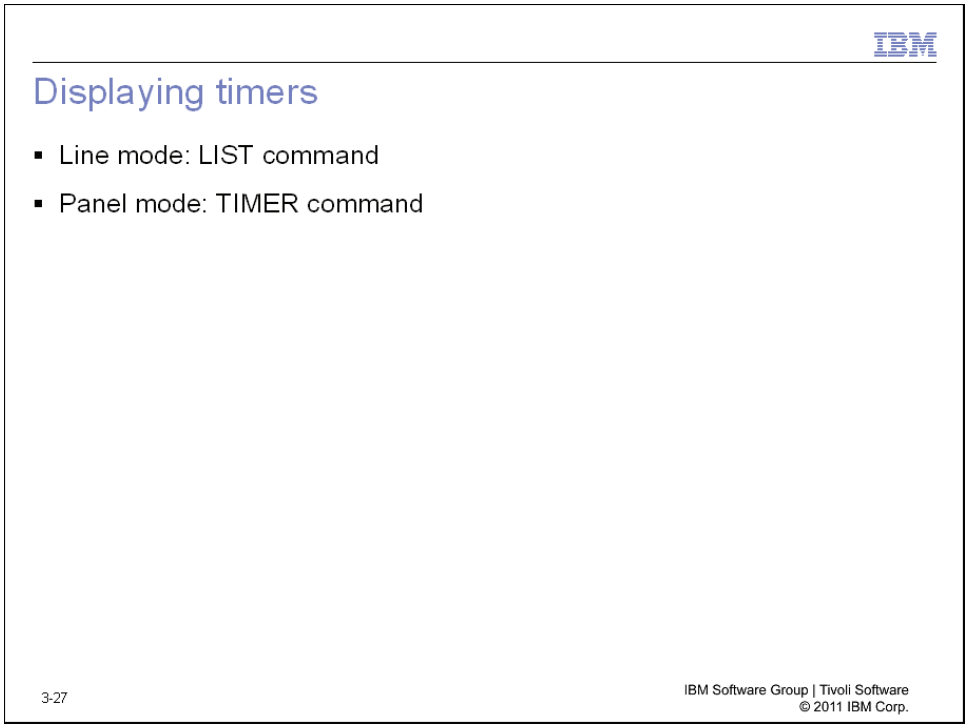
Miscellaneous timer topics



You can combine NetView timers. This slide shows an example of combining an EVERY timer with an AT timer. In this case, the BKUPDBS command runs on the AUTODB task every seventh day at midnight, an example of a timer scheduled on a weekly interval.

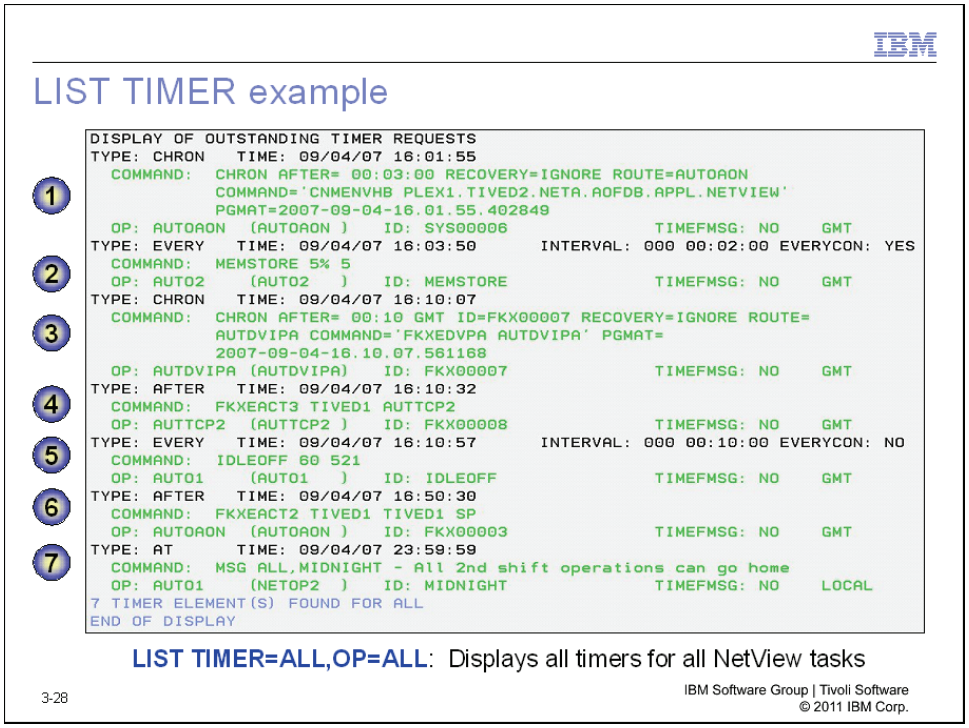
CHRON timers support more functionality than AT, EVERY, and AFTER timers by using a calendar function (DSISCHED member) defined by the user. You can schedule a CHRON timer to run on a team member's birthday.

Displaying timers



The NetView LIST TIMER command displays AT, EVER, and AFTER timers. The NetView TIMER command displays timers on a full-screen panel. You can also use TIMER to create or modify timers. In addition, TIMER supports the display of CHRON timers.

LIST TIMER example



Use the LIST TIMER command to display active timers. You can display timers for a specific task, all timers, or a specific time.

The slide, which shows seven timers, is an example of a LIST TIMER=ALL,OP=ALL command to display all timers for all NetView tasks. Field characteristics are as follows:

- The *TYPE:* field indicates the ID type of timer: two EVERY, two AFTER, two CHRON, and one AT.
- The *OP:* field indicates the task that the timers are scheduled under: AUTO1, AUTO2, AUTOAON, AUTTCP2, and AUTDVIPA.

To display the timers for AUTO1 task, issue the following command:

```
LIST TIMER=ALL,OP=AUTO1
```

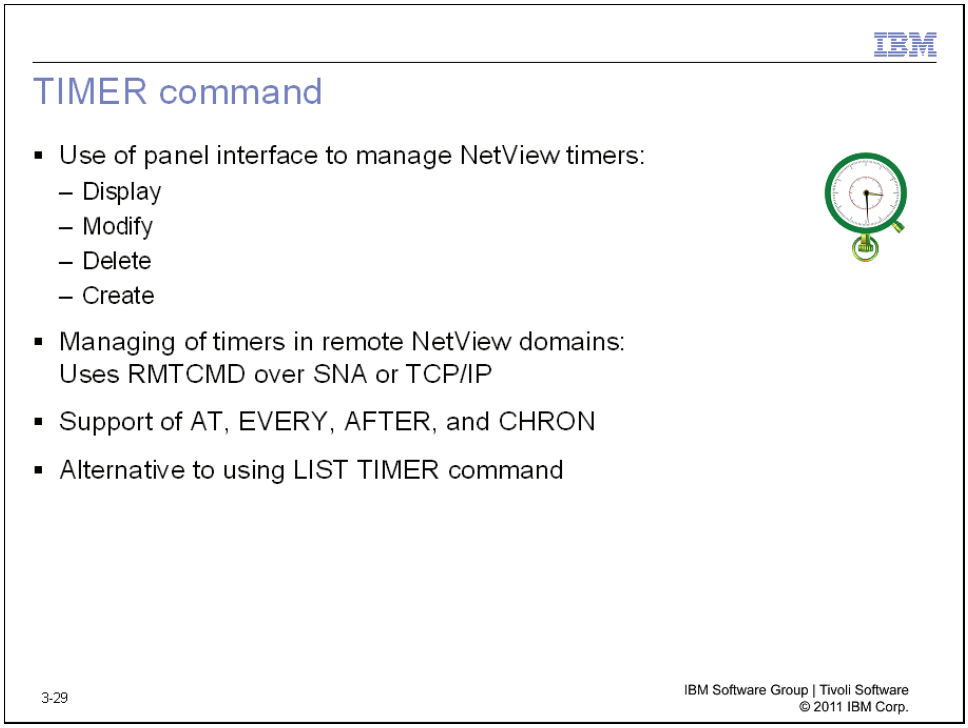
To display only the MEMSTORE timer that is scheduled under AUTO2, issue the following command:

```
LIST TIMER=MEMSTORE,OP=AUTO2
```

The MEMSTORE timer monitors the usage of several NetView files (for example, EXECs and DSIPARM members). MEMSTORE loads the most frequently used files in NetView storage.

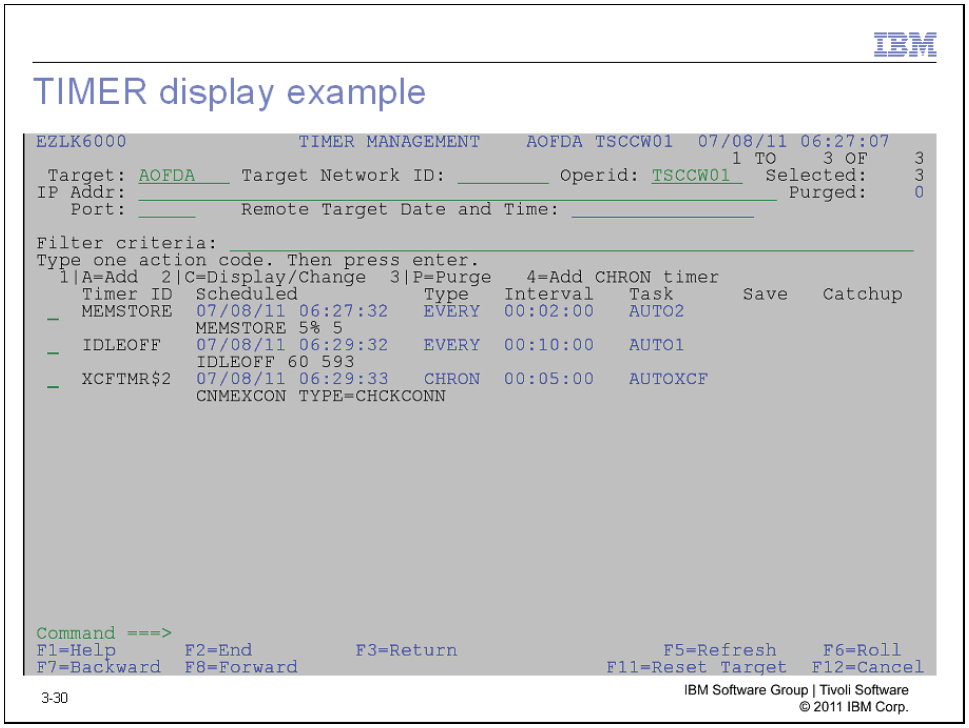
If you have many timers scheduled, you can use the TIMER command. The IDLEOFF timer monitors NetView tasks. If a task is inactive, it can be logged off.

TIMER command



The TIMER command displays AT, EVERY, AFTER, and CHRON timers on a full-screen panel and can be used for deleting, modifying, or creating timers. The TIMER command can be used for displaying timers on remote NetView systems that use RMTCMD over SNA or TCP/IP. If you run SA z/OS, the TIMER command also uses the SA z/OS XCF communication between NetView domains.

TIMER display example



The slide shows an example of the panel resulting from a TIMER command. Operators can work with timers in other NetView domains by specifying the following information:

- Domain ID in the *Target:* field for TCP/IP communication
- Network ID in the *Target Network ID:* field for SNA communication)

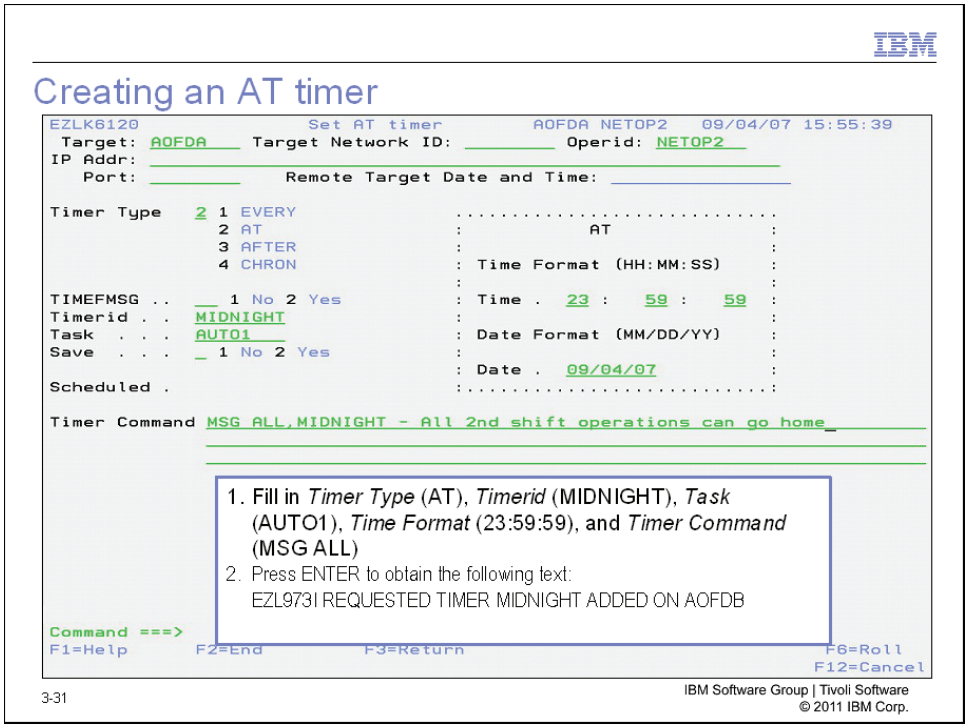
The EZLRMTTIMER statement in CNMSTYLE controls communication between NetView domains as follows:

- EZLRMTTIMER=NETV: Systems use the RMTCMD command between them over TCP/IP or SNA.
- EZLRMTTIMER=SA: Systems use SA z/OS XCF communication to SA z/OS NetView domains and RMTCMD over SNA communication to others.

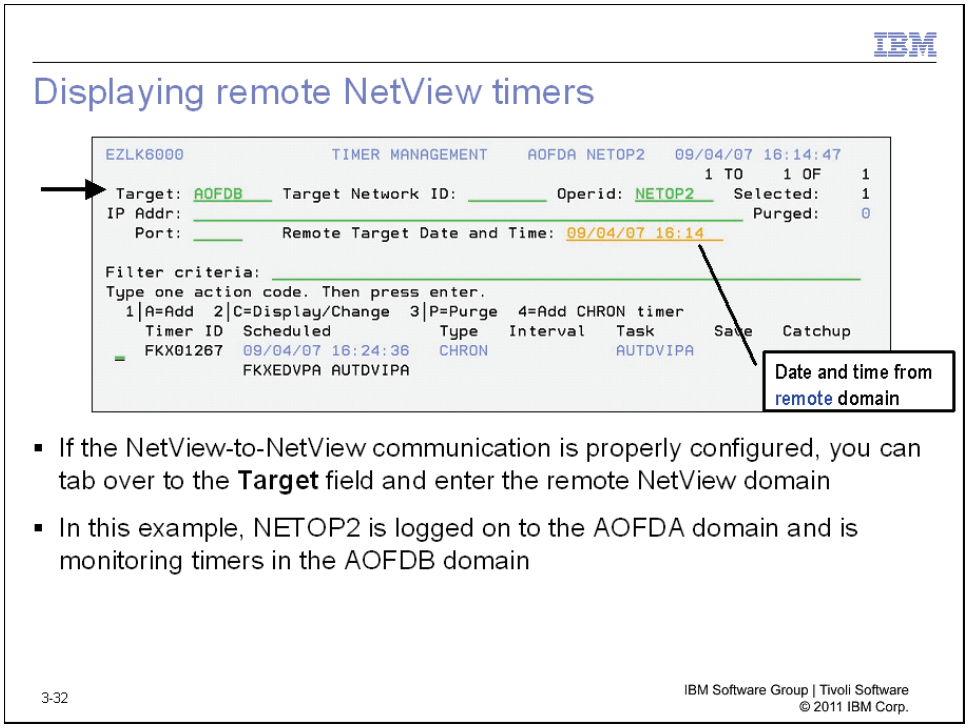
Tip: The *Remote Target Date and Time* field can be helpful if you connect to a NetView domain in a different time zone.

After purging (deleting) a timer, an additional PF key (PF9=Purged Timers) is shown on the panel. Press PF9 to display timers that have been purged. You can recover one or more of the purged timers.

Creating an AT timer

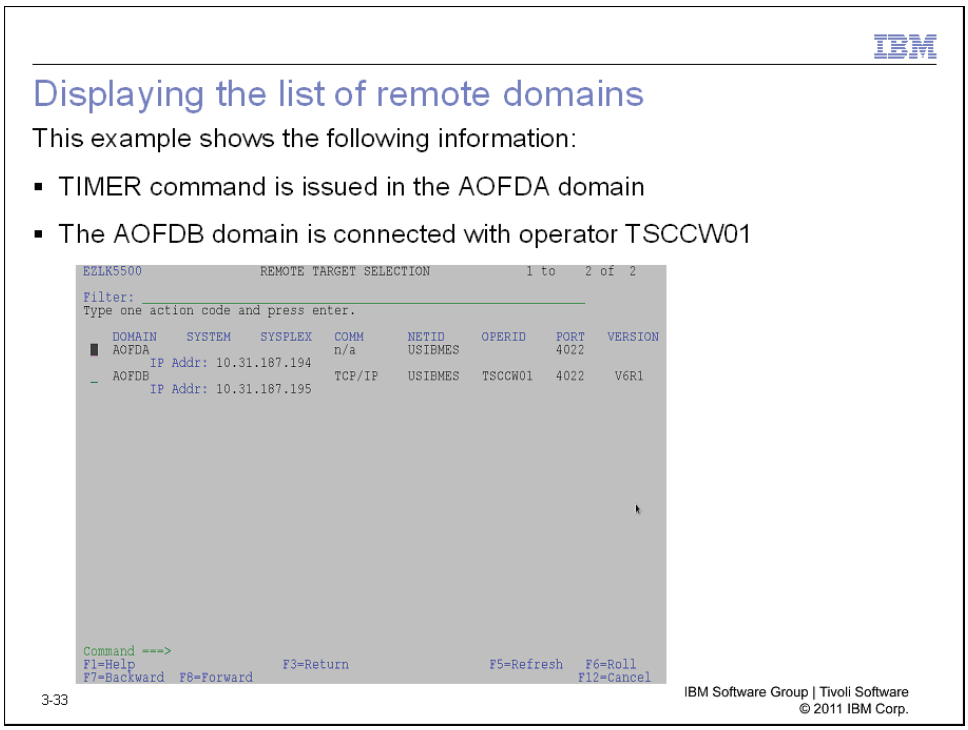


Displaying remote NetView timers



Notice the *Remote Target Date and Time* field. In this example, the local and remote NetView domains are in the same time zone.

Displaying the list of remote domains

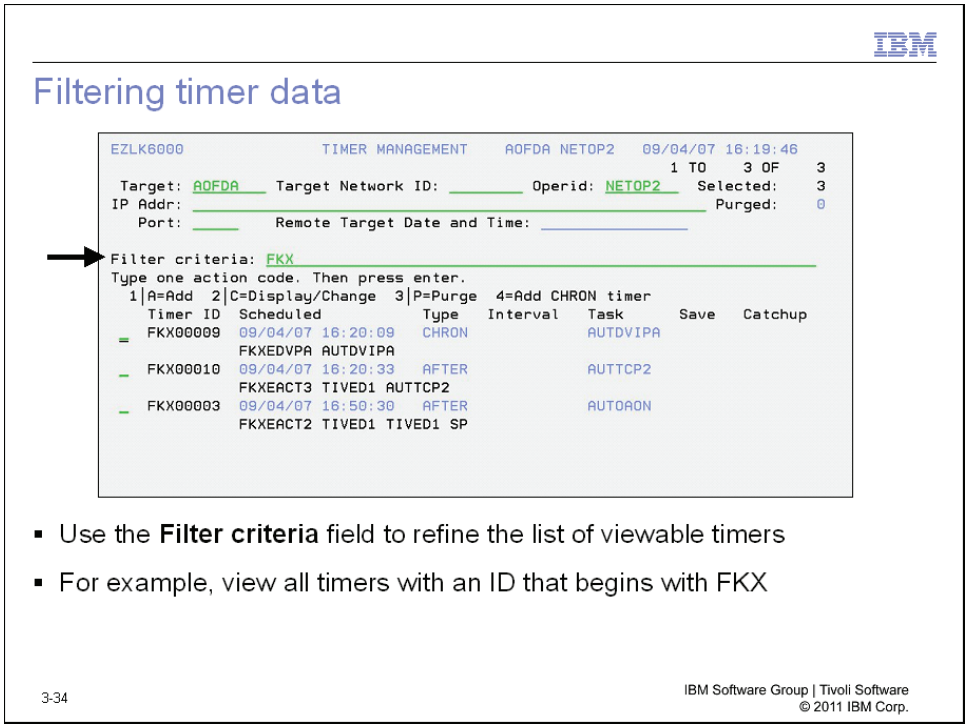


If you are unsure of the NetView domains that you can connect to, type a question mark (**?**) in the *Target* field of the TIMER panel to see a panel similar to the slide.

Note: The domain list is built from RMTSYN statements and also any active RMTCMD sessions (SNA and TCP/IP). You might have other NetView domains that are capable of receiving a RMTCMD session that are not in the list. They might not be in the list because they are not defined with RMTSYN or are not currently active.

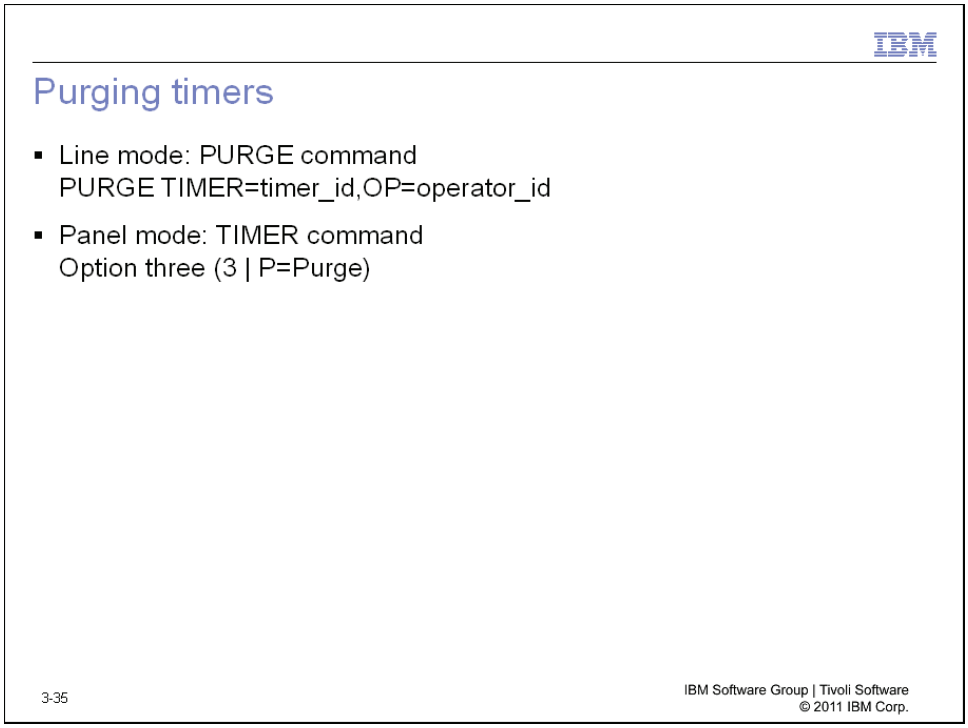
In this example, the TIMER command can connect to the AOFDB domain by using RMTCMD over TCP/IP. This panel also shows an existing connection to the AOFDB domain as TSCCW01.

Filtering timer data



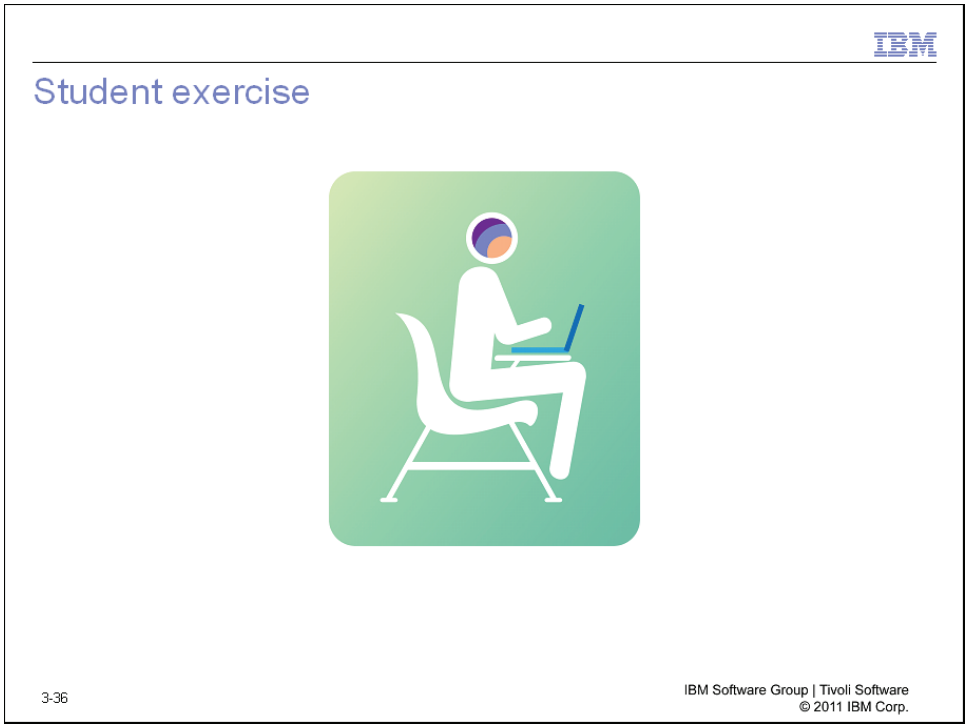
You can use the *Filter criteria* field to reduce the number of timers that are displayed on the TIMER panel. In this case, NETOP2 filters for all timers with a timer ID that begins with FKX.

Purging timers



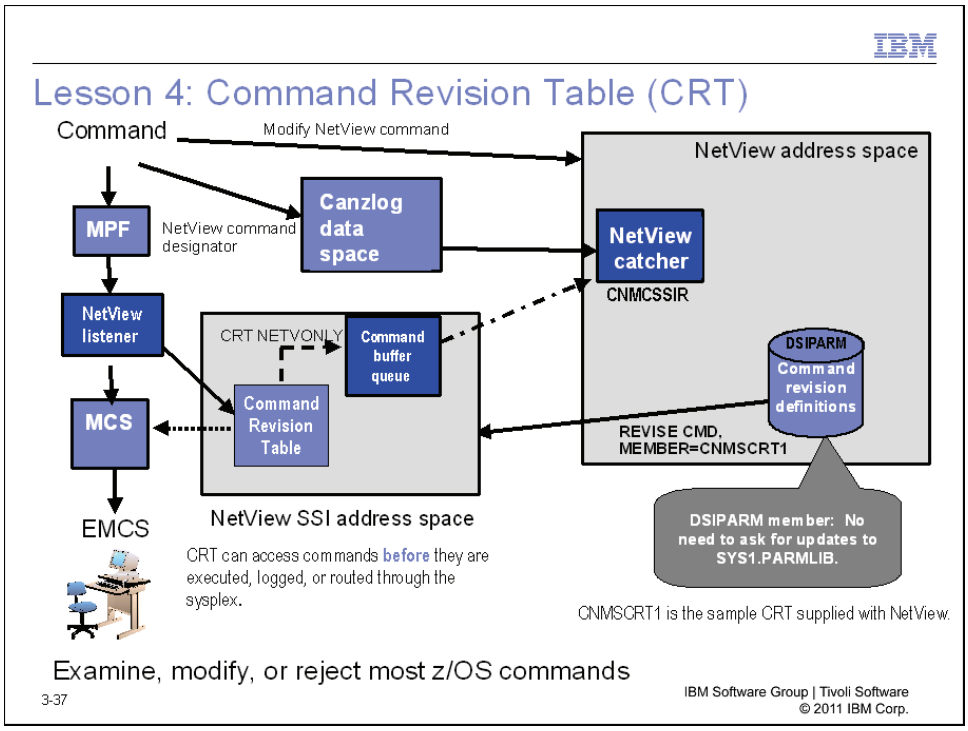
Use the PURGE TIMER command to delete timers. If you do not know the timer ID or the task, use the TIMER command to display a list of timers and select option three to delete a timer.

Student exercise



Open your *Student Exercises* book and perform Exercise 3-3.

Lesson 4: Command Revision Table (CRT)



In NetView for z/OS 5.4, the MVS Command Management was deprecated and replaced by Command Revision Table. With the Command Revision Table (CRT), you can intercept MVS commands before they process. Command sources include the MVS console and the NetView MVS command.

The CRT intercepts any text that is entered on an MVS console command line. The text might be as follows:

- An SDSF system command
- A command using the JCL COMMAND parameter
- From a program using the MGCRE macro or direct SVC 34

The text entered might or might not be a valid MVS command before it is altered or redirected by CRT processing.

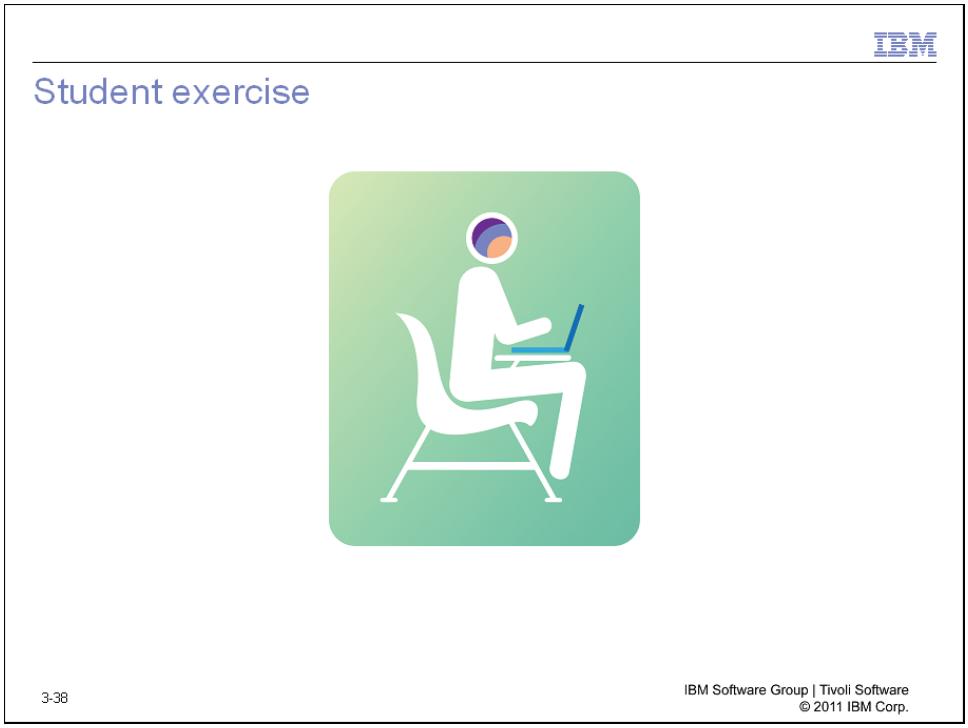
Note: If a command is reissued altered or unaltered by means of the REISSUE command as part of CRT processing, that command is exempt from CRT action. Issuing a command by using a NetView MVS command does not pass control to the CRT.

You can change the command text, write a message to the command issuer, and run the command, or suppress the command. You can also transfer the command to the NetView program for more detailed actions. The CRT can remain active even while the NetView program is not, but the SSI address space is required.

The syntax of the Command Revision Table is very similar to the syntax of the MRT. To enable the CRT, you must enable the MPF DSIRVCEX exit. This exit is in the SCNMLNKN library. This library must be in the LINKLIST. See the *IBM Tivoli NetView for z/OS Automation Guide* for more details.

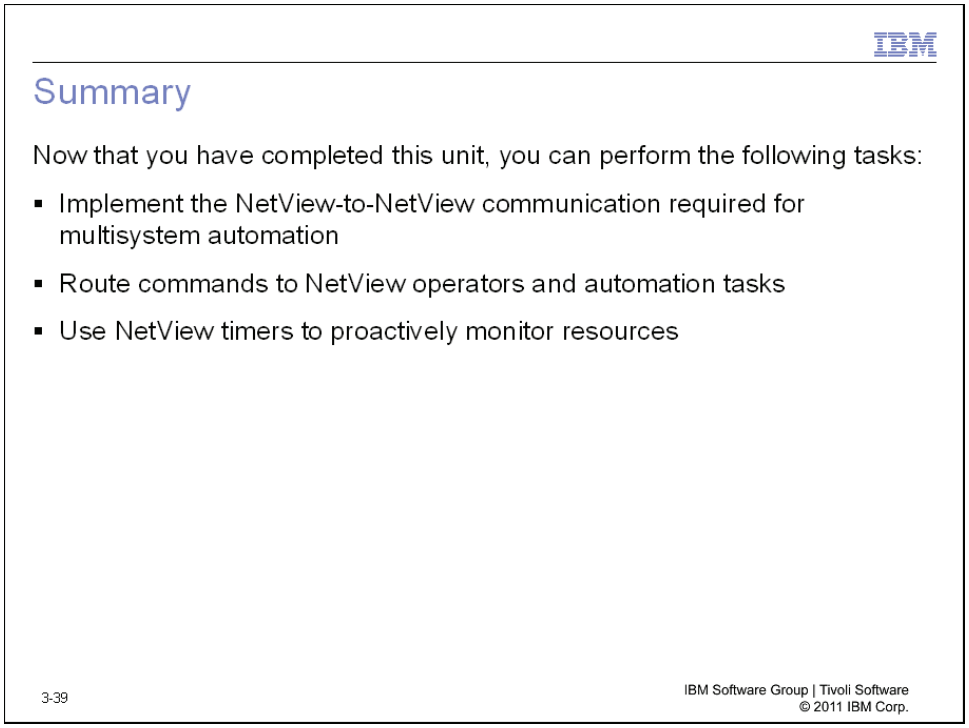
Note: Commands that the CRT's NETVONLY statement routes to NetView make use of the SSI address space. Commands with a NetView command designator are routed through the Canzlog data space. Commands that are sent to NetView through a MODIFY command are retrieved directly from z/OS.

Student exercise

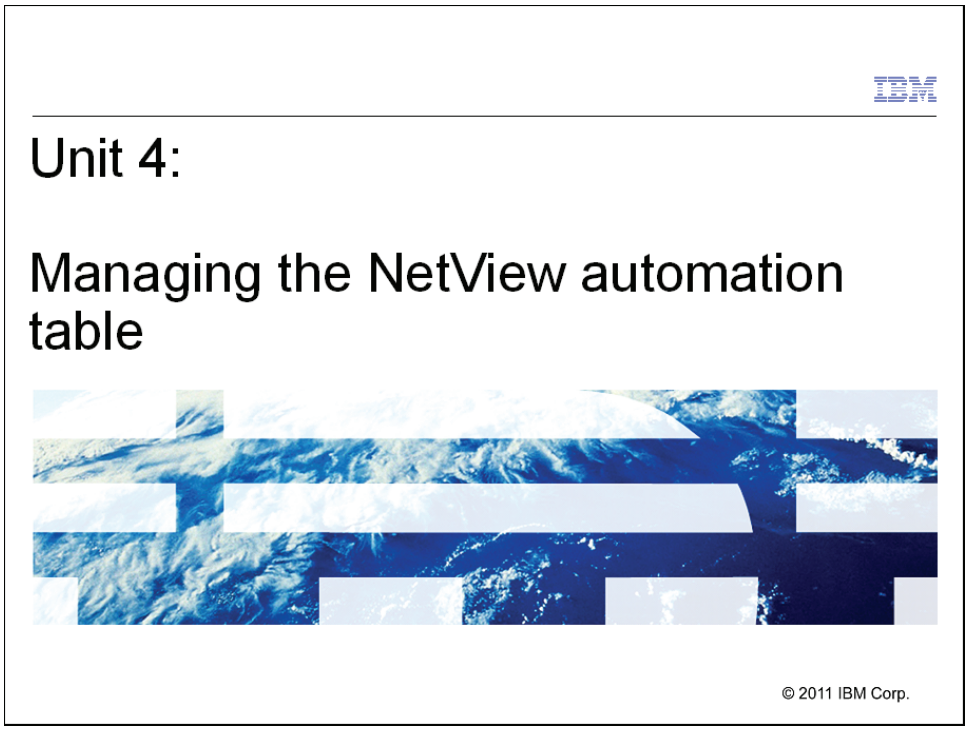


Open your *Student Exercises* book and perform Exercise 3-4.

Summary



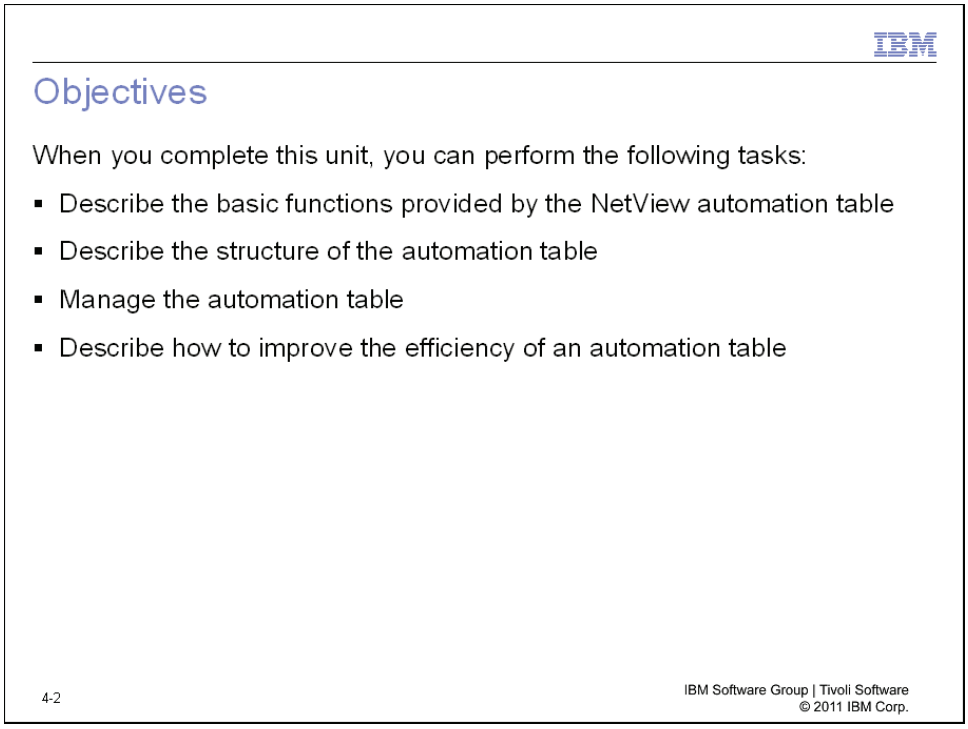
Unit 4: Managing the NetView automation table



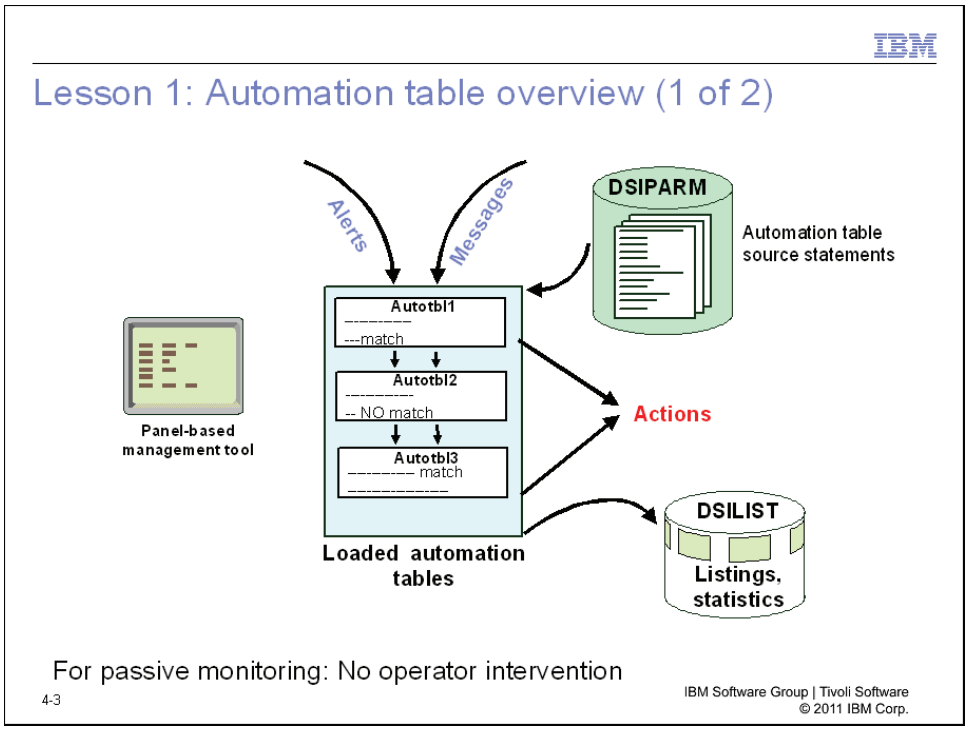
Introduction

This unit discusses the basics of the NetView automation table and how to manage one. The unit also includes information about multiple automation tables.

Objectives



Lesson 1: Automation table overview



The automation table identifies events (messages and MSUs) to trap, actions to take, and where to schedule those actions. This unit focuses on the management of the automation table. Discussion of the syntax of the statements is in Unit 5: Coding an automation table.

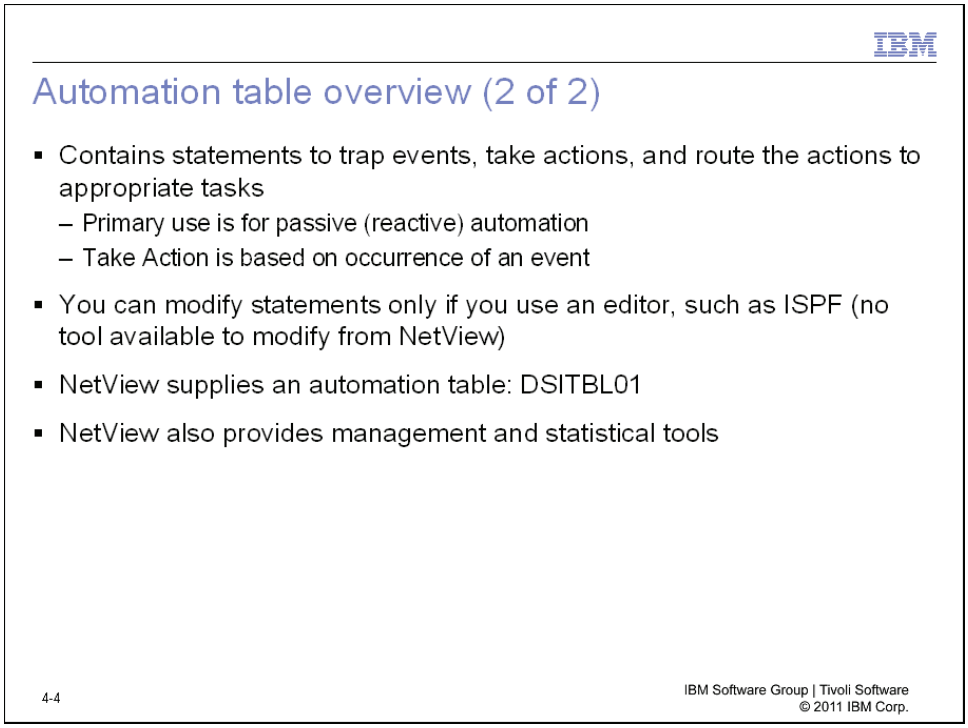
The automation table is a set of IF-THEN statements that you can use for trapping messages and alerts and taking actions. The table might consist of one member or a combination of several members.The table is in your DSIPARM data set.

The automation table is the key to *passive automation*. As messages and SNA Management Services Unit (MSU) events enter NetView, they route to the active automation tables for matching. If a match for a message or MSU is within a table, an action might be taken as a result. Automation tables can be tested before use, and the results written to a data set (DSIARPT).

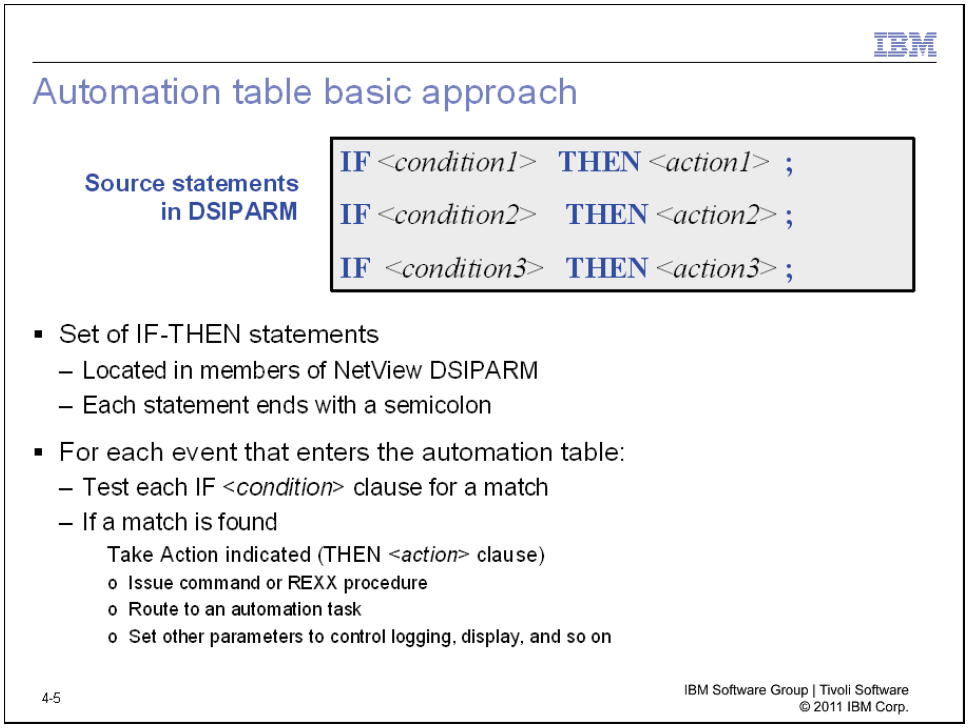
Statistics pertaining to the use of the table are available by using the AUTOCNT command. Output from this command can be written to a member of the DSILIST data set for further analysis.

A panel-driven command (AUTOMAN) can be used for managing the automation tables that are in use.

Automation table overview (2 of 2)



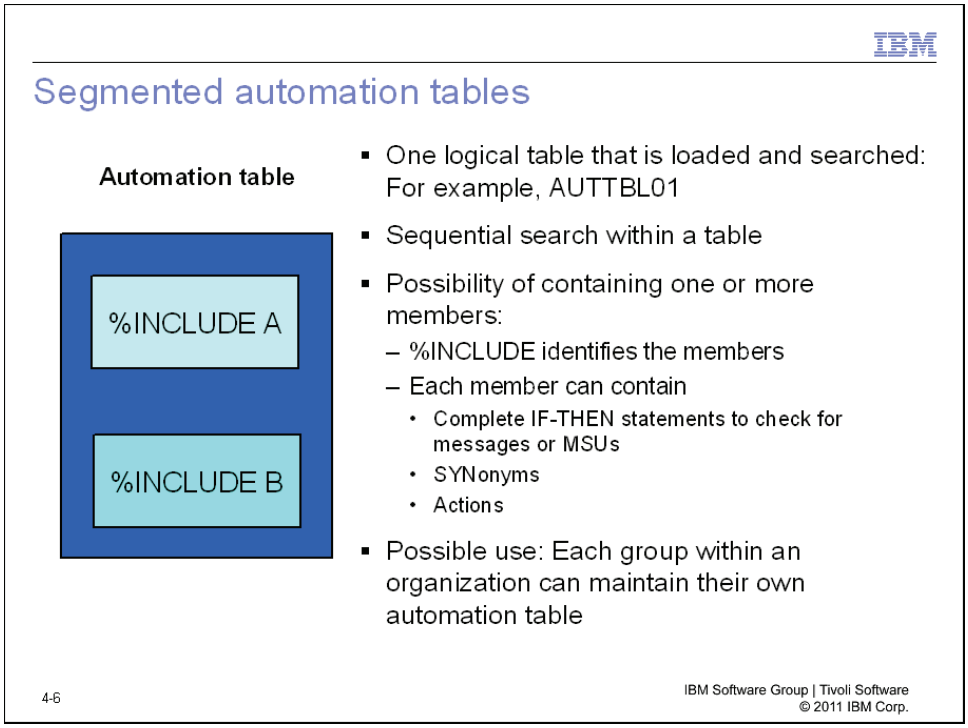
Automation table basic approach



The basic approach is testing each message that arrives in NetView against the IF clause of each statement of the table in sequence. If a match is made, the actions that are indicated in the THEN clause occur. Automation processing is complete for that message. Any IF-THEN statements after the match are ignored for that message. The basic approach has exceptions.

The simplest automation table approach is having a single automation table that has only IF-THEN statements. The single table is a single member of the DSIPARM data set.

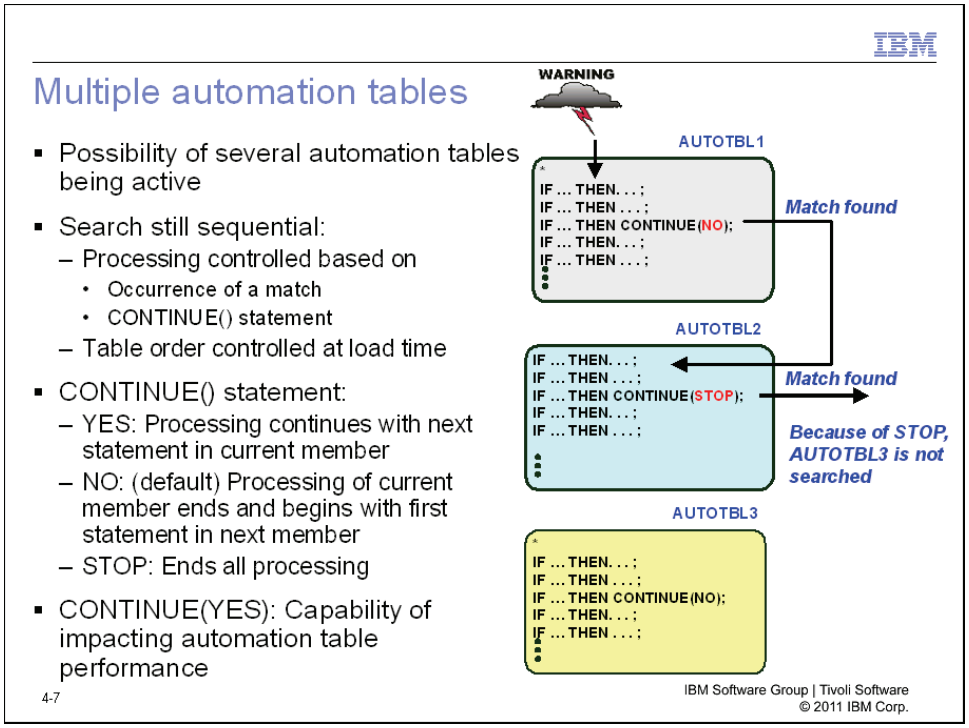
Segmented automation tables



An automation table can, like many NetView definitions, be built up from many separate members. It can be coded by using %INCLUDE statements for other DSIPARM members, each containing table statements. Thus, you can create one logical table from many physical elements. A member that is included can contain %INCLUDE statements for other members.

Such a merged table is checked, loaded, and searched as one logical unit. Testing for a message or MSU starts with the first statement in the (logical) table and progresses by testing each statement in sequence until a match occurs. This basic approach has exceptions.

Multiple automation tables



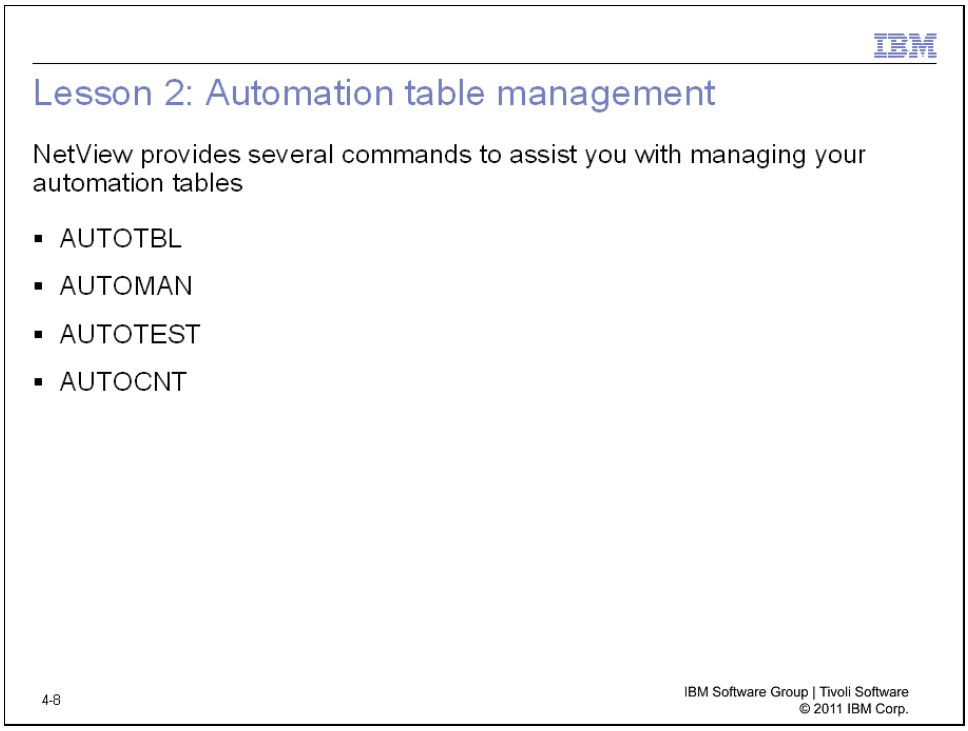
Multiple logical automation tables can be active at any time. When several tables are active, they are managed in sequence. When a message or MSU arrives, the first table processes it. When the processing in that table completes (either a match, or reaching the table end), processing begins at the start of the next table. This process repeats until all tables are processed. The order is the same for all messages and MSUs.

Exceptions to this sequential order are as follows:

- CONTINUE(STOP) action: Indicates that all automation table processing ends, even if later tables have not been processed.
- CONTINUE(YES) action: Indicates that automation table processing continues with the next statement in the current member. If not used carefully, overuse of CONTINUE(YES) can impact the performance of your automation table.

Multiple tables are useful for reducing the complexity of managing a single table. With a single table, it could be difficult to ensure that a statement processes for the intended message. By using multiple tables, you can initially develop and test smaller units, then implement them with confidence.

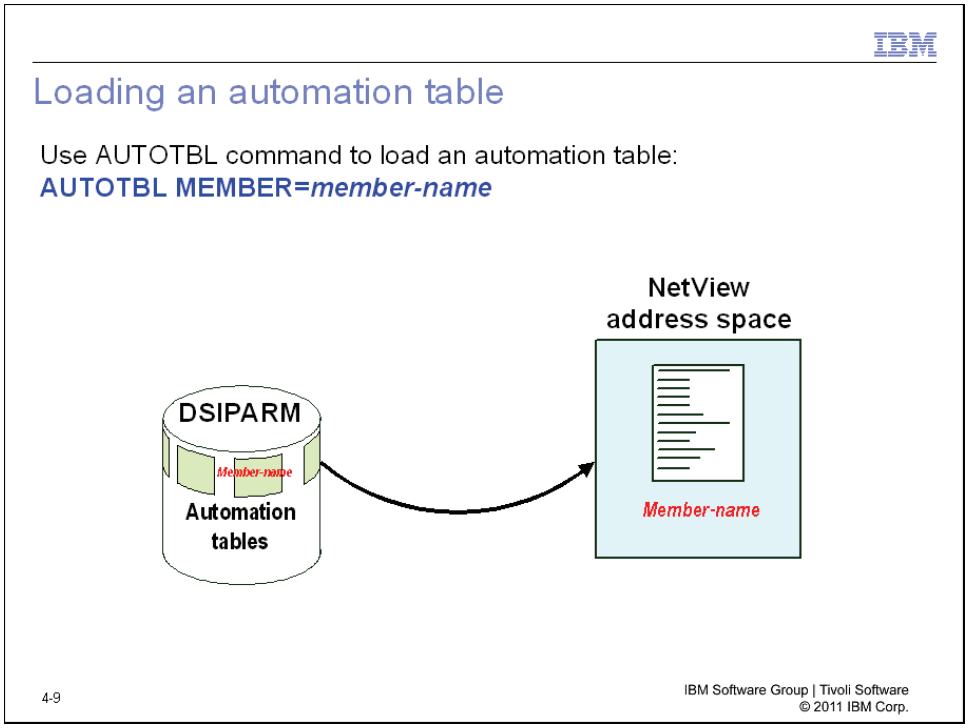
Lesson 2: Automation table management



Use the AUTOTBL and AUTOMAN commands to load and manage the NetView automation table. Example uses are as follows:

- Use the AUTOTEST command for testing changes to an automation table against messages stored over a typical day.
- Use the AUTOCNT command for displaying statistical information for tuning an automation table.

Loading an automation table



Use the AUTOTBL command for activating, testing, and checking statuses of automation tables. The basic command for activating a single automation table is as follows:

```
AUTOTBL MEMBER=member-name
```

Use the AUTOTBL command for the additional following purposes:

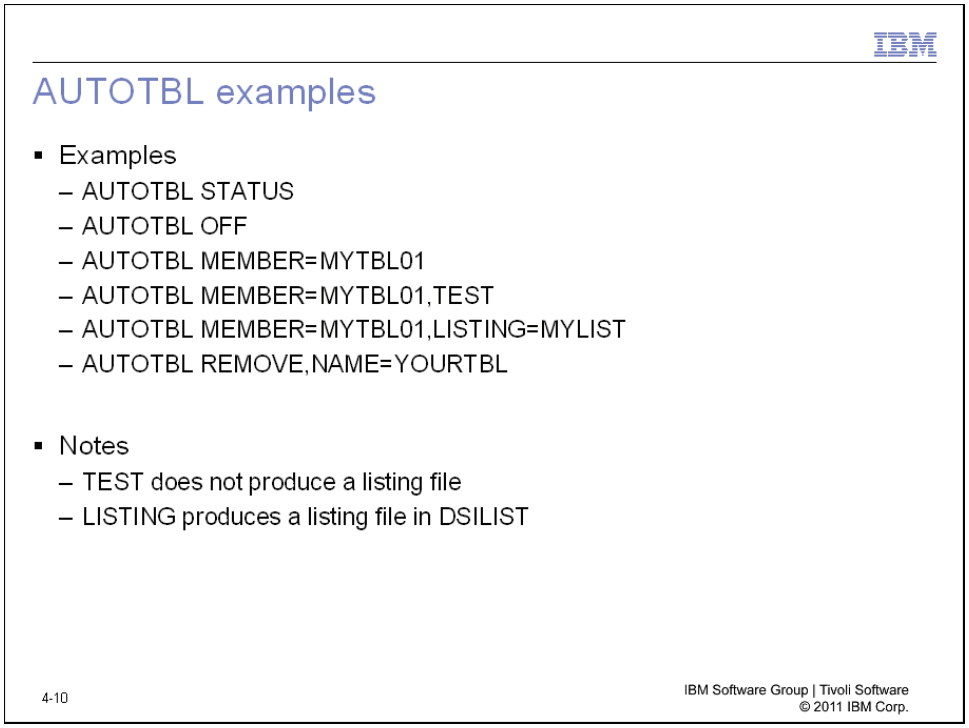
- Inserting, removing, and replacing individual automation tables
- Loading additional tables into the list of active tables
- Displaying the statuses of tables
- Disabling and enabling automation table statements or groups of statements
- Listing details of automation tables

NetView can load automation tables during initialization based on CNMSTYLE statements. Discussion about loading automation tables from CNMSTYLE occurs later in this lesson.

Table activation results in resolution of %INCLUDE, >%IF, and SYN statements, then the creation of in-storage table definitions in a form that can yield higher performance.

Note: Automation table members are in DSIPARM and can be loaded in storage with the memStore statement. If you change an automation table and do not see the effects of the change, issue the LIST MEMSTOUT command to see if your table is loaded in storage.

AUTOTBL examples



You can test an automation table before using it. You can find common syntax errors before attempting to load the table into production. If the table has errors, the load fails and all other updates (including those from co-workers) do not load. A table that has invalid statements or BEGIN-END sections also do not load.

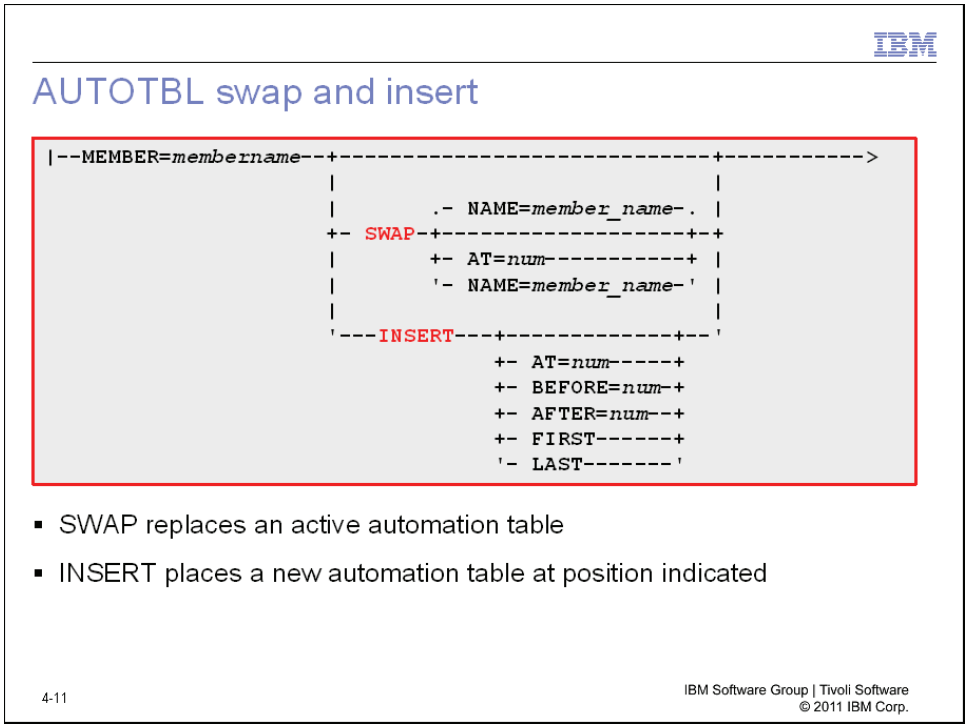
While a table reloads, automation is always available. The new table is building while the old one is still being used. Afterward, the new one is switched into use after it finishes building.

The AUTOTBL listing helps in correcting errors. It provides an expanded copy of the automation table with all synonyms resolved. The listing is written to a member of the DSILIST data set when the LISTING keyword is specified.

Use the name parameter to specify an entire table within a list of active automation tables. You can enable or disable individual statements or a group or block of statements.

You can use a panel interface to issue the AUTOMAN command to load and manage your automation tables.

AUTOTBL swap and insert

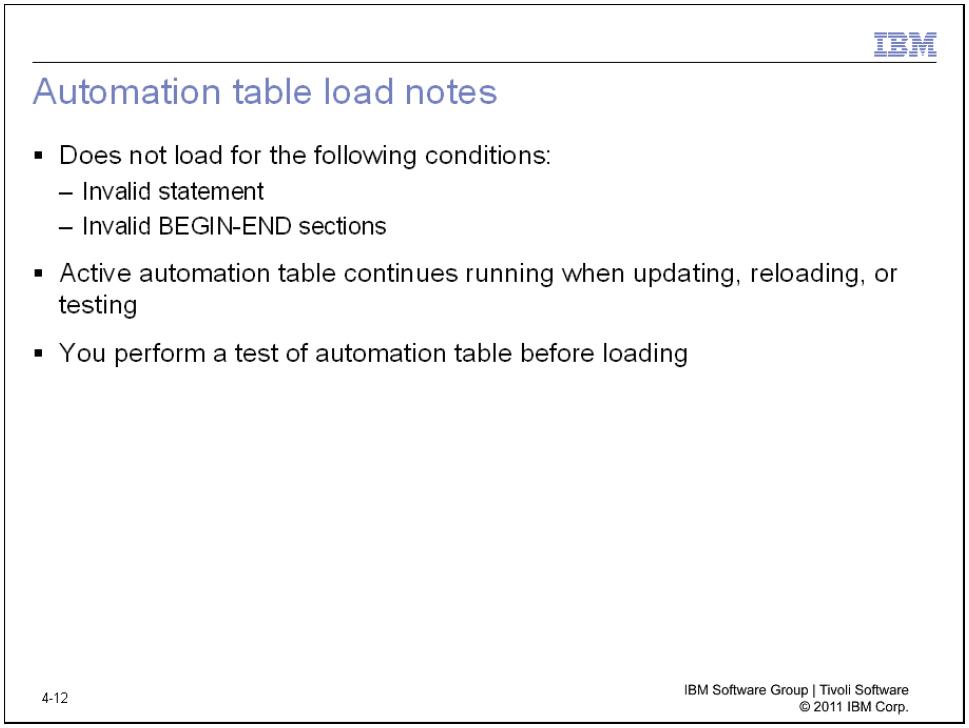


If you run more than one automation table, two important operands of the AUTOTBL command are as follows:

- SWAP: Used for replacing an active table with a new member. If the same name is used, this action is effectively a reload in place. If the current active table has any disabled statements, similar statements do not disable when the new table loads.
- INSERT: Used for placing a new table within the active table as indicated, the number specified indicating its position.

  FIRST and LAST do not require use of a number and can be removed only by using the REMOVE option.

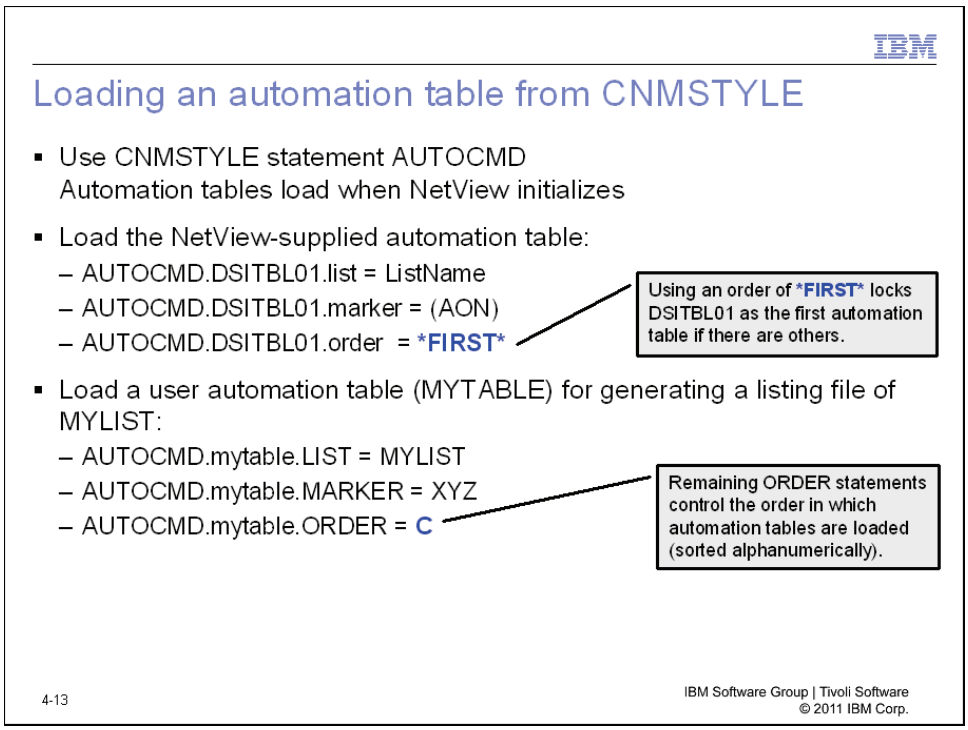
Automation table load notes



If an automation table contains syntax errors, it does not load. If an active automation table already exists, it remains the active table. The current automation table is replaced only when the new table does not contain any errors.

By default, the automation table does not load if you call commands or EXECs that do not exist. For example, suppose you create automation table statements to issue an EXEC called RECOVER but have not created the EXEC when the table loads. This can result in an error unless you set a constant in the NetView DSICTMOD constants module. As an alternative, you can create the RECOVER EXEC with only a comment line.

Loading an automation table from CNMSTYLE



The automation table can be loaded as follows:

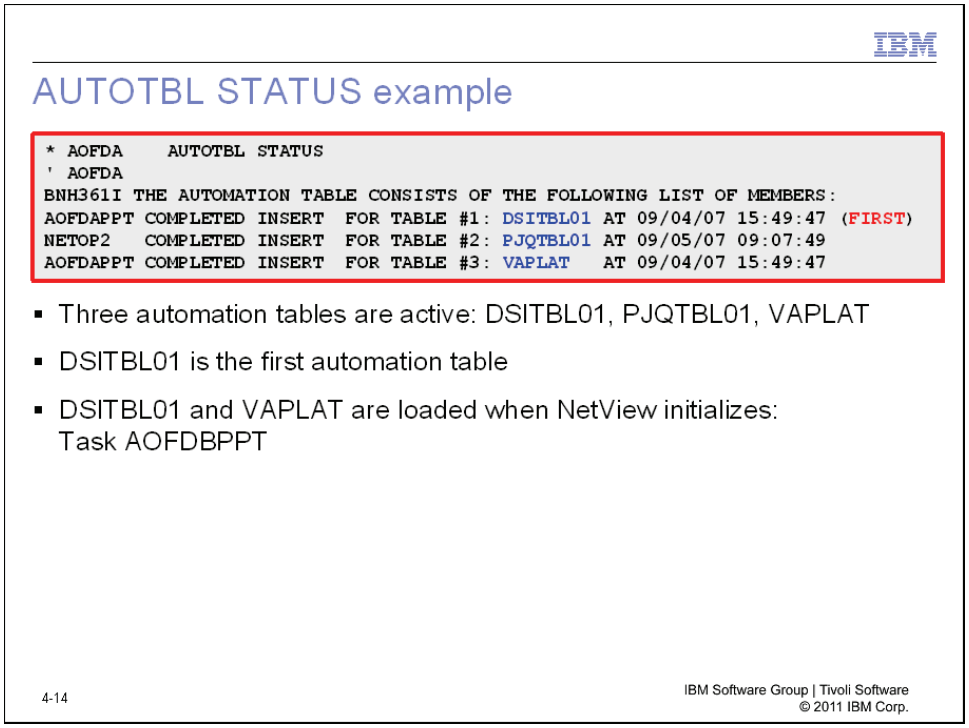
- Manually with the AUTOTBL or AUTOMAN commands
- When NetView initializes

Use the AUTOCMD definition statements in CNMSTYLE to load automation tables when NetView initializes. The syntax is as follows:

```
AUTOCMD.member-name.LIST = listfile
AUTOCMD.member-name.marker = marker
AUTOCMD.member-name.order = A | B | C ...
```

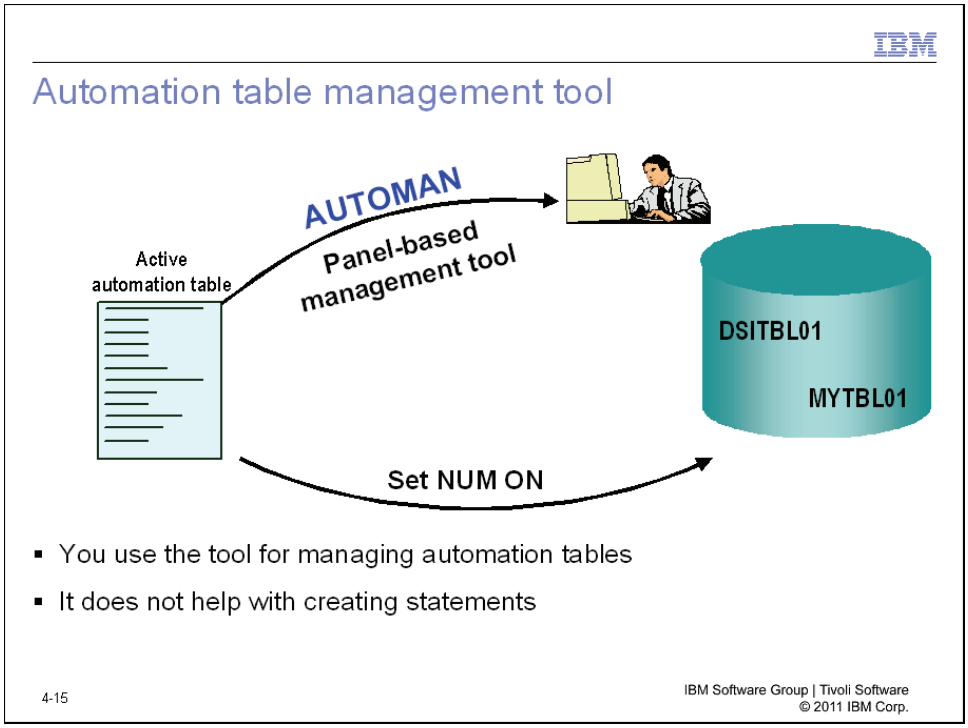
- *Member-name* is the automation table member name in DSIPARM. An example is DSITBL01 or MYTABLE.
- *Marker* associates the table with descriptive text that can be displayed when the table status is queried. For example, NIGHT can identify the table for third-shift operations.
- *Order* identifies the position that the table is to load in. *FIRST* locks the member as the first automation table. The tables load based on an alphanumeric sort of the order statement.

AUTOTBL STATUS example



AUTOTBL STATUS displays the status of the automation tables. In this example slide, three automation tables are active. DSITBL01 is locked as the first table. PJQTBL01 is in position two and is loaded by NETOP2 after NetView initialized. All tables that loaded during initialization are loaded under the PPT task.

Automation table management tool

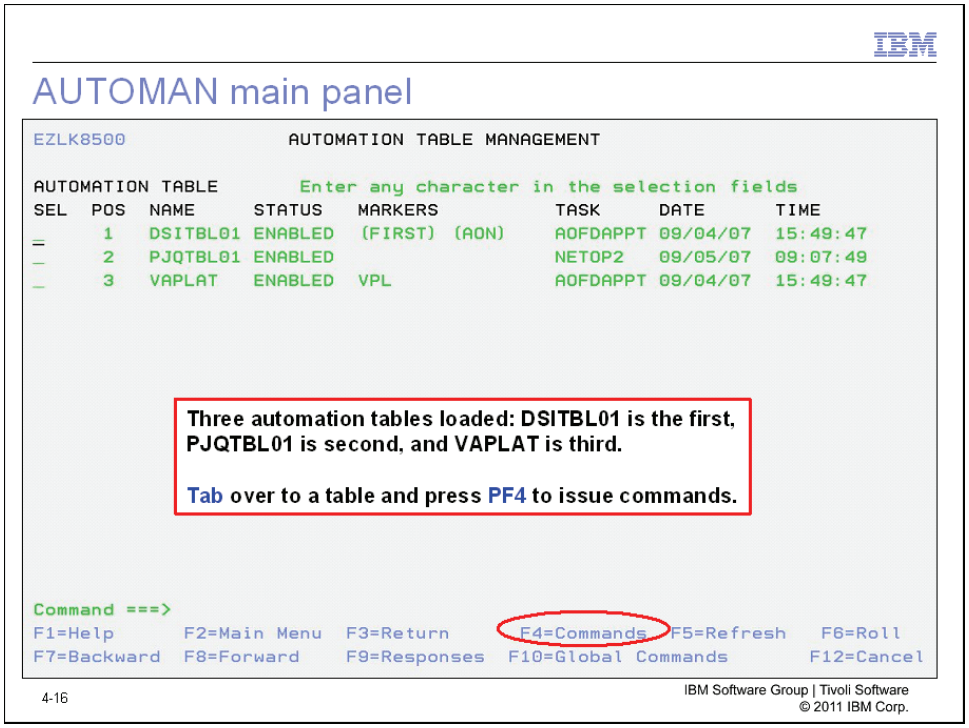


NetView supplies an automation table management tool called AUTOMAN, which is a command that displays a color-coded panel to assist with the following tasks:

- Table loading, reloading, and removing
- Table and member insertion
- Enabling and disabling parts of a table
- Browse table or listing (full or partial)
- Viewing of table statistics

AUTOMAN is an alternative to the AUTOTBL command, but some actions require that the table statements have sequence numbering. (Use TSO EDIT to set NUM ON.)

AUTOMAN main panel



The AUTOMAN command displays the automation table management tool, which permits all AUTOTBL commands to be issued. At invocation, it produces a panel similar to the slide example.

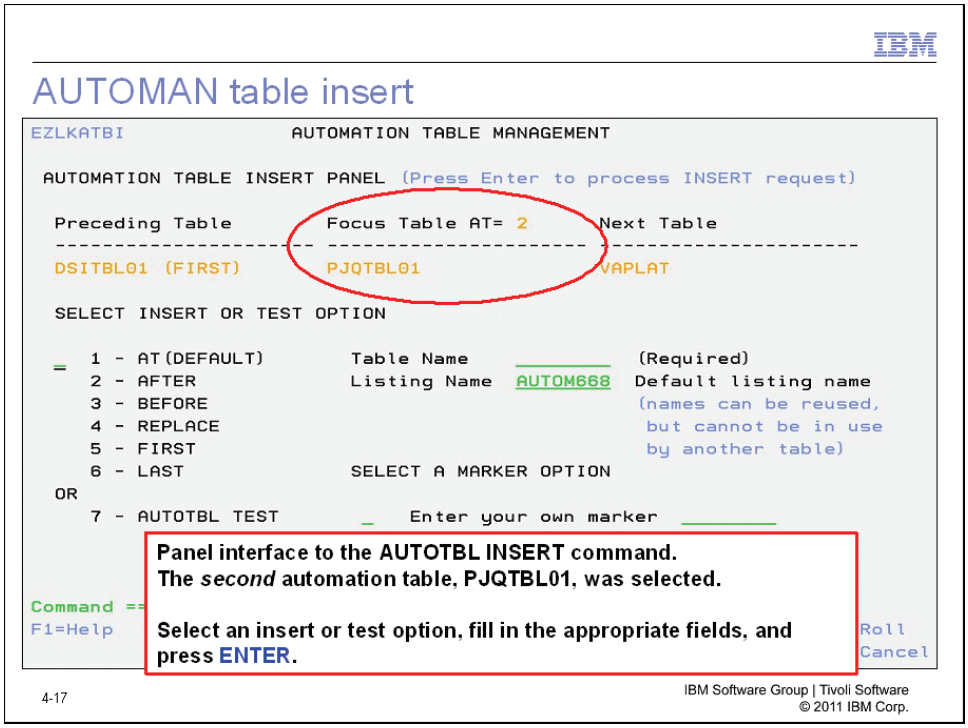
The initial display shows all loaded tables in their current order. By default, the NetView automation table, DSITBL01, is *marked* as the first table. If you have the AON component active, it indicates that AON is active and using DSITBL01.

To take an action against a member, perform the following steps:

1. Tab over to the SEL column for that member:
2. Type any non-blank character, and press ENTER.
3. Press PF4 (Commands).

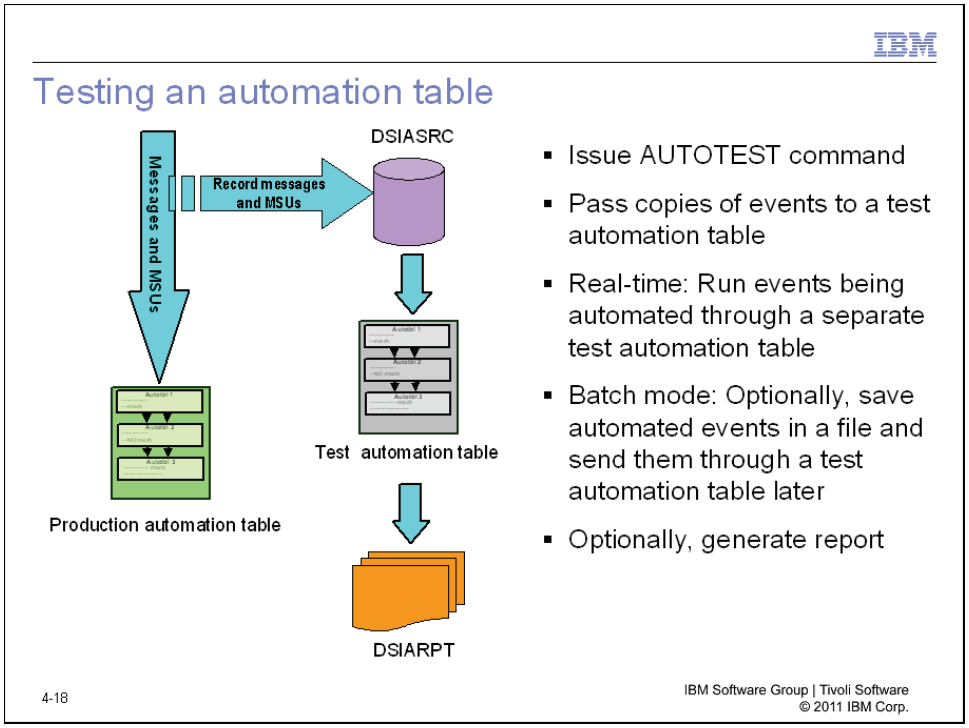
Press PFF9 to display a panel that contains the results of any tests or error messages.

AUTOMAN table insert



This example shows how to insert an automation table member by using AUTOMAN. The current member (focus table) is PJQTBL01. Select an insert or test option. For example, option **2** (AFTER) inserts a new table after PJQTBL01. Specify the table name in the *Table Name* field, and press ENTER.

Testing an automation table

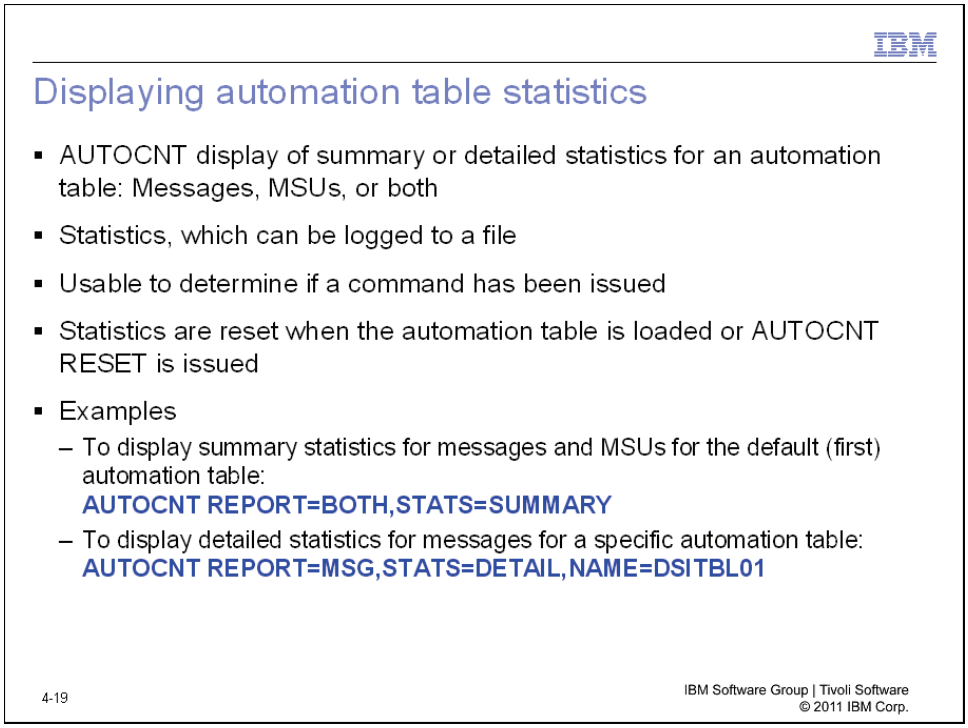


You can use the AUTOTEST command to perform the following tasks:

- Create a member that contains the messages and MSUs that entered the production automation table.

  You can use these messages later as input to the test automation table. The collection does not require the test automation table to be active. It does not affect regular automation table processing.
- Discover and correct any logic, typographical, or ordering problems before putting an automation table into the production environment.
- Compile a test automation table and load it into storage in preparation for activating an automation table test.
- Activate or deactivate the testing of a test automation table.
- Specify the source of messages and MSUs to be used as input to the test automation table being tested.
- Display the status of the test automation table testing function.

Displaying automation table statistics

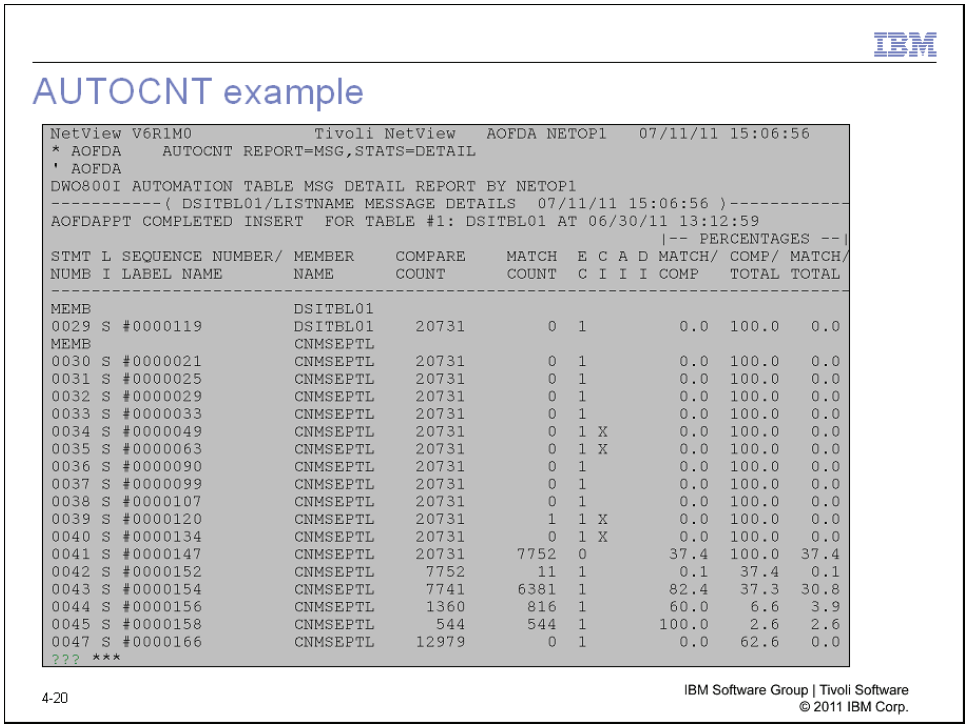


An automation table must be designed correctly and then tuned. To assist with tuning, NetView provides the AUTOCNT command. The AUTOCNT command creates performance statistics pertaining to the active automation tables. These statistics can be written to a member of the DSILIST data set for further analysis.

The statistics give information about the flow of messages through the automation table. Data is included for the average number of matches that are made for each message. Additional information about each statement includes the number of compares are made for it, and how matches there are. With the statistics, a user can determine whether the table is well-constructed and efficient or not.

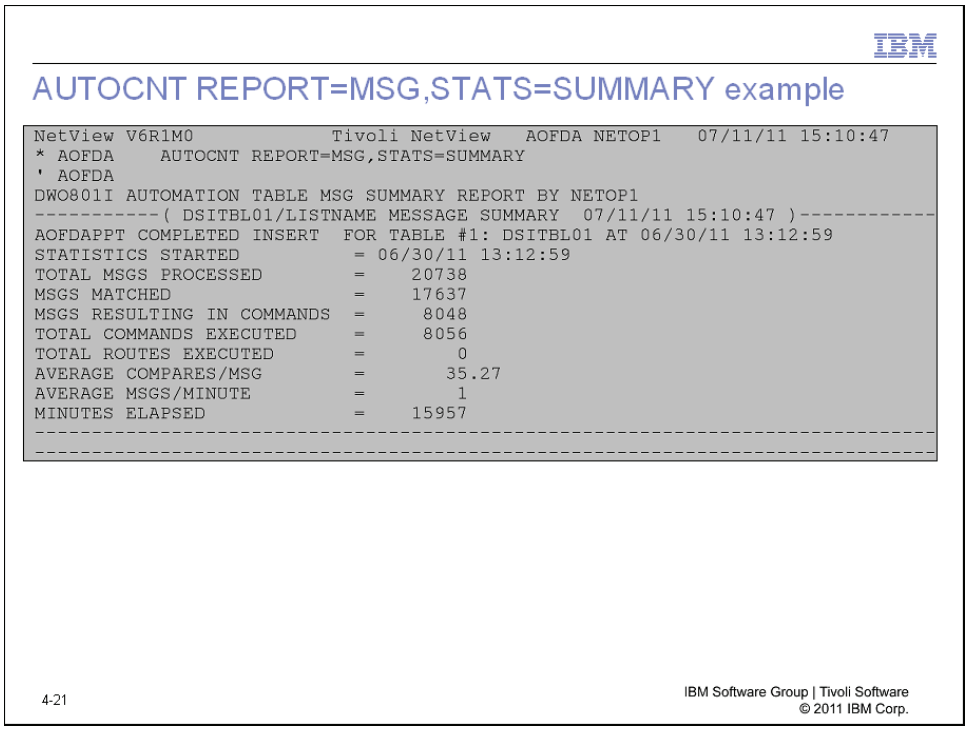
Statistics are removed when a table reloads or if the AUTOCNT RESET command is issued.

AUTOCNT example



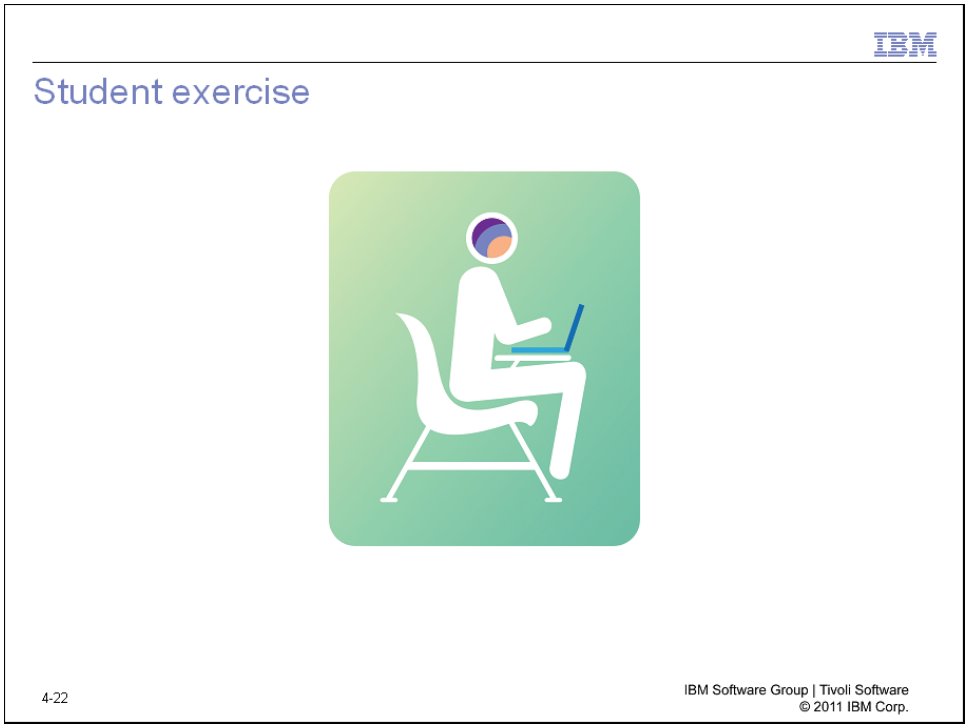
You can also trace the processing of a message or MSU through the automation table using the TRACE action. The TRACE action also sets a trace tag as an indicator that the event is to be traced while the automation table processes the event. Detailed trace information is displayed near BNH370I message for each part of each automation table statement that is processed. See the *IBM Tivoli NetView for z/OS Automation Guide* for more details on debugging tools.

AUTOCNT REPORT=MSG,STATS=SUMMARY example



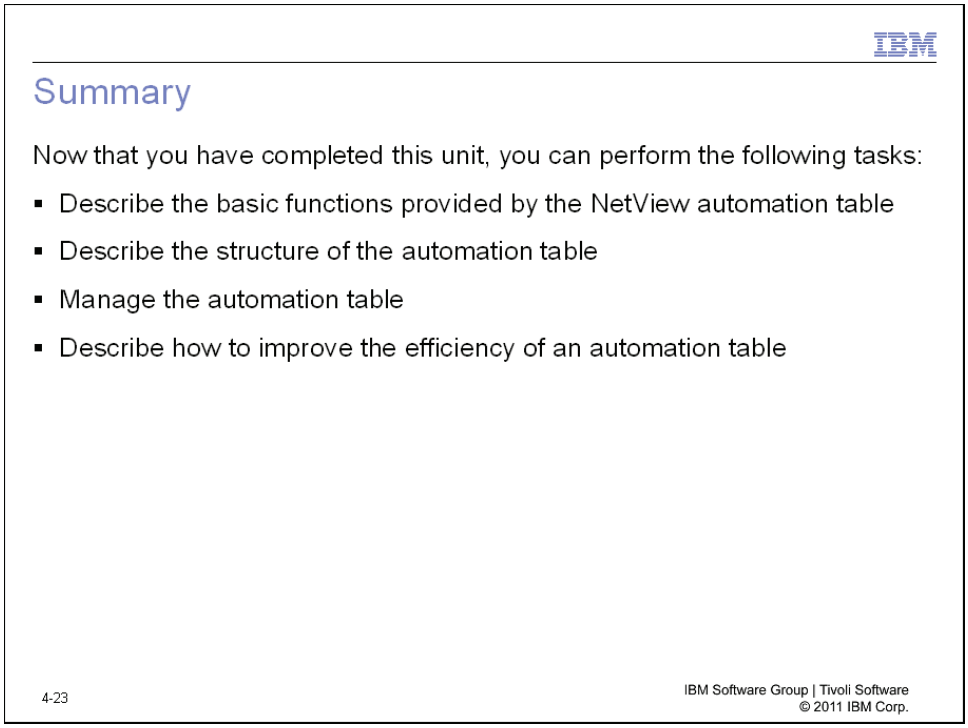
If statistics for AVERAGE COMPARES/MSG seem very high, you should consider restructuring your automation table.

Student exercise

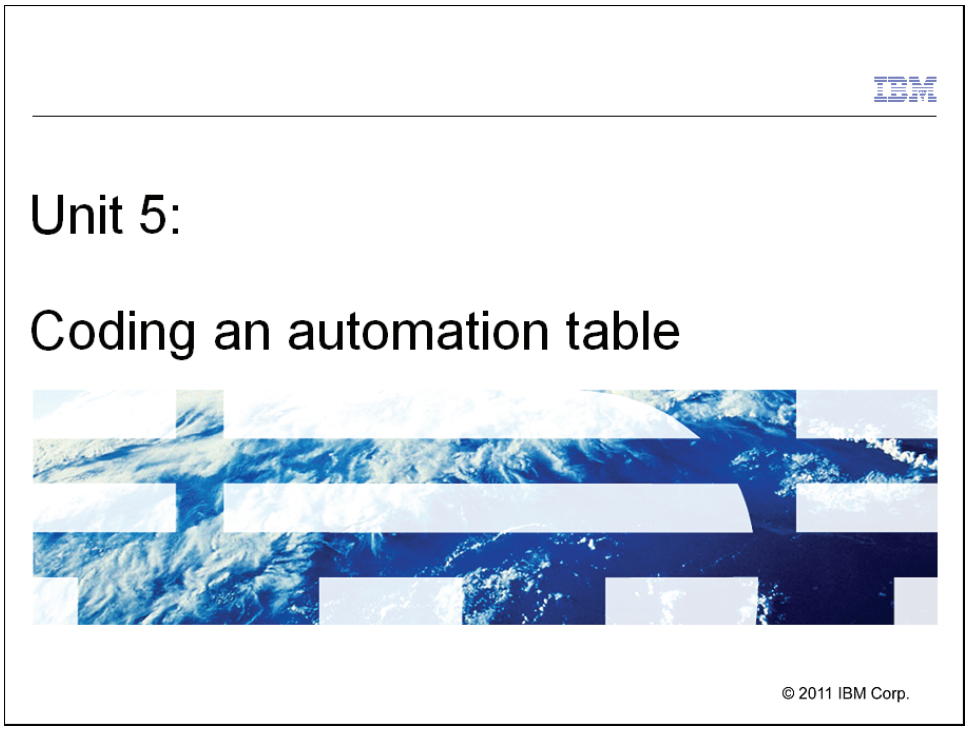


Open your *Student Exercises* book and perform Exercise 4-1.

Summary



Unit 5: Coding an automation table

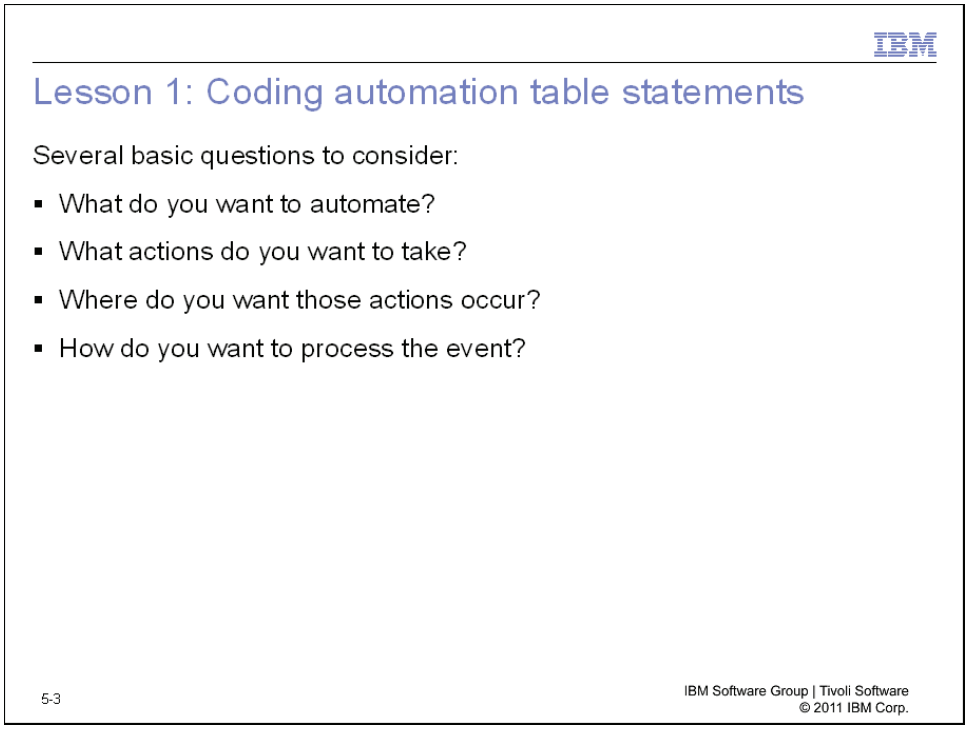


Introduction

This unit discusses the syntax of the NetView automation table statements. Examples illustrate various automation table coding techniques.

Objectives

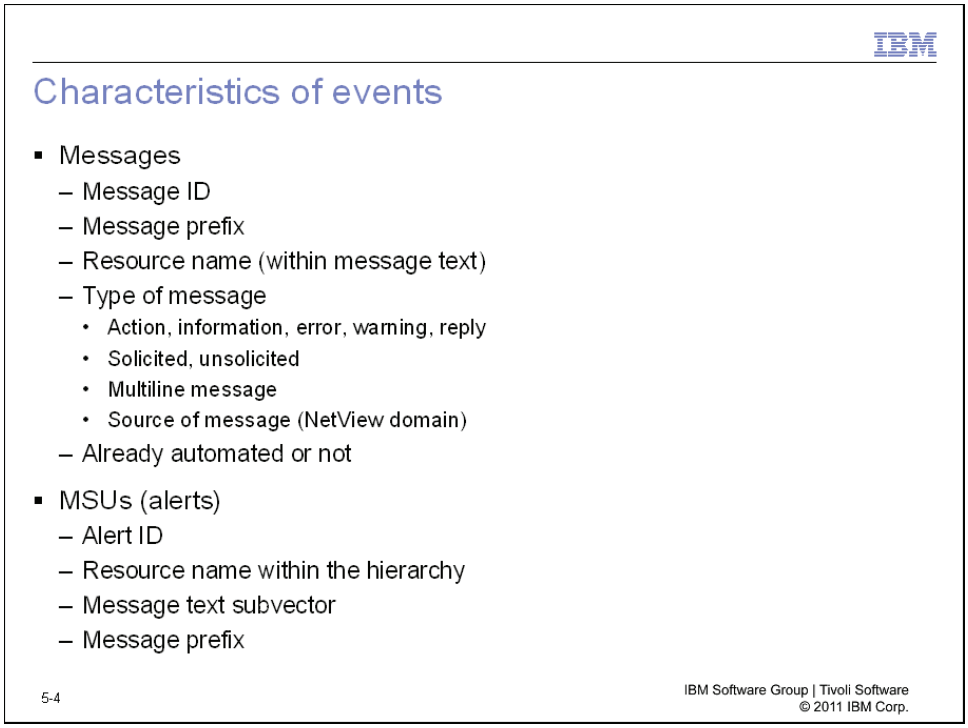
Lesson 1: Coding automation table statements



Considerations when creating automation table statements are as follows:

- Events to automate: Events can be noted with a single message, multiple messages, a multiline message, an alert and a message, or combination.
- Actions to take: You can choose to issue commands directly from the automation table. You can issue a REXX EXEC and use it to issue the commands and validate their responses. The action to take might be routing the event to a focal point NetView in a multisystem environment.
- Areas to run the actions: By default, the actions run under the primary receiver of the message. That task might not be able to fully support the actions or, there might be no primary receiver.
- Ways to process the event: You could suppress the event. You could have it routed to an operator and held on their screen. You might use other means of processing the event.

Characteristics of events

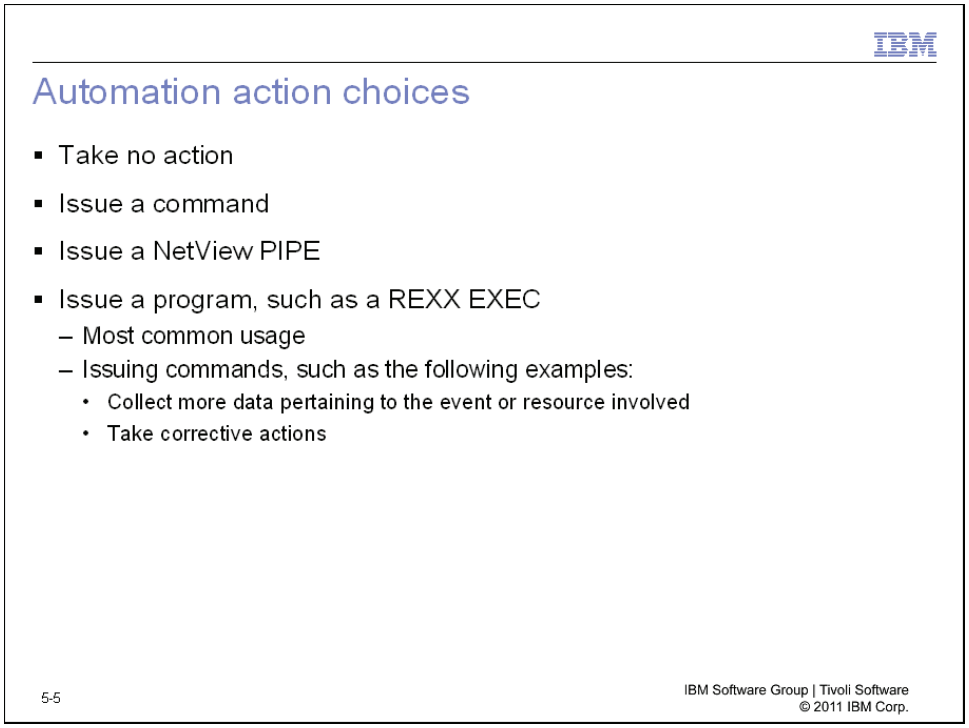


This slide lists characteristics of messages and MSUs that you can use for automation. The most common automation is message-based, using the message ID or specific text within the message string. The event that drives the automation table can be a Common Base Event (CBE) or TCP/IP trap that translates into a message or MSU.

All solicited and unsolicited messages are processed on the automation table except the ones as follows:

- Those written directly to DSILOG by the following items:
  - MSG LOG command
  - PIPE LOGTO output
- Intradomain copies created by the following items:
  - ASSIGN SEC or COPY command
  - MSGROUTE command
  - Automation table ROUTE actions

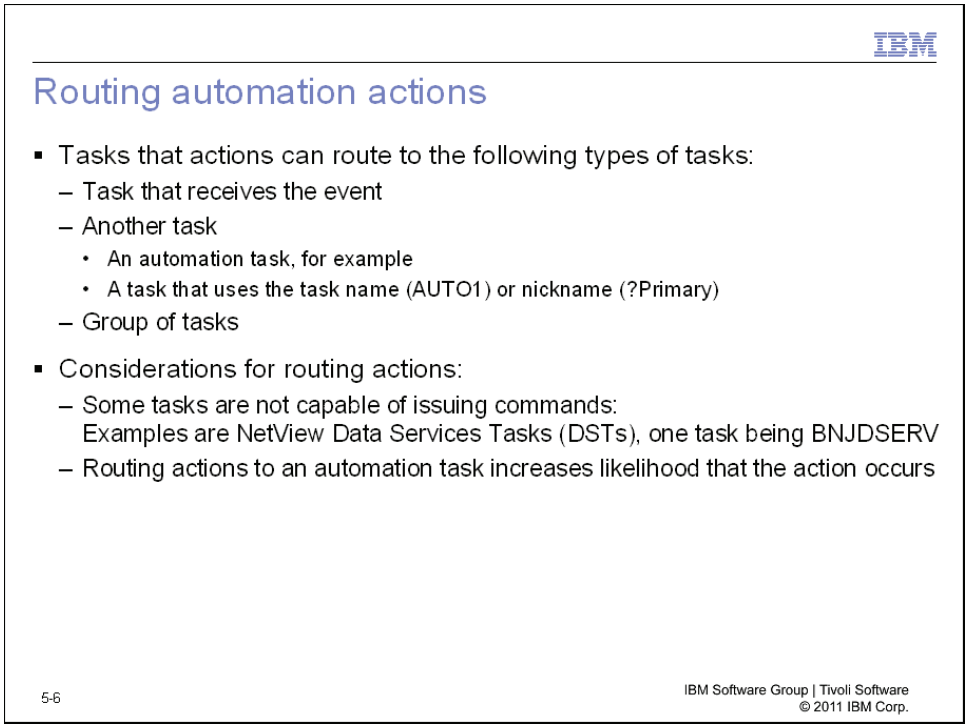
Automation action choices



Most automation actions call a program, such as a REXX EXEC, to perform the following tasks:

- Further interrogate the event. Perhaps it is a multiline message and you have a REXX EXEC to parse the multiline message for a resource name or resource status.
- Issue one or more commands for gathering more information about the event. For example, the event might be a symptom of an previously known problem.
- Issue one or more commands for taking corrective actions, waiting for the response to the command before proceeding to the next one.

Routing automation actions



The automation table supports routing the event to one or more NetView tasks by coding the tasks as part of the automation table statement. The routing can go to the following destinations:

- The task that receives the event
- The primary receiver
- A different NetView task (for example, an automation task)
- A group of tasks
- An automation task by means of a *nickname*

  The nickname is defined in CNMSTYLE. The task nickname function is similar to a variable. An example is ?Primary.

Tip: Use the QOS command for querying the autotask that is associated with a nickname. For example, to display the autotask known as ?Primary, issue the following command:

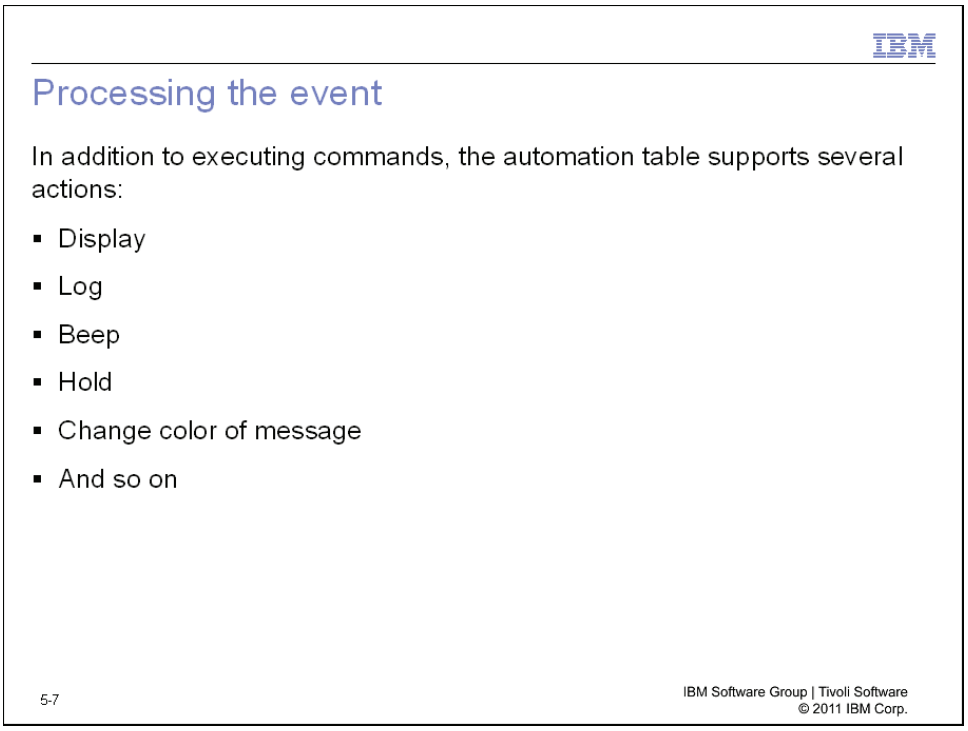
```
QOS OPID=?PRIMARY
DWO837I AUTO1 IS DEFINED AND LOGGED ON TO NETVIEW
```

Actions from the automation table are, by default, routed to the task that receives the event. For messages, this task can be the primary receiver. If no primary receiver is defined, the default task

could be the task that started CNMCSSIR for z/OS messages. Or the default task could be the PPT task for VTAM messages.

In the case of alerts and MSUs, the default task for actions is the BNJDSERV task. Because BNJDSERV does not support commands, always code a ROUTE statement for alert and MSU automation.

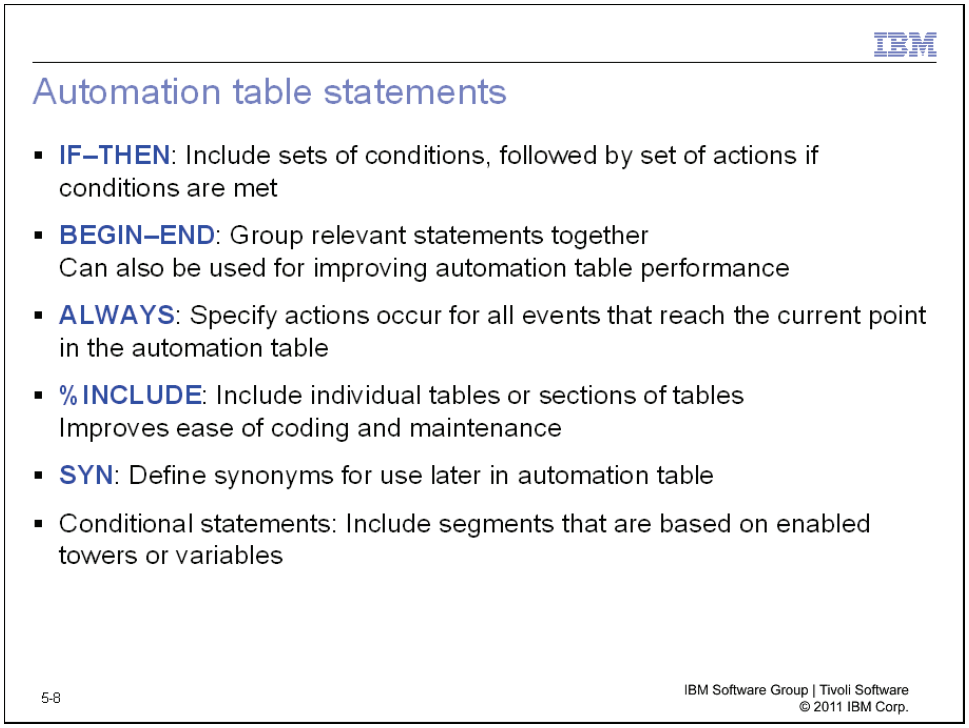
Processing the event



Processing the event refers to the handling of the event. Should it be displayed or suppressed? Should it be logged? Should it be held on an operator screen until action is taken to resolve the problem?

Most of these flags are binary, for example, DISPLAY(Y) versus DISPLAY(N).

Automation table statements



You can code the following statement types in an automation table:

- IF-THEN statement: A IF-THEN statement tests for specified conditions (characteristics of a message or MSU). If the condition matches the statement, one or more actions can be taken. The IF-THEN statement can also specify where to route the actions.

  The IF-THEN statement is the most prevalent statement in an automation table.
- BEGIN-END section: Groups statements for processing.
- ALWAYS statement: Specifies action that occurs for all messages and MSUs that reach this statement.
- %INCLUDE statement: Includes separately coded and maintained sections of the table (so that the table can be built from sections). The included members must also be in DSIPARM.
- Conditional statement: Identified by %>, this statement includes specific statements conditionally. Examples include statements that are used with TOWER definitions that are coded in CNMSTYLE.

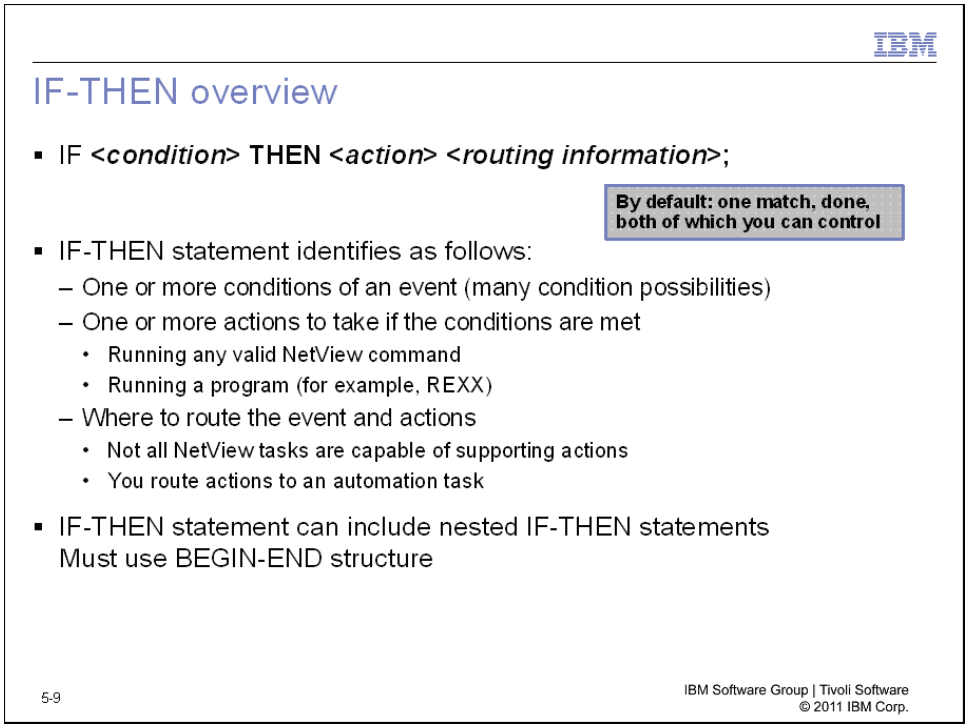
  Conditional statements are also usable with user-specified entries for creating separate flows through an automation table when shared among multiple NetView domains.
- SYN statement: Defines synonyms for later use in the table.

Automation table statements can be entered in upper or lower case. When mixed-case statements are used, they are converted internally to upper case as this unit shows. Text string comparisons are always case-sensitive. Error messages might display the statement in upper-case format.

Some elements must always be entered in correct case and are not converted. Examples include the following text types:

- Text strings, including message IDs
- Member names that are used in %INCLUDE statements
- Synonym names, which are case-sensitive

IF-THEN overview

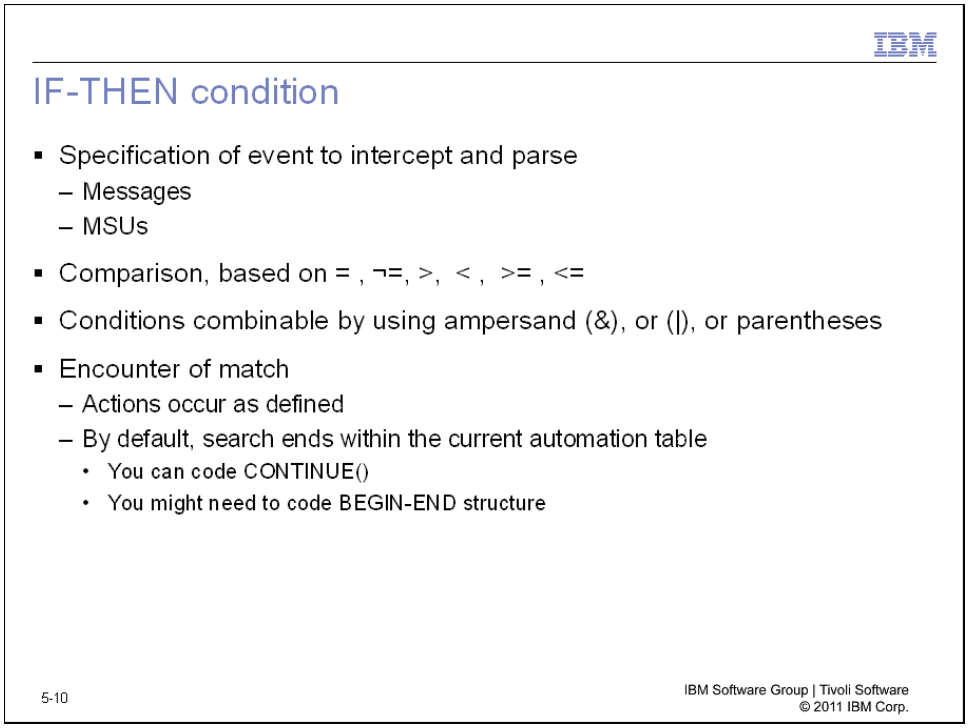


The IF-THEN statement specifies the following items:

- One or more tests that are to be made against a message or MSU (by using characteristics, such as message ID)
- The actions that are to be taken if the test is true
- The place or places to route the actions
- The way to process the event

The IF-THEN statement can include nested IF-THEN statements when you use the BEGIN-END structure in the automation table.

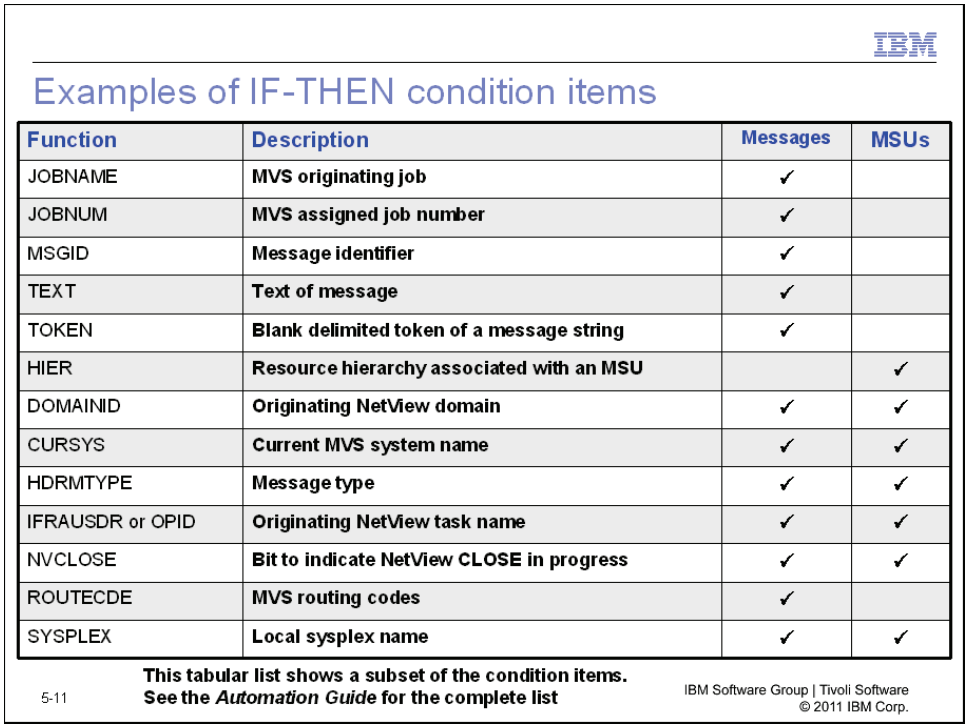
IF-THEN condition



By default, after a match is found, the search of the current table stops and the search begins with the first statement in any subsequent (logical) table. You can control this process with the CONTINUE() function. If CONTINUE(YES) is coded as part of the action, the search continues at the next statement. Note the following syntax rules:

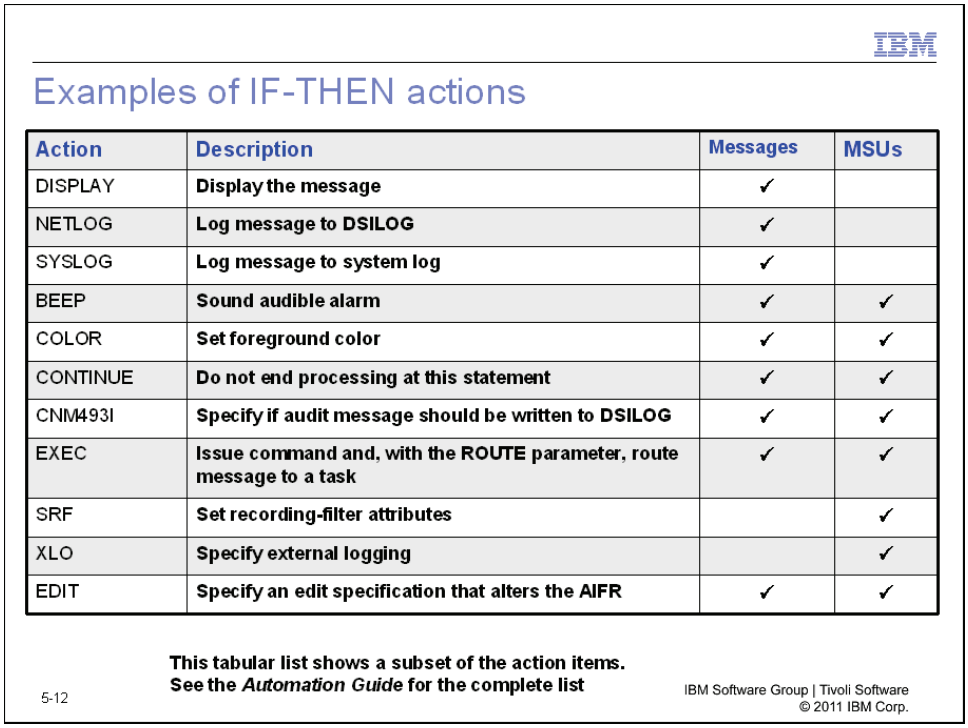
- Valid comparisons are =, ¬ =, >, <, >=, and <=.
- Tests are combinable by using a logical AND (&) or a logical OR (|) operator.
- Conditions are specified by using a bit string or parse template
- For multiline messages, the parse template can trap only the first non-blank line of the message.(An example later in this unit shows the use of the ACQUIRE function to access the remaining lines.
- Each statement ends with a semicolon (;)

Examples of IF-THEN condition items



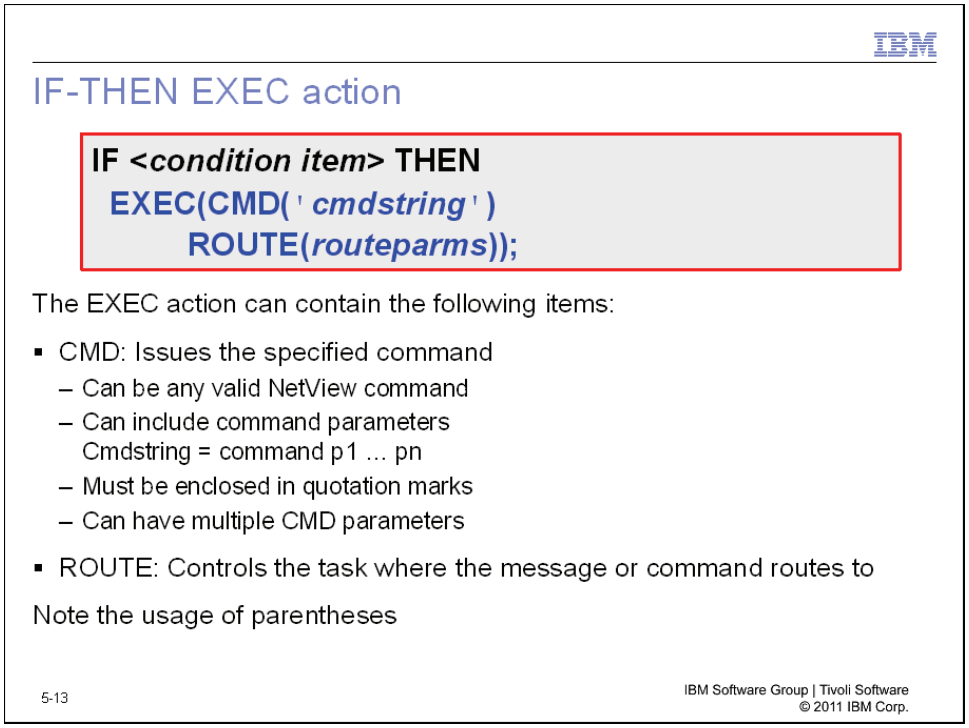
This slide illustrates a subset of the available condition items for an IF-THEN statement and whether each applies to messages or events. The lesson provides examples of using JOBNAME, MSGID, TEXT, TOKEN, HDRMTYPE, and more. Descriptions of all condition items is in the *Automation Guide* manual.

Examples of IF-THEN actions



This slide illustrates some of the actions that are possible when the automation table receives a message or MSU. Some actions might revert to defaults. Even if a matched automation table statement does not specify an action, an action might occur because of settings that the DEFAULTS or OVERRIDE commands establish. Exact descriptions of actions are in the *Automation Guide* manual.

IF-THEN EXEC action



The purposes of the EXEC actions are as follows:

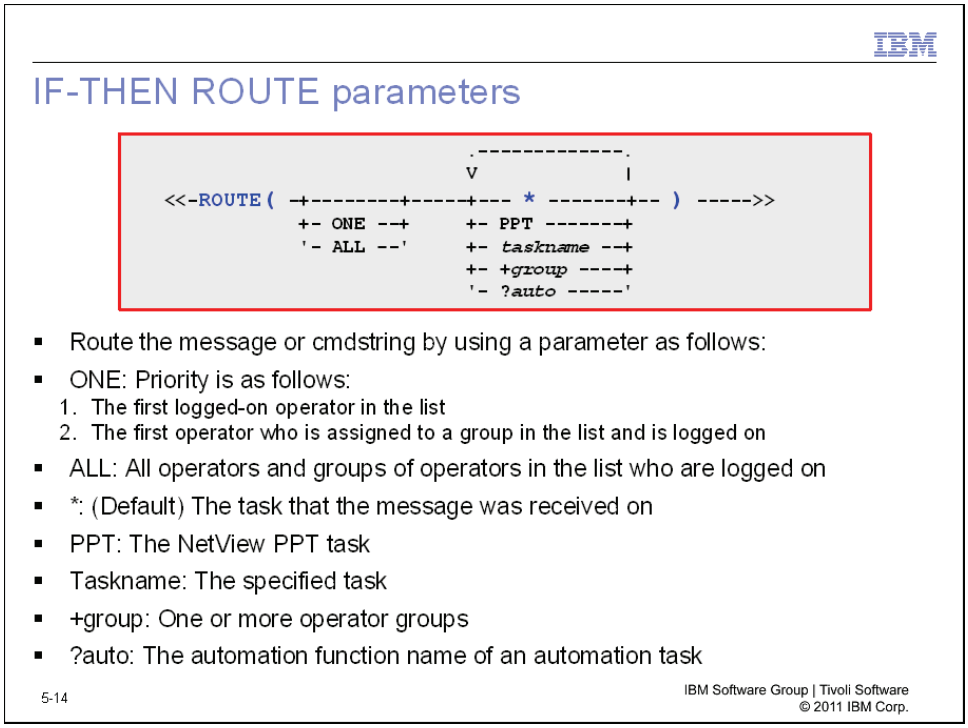
- Issuing commands based on a match to the *condition item*.

  The command string must be enclosed in quotation marks. The message text can be parsed and stored into variables to be passed to the command string as parameters.

- Controlling where the commands run.

Note: The EXEC statement must contain an even number of parentheses, commonly overlooked by new users.

IF-THEN ROUTE parameters



This slide explains the possible ROUTE parameters. This type of parameter routes the message and the command to the tasks specified. The most common usage is routing the message or action to one task from a list that can include the following destinations:

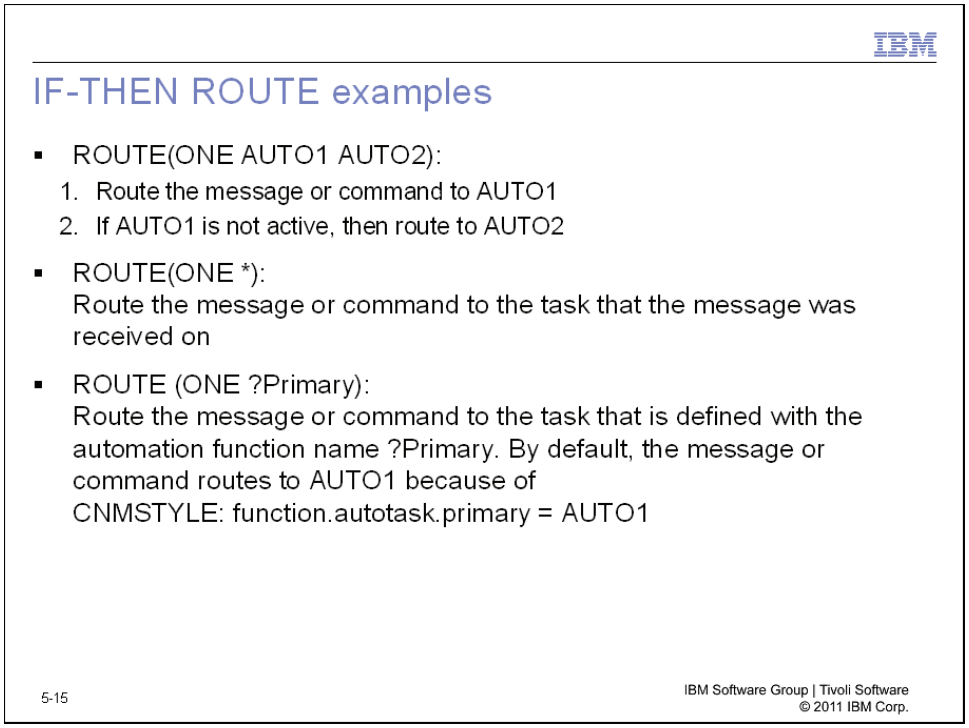
- The task that is to receive the message (*)
- A list of task names (for example, AUTO1 AUTO2)
- An operator group (for example, +CICSOPS)
- an *automation function* name

One of the keywords, ONE or ALL, must be specified.

Tip: Use the QOS command to query the autotask that is associated with a nickname, For example, to display the autotask known as ?Primary by using the following code:

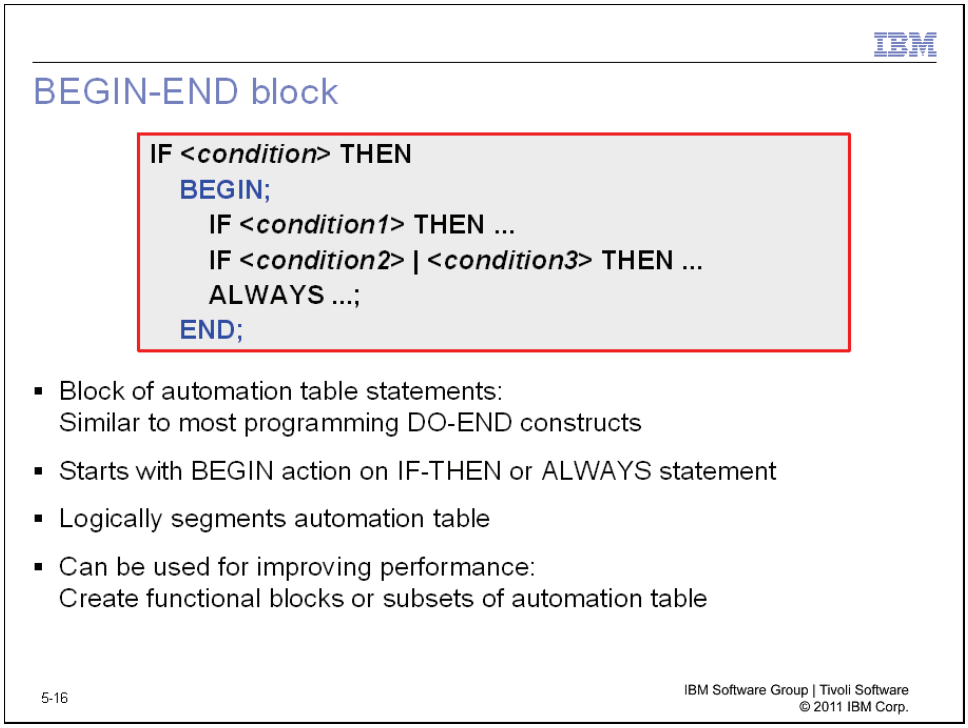
```
QOS OPID=?PRIMARY
DWO837I AUTO1 IS DEFINED AND LOGGED ON TO NETVIEW
```

IF-THEN ROUTE examples



- ROUTE(ONE AUTO1 AUTO2): The message and action route to the first logged-on task between AUTO1 and AUTO2.
- ROUTE(ONE * AUTO1 AUTO2): The message and action route to one of three tasks: the first logged-on operator of: the task receiving the message, AUTO1, or AUTO2.
- ROUTE(ONE *): The message and action route to the task that is to receive the message.
- ROUTE (ONE ?Primary): The message and action route to the task known as the primary task, which is defined in CNMSTYLE as AUTO1.
- ROUTE(ONE +MVSOPS): The message and action route to the first logged-on operator in the +MVSOPS operator group list.

BEGIN-END block



You can use a BEGIN-END block to logically segment an automation table similar to using a DO END programming construct.

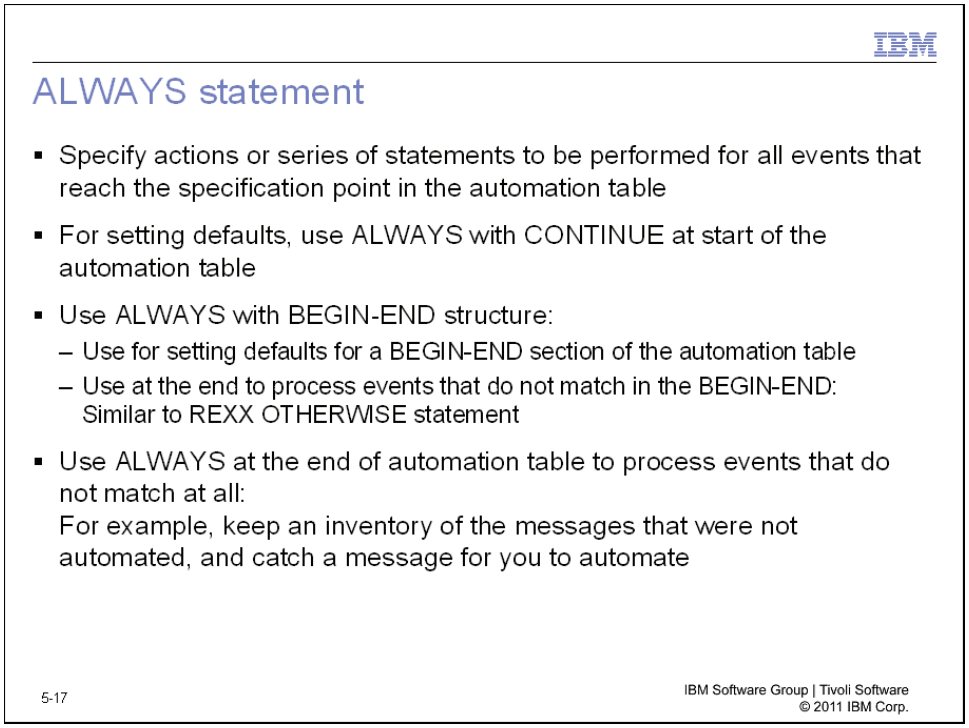
Use BEGIN-END blocks to improve the performance of an automation table. For example, you can check a message prefix and define a BEGIN-END group to process all messages that match the prefix. If the prefix does not match, not all of the more specific checks (located in the BEGIN-END group) occur:

```
IF MSGID = 'EZL' . & TEXT=MESSAGE THEN
BEGIN;
* Process all NetView EZL messages here
...
END;

IF MSGID = 'IST' . & TEXT=MESSAGE THEN
BEGIN;
* Process all VTAM (IST) messages here
...
END;

IF MSGID = 'DSI' . & TEXT=MESSAGE THEN
BEGIN;
* Process all NetView DSI messages here
...
END;
```

ALWAYS statement



For all messages and MSUs that reach the specification point in the automation table, specify either actions or a series of statements that always process. You can use an ALWAYS statement with the CONTINUE operand at the start of the automation table or a BEGIN-END section for setting defaults for that table or section. Doing so sets an action for all messages and MSUs that reach that point.

Code an ALWAYS statement at the end of a table for processing all messages and MSUs that have not been trapped in that table. Examples are as follows:

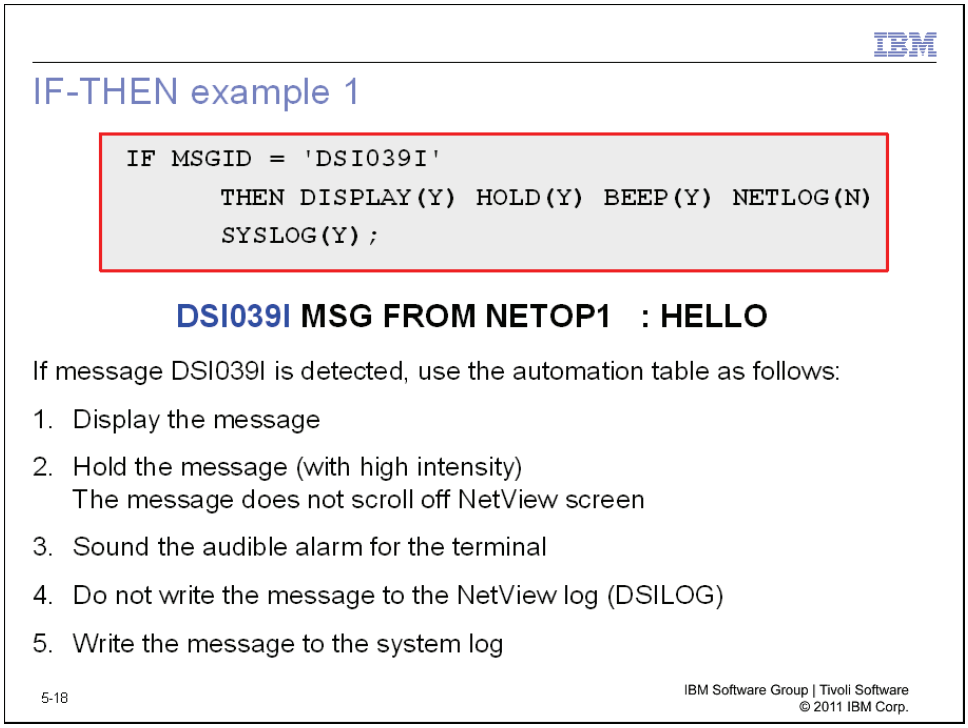
- Defining an automation table defaults for logging and continuation:

```
ALWAYS SYSLOG(Y) NETLOG(Y) CONTINUE(Y);
```

- Defining an automation table default for CNM493I message:

```
ALWAYS CNM493I(Y);
```

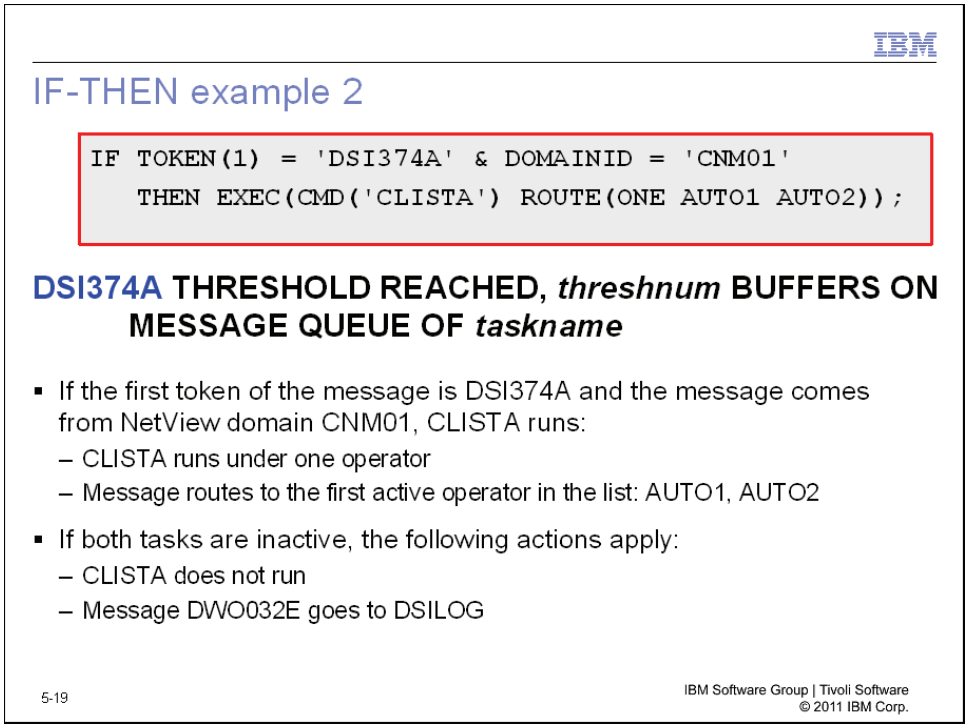
IF-THEN example 1



Message DSI039I indicates that a message has been received from a NetView task using the MSG command. The following actions are to occur:

1. Display the message to the recipient at their NetView screen.
2. Hold it so that it does not flow off the screen as further messages arrive.
3. Sound the terminal alarm as the message is displayed.
4. Write the message to the system log but not the network log, DSILOG.
5. Do not issue a command.

IF-THEN example 2



In this example, the IF clause contains a test for the first token of the message. In most cases, this is the same as the message ID. An exception is a WTOR message, where the first token is the reply number, and the message ID is the second token.

Two tests are made against the message (TOKEN(1)='DSI374A' and DOMAINID='CNM01'), Both conditions need to be true in this case. You can make many tests and combinations by using AND or OR conditions.

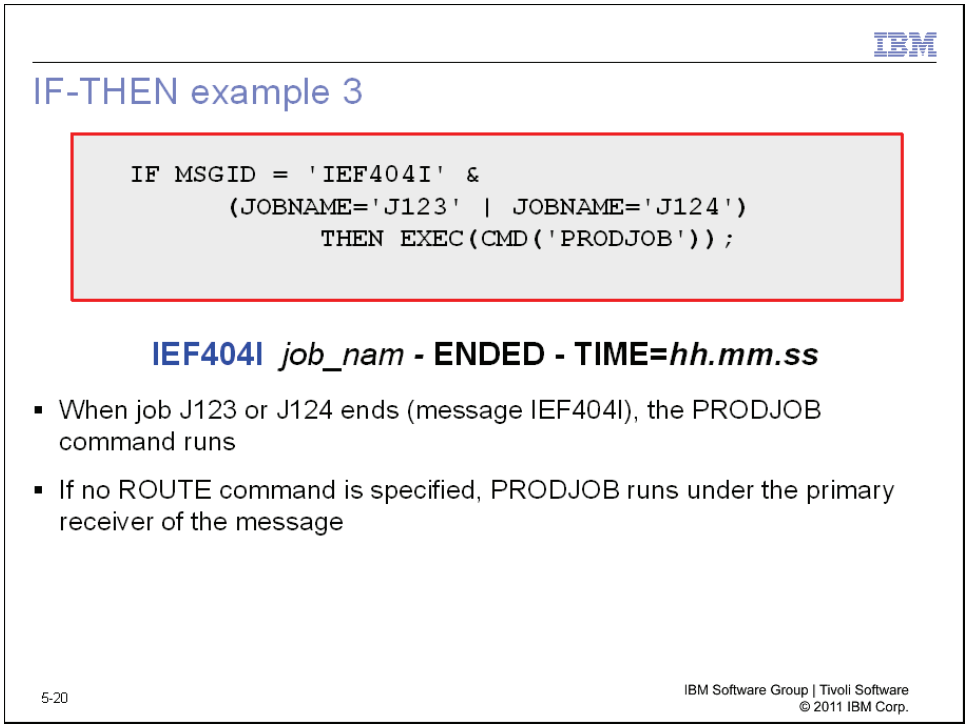
In this example, the action is running a command named CLISTA. This could be a REXX EXEC, command list (Clist), or any valid NetView command.

The ROUTE operand designates the task where the command runs. The ROUTE operand specifies the ONE keyword, which indicates that the command runs on only one of the tasks listed. The tasks test in the order listed. If an operator is active, the command is scheduled to run on that operator task.

In this example, if the AUTO1 task is active, the command runs only there. If AUTO1 is not active, CLISTA runs under the AUTO2 task if it is active. If neither task is active, the command does not run. The DWO032E message is written to the NetView log.

If ALL is specified instead of ONE, CLISTA runs on each active task in the list. An alternative approach is to specify ROUTE(ONE * AUTO1 AUTO2). The action then routes to the task that receives the message first, known as the *primary receiver* for the message.

IF-THEN example 3

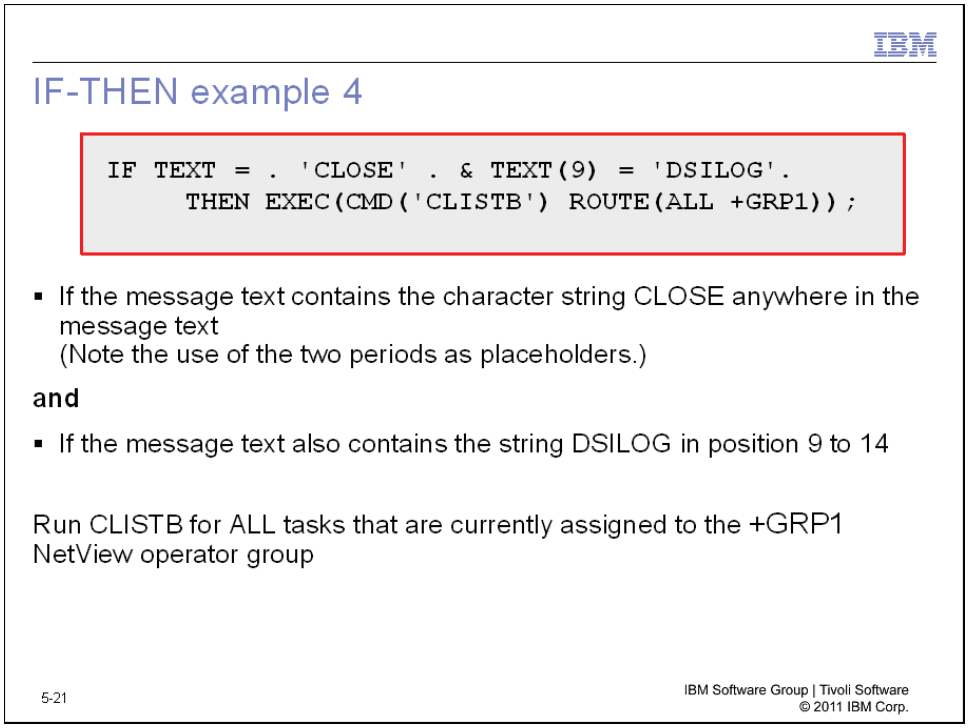


This example includes more complexity in the set of statements. The test is for message ID IEF404I, which occurs for only jobs J123 or J124.

The action causes the PRODJOB command to run. No operator is specified with a ROUTE, which causes the PRODJOB command run under the task that receives the IEF404I message, known as the primary receiver, which is almost always designated to be an active autotask.

You can achieve same effect if the CMD clause has a ROUTE(ONE *) specification. In this case, the * indicates the assigned primary operator.

IF-THEN example 4



In this example, the actual text of the message is checked instead of a message ID. The message text is checked for two character strings, CLOSE and DSILOG, using the TEXT conditional:

- **TEXT = . 'CLOSE' . :** Using the periods before and after the character string specifies that the word CLOSE can be *anywhere in the message string*. In other words, any text can precede or follow the string CLOSE.
- The following examples contrast the period positions:

  . 'CLOSE' specifies that the message must *end* with the character string.

  'CLOSE' . specifies that the message must *begin* with the character string.
- **TEXT(9) = 'DSILOG' . :** Tests the text of the message starting at character position nine. This is checked for the string DSILOG. The period character that follows the string DSILOG is called a placeholder. The placeholder indicates that the rest of the text (to the end of the message) can be any set of characters and is ignored.

  If the period is omitted, the test tests the message, starting at position 9 is DSILOG, with nothing following.

In this example, the THEN clause requests the CLISTB command run on ALL of the operators in operator group +GRP1. Routing actions to operator groups can be useful because +GRP1 might be a group whose members change with time. For example, the group might contain first-shift operations. If none of the members in the group are active, or if the group is empty, no command runs. The DWO032E message is issued.

Important: With this example, a STOP TASK=DSILOG command produces two messages that drive the automation table and matches this IF-THEN statement. This statement also compares against every message that contains the character string CLOSE.

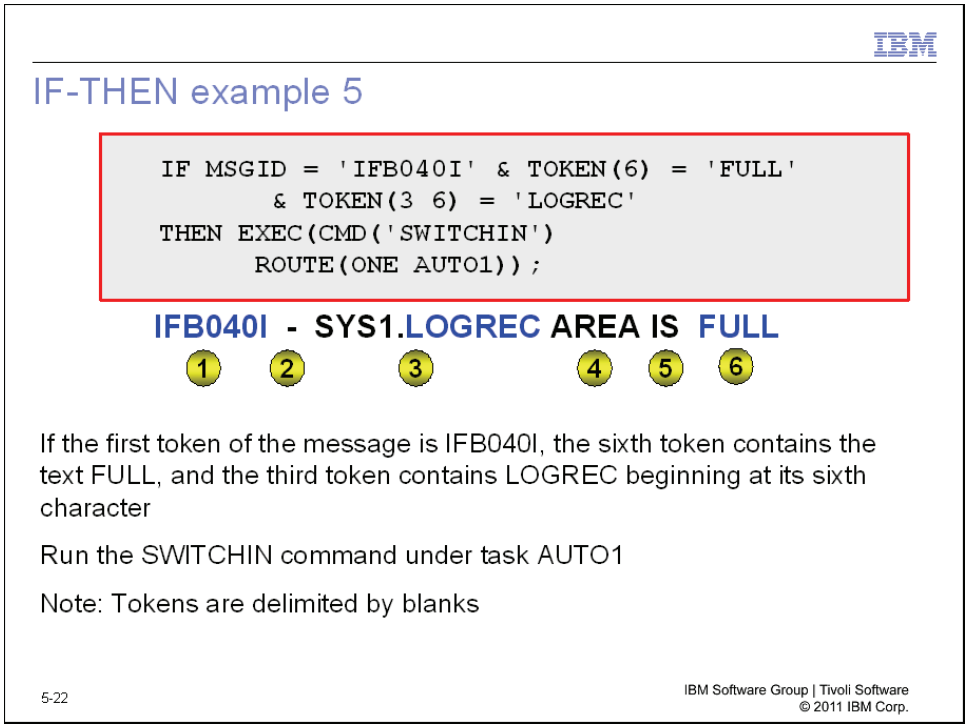
```
STOP TASK=DSILOG
DSI660I STOP TASK ISSUED FOR DSILOG BY NETOP1
DSI661I STOP TASK ISSUED FOR DSILOG BY NETOP1
DSI531I 'DSILOG' : 'DST' IS TERMINATING
DWO520I DSILOG : VSAM DATASET 'CLOSE' COMPLETED, DDNAME =
'DSILOGP' RETURN CODE = X'00', ACB ERROR FIELD = X'00'
DSI556I DSILOG : VSAM DATASET 'CLOSE' COMPLETED, DDNAME =
'DSILOGP' RETURN CODE = X'00', ACB ERROR FIELD = X'00'
DSI200I TASK DSILOG HAS TERMINATED
```

The DWO520I and DSI556I messages result in a match to the IF-THEN statement in this example. To eliminate this result, code an additional test for one of the message IDs. This example shows how to test a message ID, along with the text of a message, to prevent erroneous messages from matching your IF-THEN statement.

To avoid this condition, specify a MSGID to trap as in this following example:

```
IF MSGID = 'DSI556I' & TEXT = . 'CLOSE' . & TEXT(9) = 'DSILOG'.
THEN
   EXEC(CMD('CLISTB') ROUTE(ALL +GRP1));
```


IF-THEN example 5



In this example, the message text to be tested is as follows:

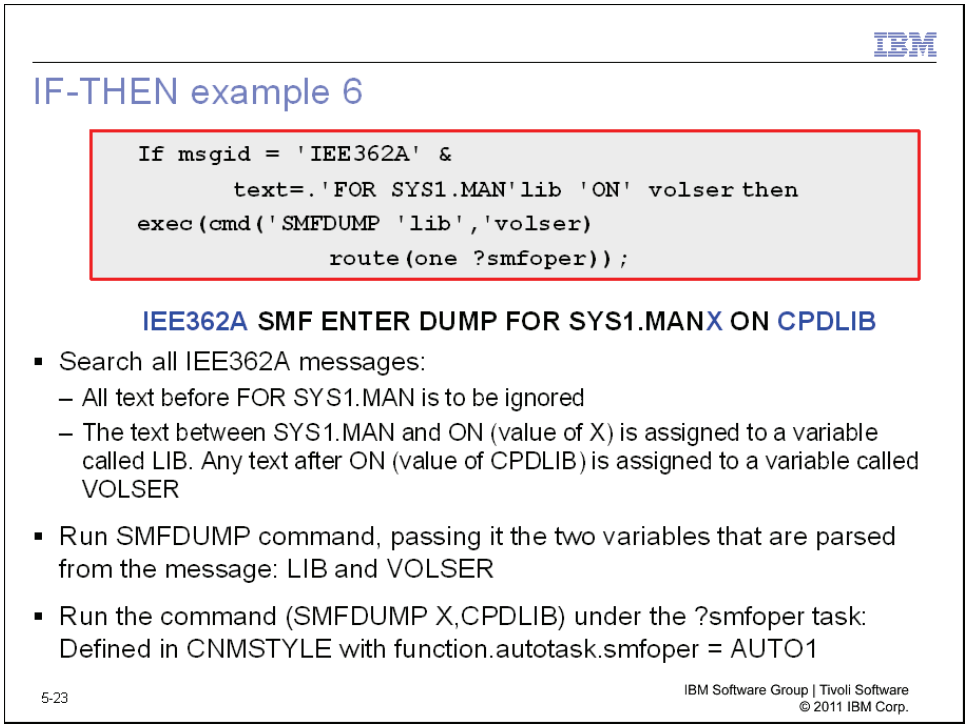
```
IEFB040I - SYS1.LOGREC AREA IS FULL
```

If the message ID is IFB040I and the sixth token is FULL, run the SWITCHIN command under the AUTO1 task. The TOKEN operand is used for searching for a blank-delimited field in a message.

To check for the exact composition of part of a token, you can use TOKEN(*x y*). For example, you can add an additional test of TOKEN(3 6) = 'LOGREC' to the preceding IF clause. This tests for the third token to contain exactly the characters LOGREC, starting in position six of the token.

The hyphen (-) after the message ID counts as the second, blank-delimited token. Also, the entire SYS1.LOGREC string is the third token because it contains no blanks.

IF-THEN example 6



This example shows how variable information is extractable from the message text, used for forming part of the command that is requested in the EXEC action. This example uses mixed-case text.

The IF clause selects a message (IEE362A) that indicates a SMF data set is full. The name of the data set is in the message text.

The TEXT keyword is used for parsing the message text and extracting information from it. Strings that are enclosed in quote marks must be matched. Text between them is assigned to either a placeholder (indicated by a period) or to a variable (identified by a variable name). The name should not have quote marks around it.

In the example, two variables, *lib* and *volser*, are parsed from the IEE632A message and passed as operands to the SMFDUMP command. The blanks that separate these operands must be explicitly specified, along with the command and its parameters. That is, all commands are constructed by appending strings and variables.

The command string (**'SMFDUMP 'lib','volser**) is built as follows:

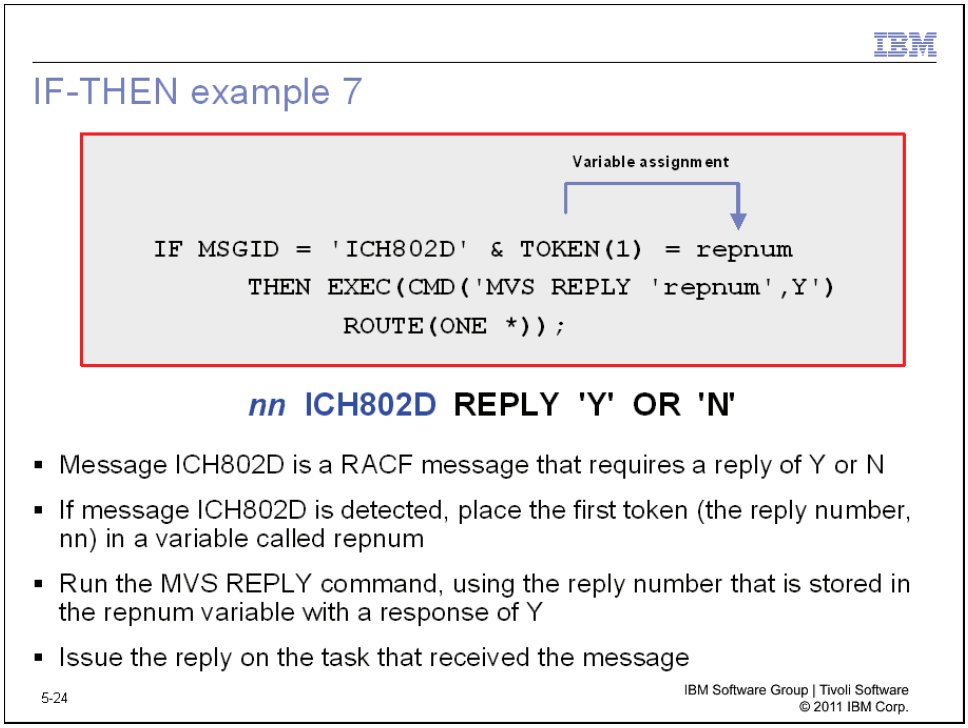
1. Beginning with the command name (SMFDUMP), including a blank at the end to act as a delimiter for the first parameter
2. Appending the *lib* variable as the first parameter
3. Appending a comma as a delimiter (,) for the SMFDUMP command

4. Appending the *volser* variable as the second parameter

Quotes are required in the command string only when using text strings. In this example, the command name (SMFDUMP) includes the blank and the comma that separate the command parameters.

Tip: The command routes to the ?smfoper. CNMSTYLE defines that ?smfoper is defined in CNMSTYLE as the AUTO1 task. You do not need to hard code the task name if you use the *function* name. If you use a function name, you do not have to change the automation table if you change the task name.

IF-THEN example 7



This example shows how to parse a WTOR message and assign the reply number to a variable. In this case, the ICH802D message, is similar to the following code:

```
33 ICH802D REPLY 'Y' OR 'N'
```

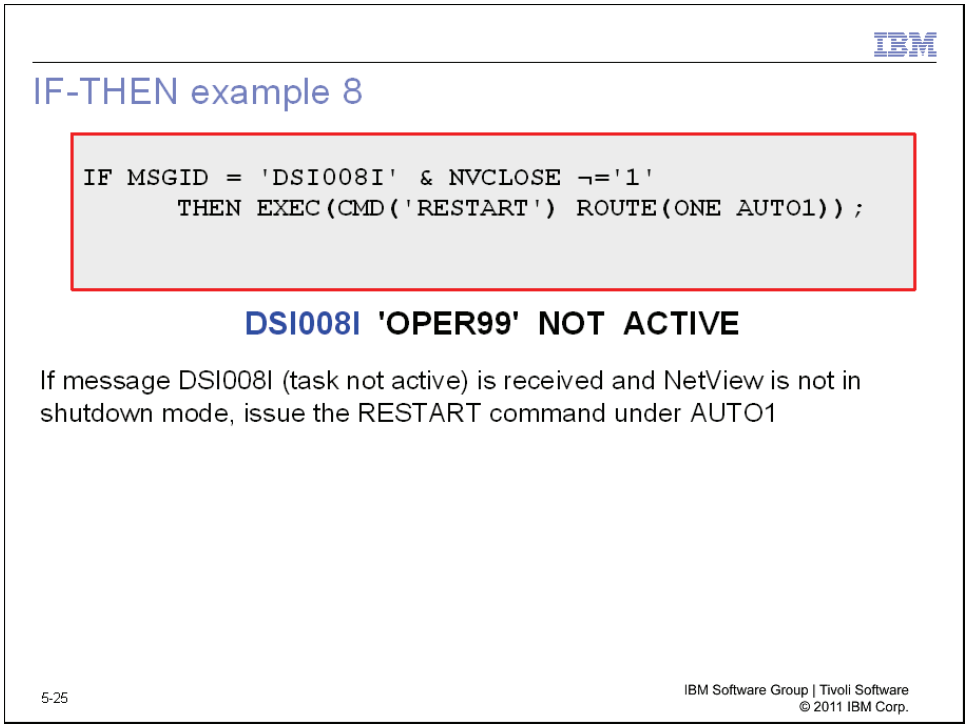
Because the reply number precedes the message ID, the reply number is in token one of the message string and the message ID in token two. The reply number is stored in a variable called REPNUM, A variable has no quotes (').

ROUTE(ONE *) indicates the MVS REPLY command is to be run by the task that the ICH802D message is assigned to. The REPNUM reply ID passes as an operand to the command. An example MVS REPLY command is MVS REPLY 33,Y.

All blanks in the command that are to run must be explicitly coded. There are no implicit blanks as there are in REXX statements. In this case, the command string is as follows

```
Token 1: MVS
Delimiter: blank
Token 2: REPLY
Delimeter: blank
Token 3: repnum variable
Delimeter: comma
Token 4: Y
```

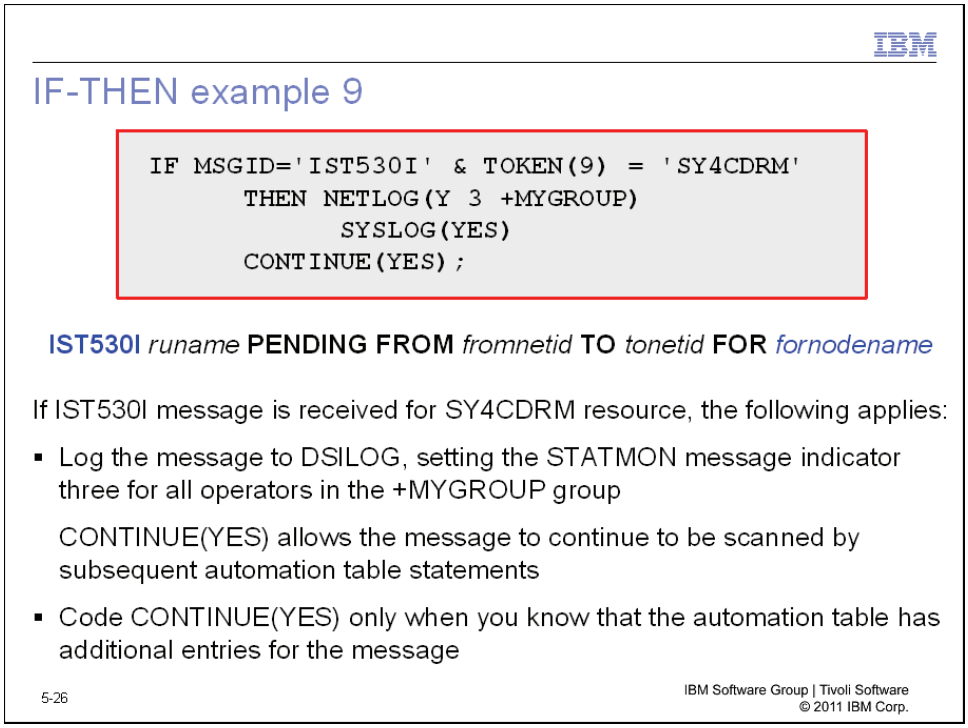
IF-THEN example 8



The NetView DSI008I message indicates a task is not active. NVCLOSE is a single bit flag that indicates if NetView is shutting down. Bit strings might also be compared. In this case, a test is for a condition that does not pertain to the message itself but to the current environment.

If the message ID is DSI008I and the flag NVCLOSE is not set (NetView is not shutting down), schedule the RESTART command. This command is to run on the AUTO1 task.

IF-THEN example 9

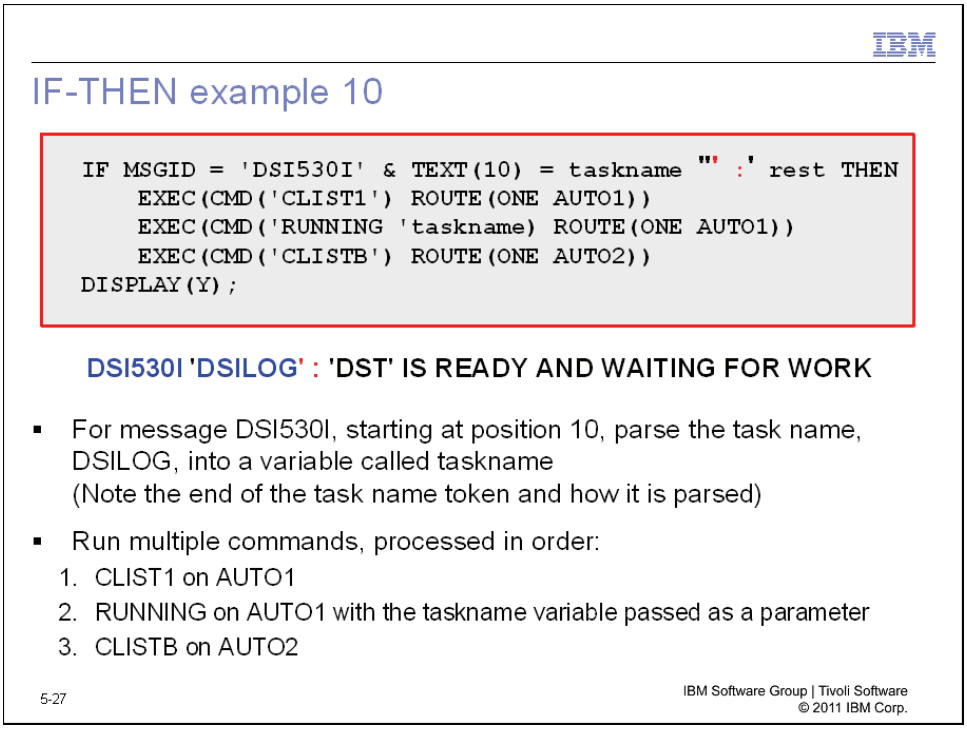


This example shows how to use CONTINUE(YES) to continue searching the automation table. If an IST530I message is for a VTAM resource called SY4CDRM (contents of token 9), log the message and set Status Monitor indicator three. The message goes to all operators that are assigned to the +MYGROUP operator group that are currently logged on. You must log the message to set an indicator. *IST530I is the first line of a multiline message.* Discussion about accessing data in the remaining lines of a multiline message occurs later in this lesson.

By default. the automation table processing ends with this match. CONTINUE(YES) allows processing to continue after the match. Any EXEC requests are scheduled, and processing continues with the next automation table statement.

If a later match in the table contradicts an action, the latest action prevails. For example, if a subsequent match specifies SYSLOG(N), that match is the one that is selected.

IF-THEN example 10



This example illustrates how to parse a message that contains single quotes in the message text. In this case, the task name is parsed, beginning at position ten using TEXT(10). Because the length of the task name is variable (up to eight characters long), parse and remove the trailing quote, colon, and the rest of the message as the following text shows:

```
TEXT(10) = taskname ''' :' rest
```

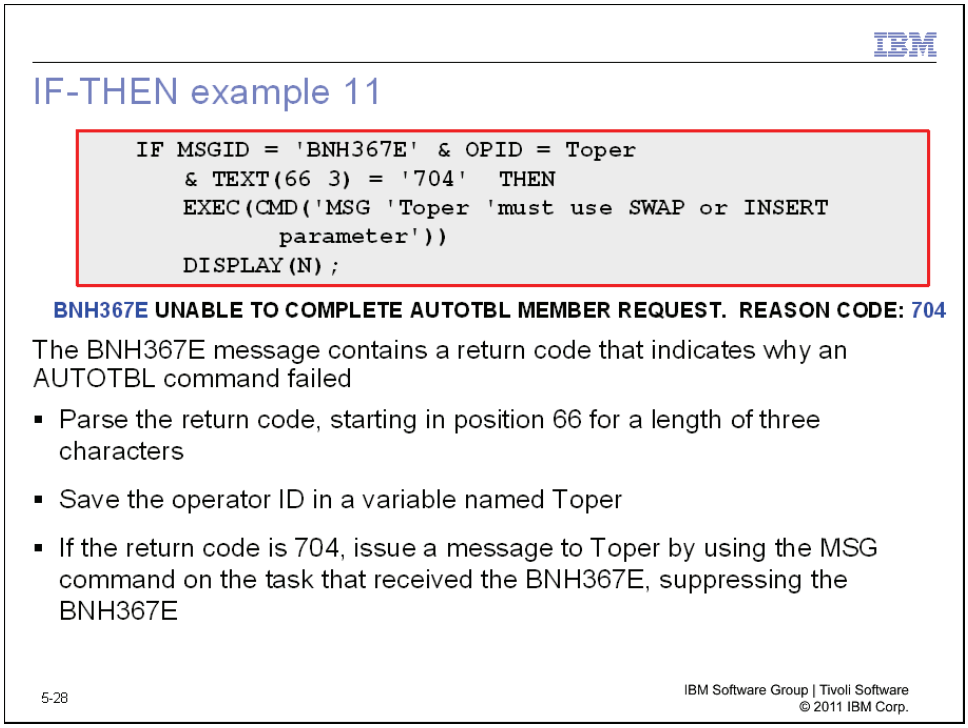
The first and fourth (last) quotes indicate the beginning and end of the text string. The second and third quotes indicate the presence of one single quote within the string. The *taskname* insert can be followed by one or more blanks if it has fewer than eight characters. The last character to parse is the colon (:).

This example also runs several commands on AUTO1 and AUTO2 tasks. Commands that are scheduled to the same task run in the specified order. For example, CLIST1 runs and completes on AUTO1 task before the RUNNING task.

CLISTB is scheduled to run on the AUTO2 task. It runs independently of the commands that are scheduled on AUTO1. CLISTB might run before CLIST1 starts, while CLIST1 runs, or even after CLIST1 finishes.

The automation table EXEC action schedules, or queues, a command to run on a task. Only one command can run at a time on a task at a time. The running of the command is said to be *asynchronous* from the processing of the message.

IF-THEN example 11



This example shows two parts as follows:

- Assigning the contents of an automation table function to a variable
- Reducing confusion that arises when using placeholders or periods within strings

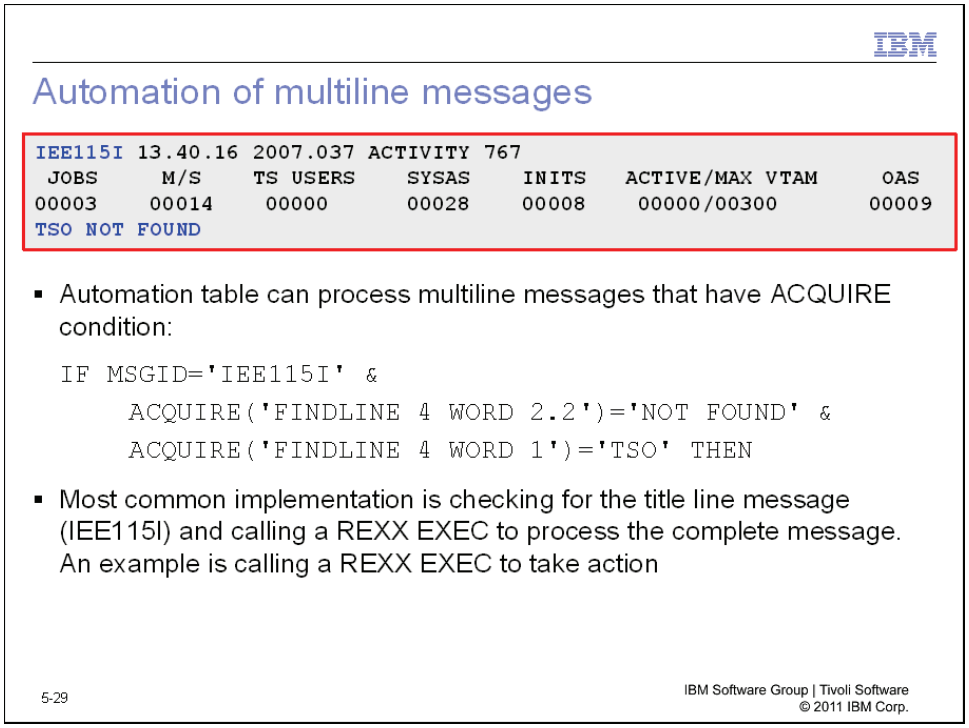
In this example, the OPID function is used for storing the operator ID to a variable, TOPER. TOPER is in the command string. The BNH367E message contains a return code that identifies whether or not an operator is permitted to issue an AUTOBL request. Several ways of accessing the reason code token are as follows:

- Use TEXT(66 3)='704' to test the contents of the message string to confirm if the reason code is 704. The starting position is at 66 for a length of three characters, You can also specify TEXT(66)='704 and omit the character length.
- Use the TOKEN() function: TOKEN(10)='704'
- Parse the text within the message: TEXT = . 'REASON CODE: 704'

The MSG command has mixed-case text as follows:

```
DSI039I MSG FROM NETOP1   : must use SWAP or INSERT parameter
```

Automation of multiline messages



This example shows automation that is based on the contents of a multiline message, using the ACQUIRE condition item. The ACQUIRE condition item uses a PIPE EDIT stage to examine data being automated. ACQUIRE is usable for examining any line in a multiline message.

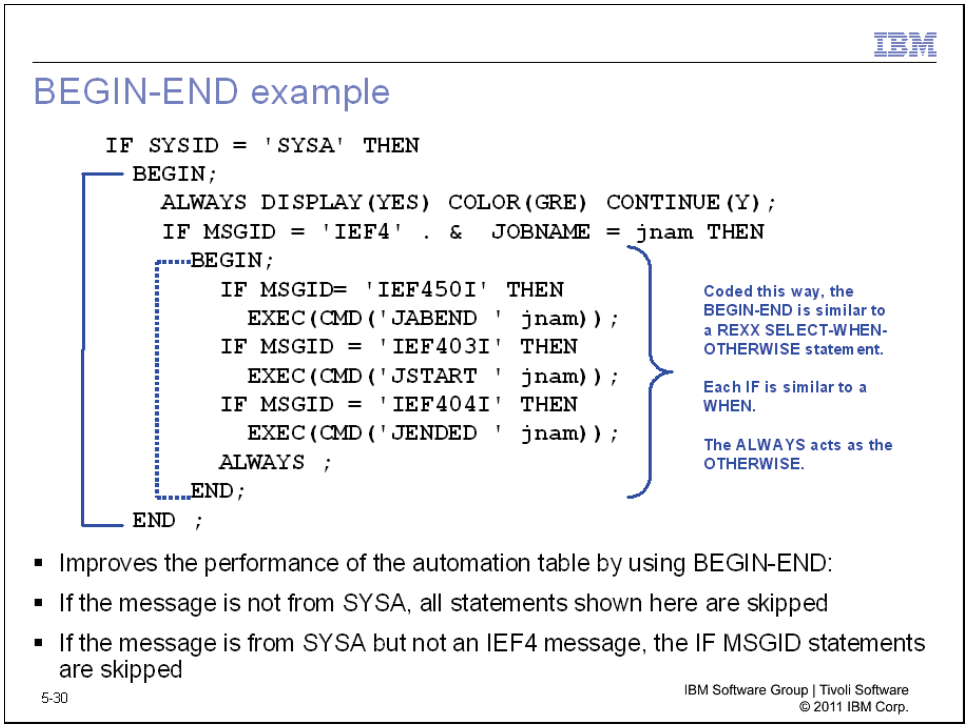
In this example, an MVS D A,TSO command has been issued. Line four of the IEE115I multiline message contains the text, TSO NOT FOUND. If TSO is not active, an action can be taken (for example, an MVS START command) to activate it. This example uses two ACQUIRE condition items to check the status of TSO. An alternative is checking WORD 1.3 for TSO NOT FOUND, using only one ACQUIRE condition item.

Using ACQUIRE avoids the need to create a REXX EXEC for processing the message and issuing a command. All of the logic is contained in the automation table and referenced.

Note: If you need to issue commands for gathering more information, you still need to code a REXX EXEC.

For more information about the PIPE EDIT stage, see the *IBM Tivoli NetView for z/OS Programming: Pipes* manual.

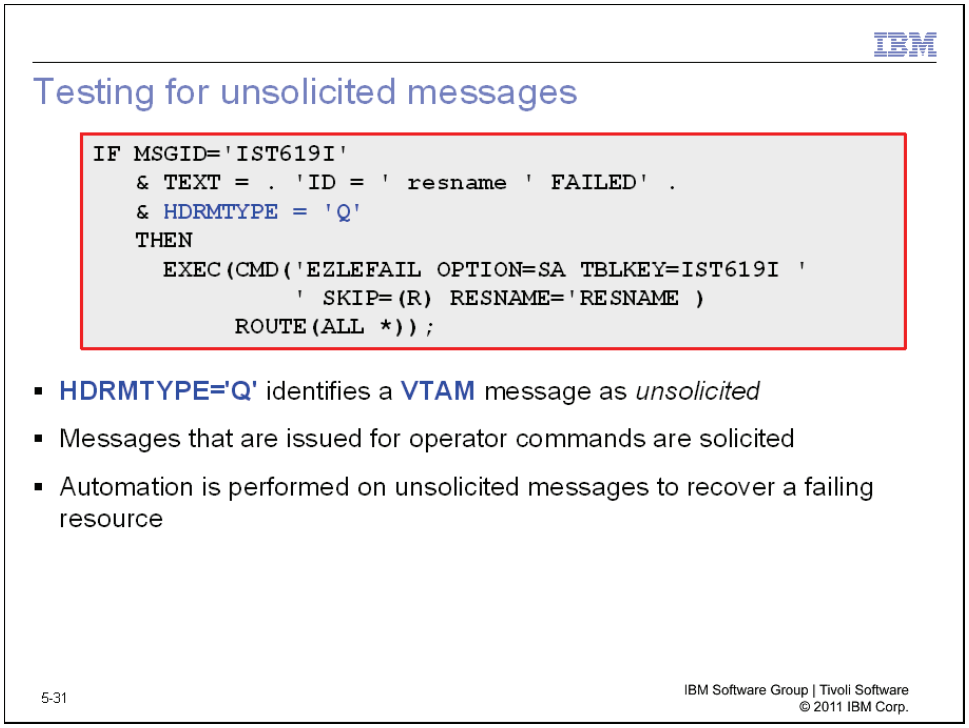
BEGIN-END example



This example shows how to specify BEGIN-END blocks, which can be nested. Use BEGIN-END blocks to reduce processing. For example, a conditional set of statements for SYSA is included. These statements are loaded and processed in all automation tables on all systems, but are effective when run on only the SYSA system. On systems other than SYSA, only the SYSID check (first IF-THEN clause) is performed. The remaining statements in the BEGIN-END block are ignored, improving performance of the automation table.

Automation table variables (*jnam* in the example) are usable by all actions within the BEGIN-END block.

Testing for unsolicited messages



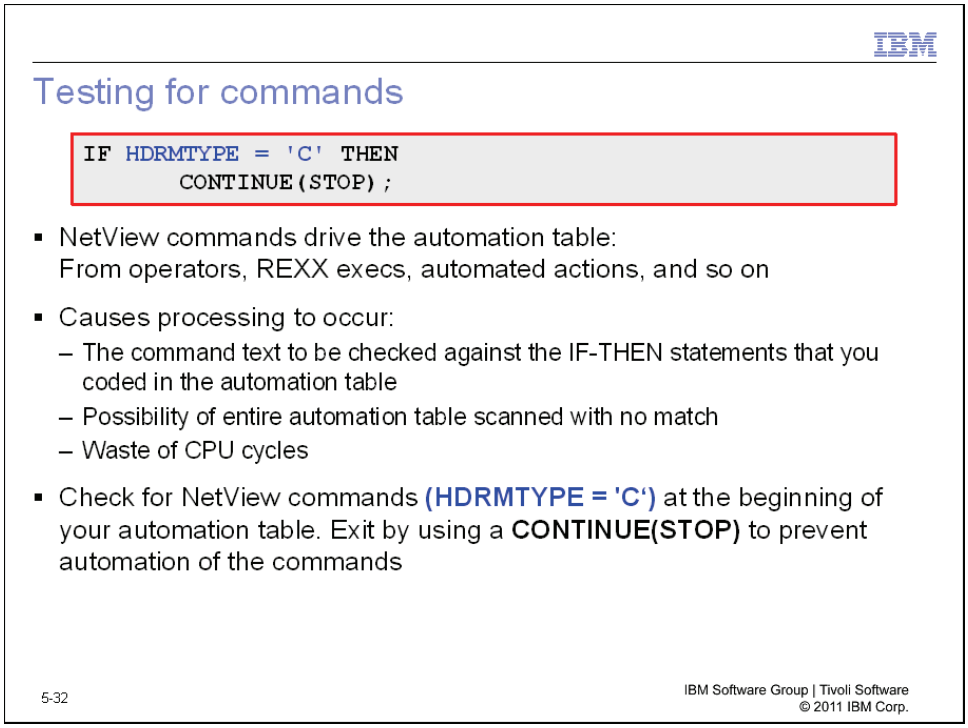
In some cases, you might want to automate only unsolicited messages. These messages identify actual events that have occurred, such as a VTAM resource failure, a database error, or an application abend.

Note: Most customers do not automate solicited messages. Solicited messages are the direct results of an operator (or automation operator) command.

All unsolicited VTAM messages have a HDRMTYPE message type of Q. You can test for that message type in the automation table, similar to the example. (HDRMTYPE documentation is in the *Automation Guide* and *Programming: Assembler* books.)

You can use the automation table functions to test AIFR bits for determining if the message is unsolicited or not. Use IFRAUIND and IFRAUIN2, bit 16 to indicate whether the message was unsolicited (1) or solicited (0).

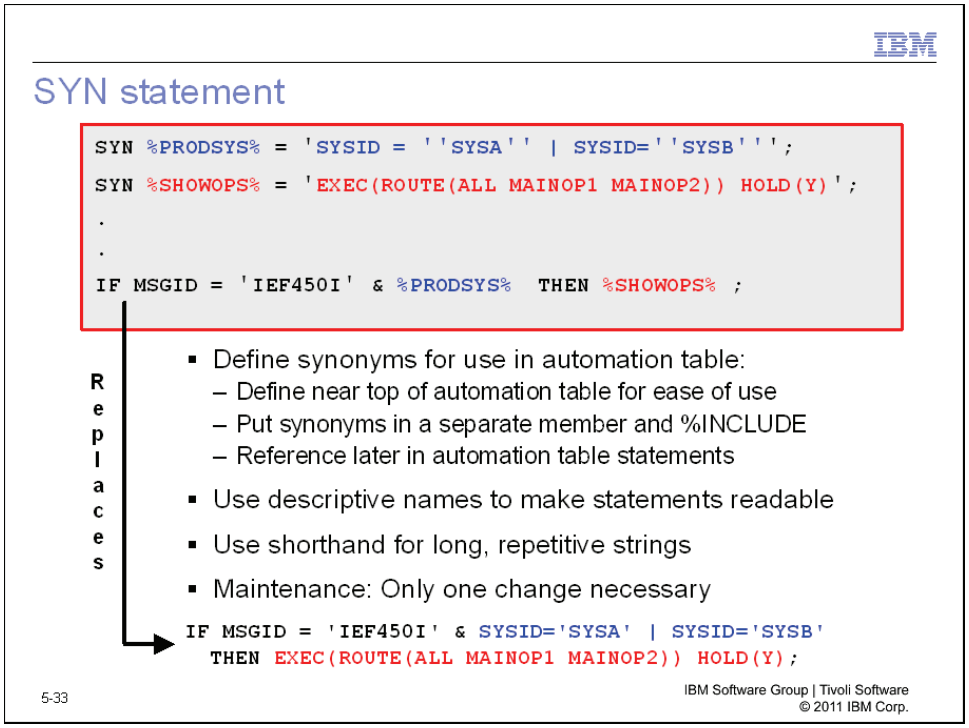
Testing for commands



Commands that NetView operators issue also drive the automation table. You can improve the overall performance of your automation table by coding a test for the commands at the beginning of your automation table. HDRMTYPE=C indicates that the event is a command from NetView or a message from a REXX EXEC running in NetView (using the SAY instruction).

Use code and action of CONTINUE(STOP) to prevent automation of the commands.

SYN statement



The SYN statement defines synonyms for use in the automation table. With synonyms, you can use shorthand for long, repetitive strings, which can help with maintenance and modification of an automation table. With synonyms, you can have a single specification of values that appear frequently in subsequent automation table segments. These values might vary in different systems or with changes in environment.

The syntax is as follows:

```
SYN %synname% = 'synvalue';
```

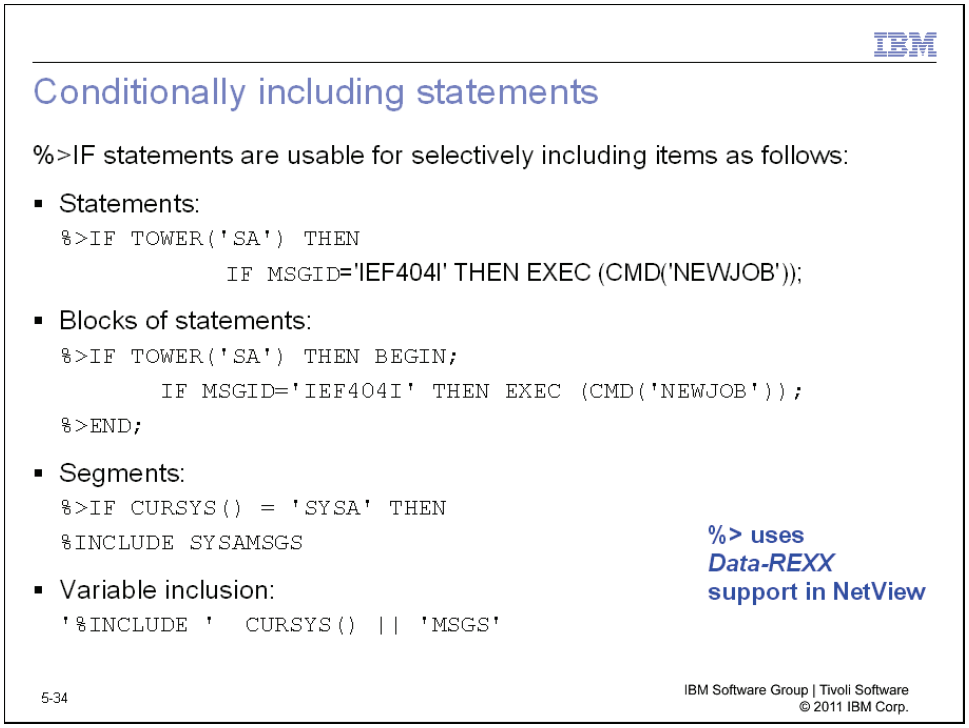
- *Synname* can have as many as 256 characters.
- *Synvalue* is the value of the synonym.
  - The value must not contain a semicolon.
  - You must exercise caution if a value contains quotes.

Additional significant information about synonyms are as follows:

- Must have a name and a value.
- Must be defined before it is used.
- Cannot be redefined later in the table (for example, in a later automation table segment), even with the same value.

When an automation table is loaded, the synonym value is substituted within that table wherever the name is encountered. Synonyms do not carry over to subsequent logical tables.

Conditionally including statements



You can conditionally include statements by using the %>IF statements. These statements designate actual inclusions in the automation table definition. Statements that are not selected do not load when building the automation table.

Using %> to conditionally include statements makes use of NetView *Data-REXX* functionality. The automation table is one of many NetView members that support Data-REXX. Use Data-REXX to install and customize NetView. For example, by enabling a NetView tower or subtower, you automatically enable all command definitions, automation table statements, operator definitions, and so on.

Statements that are excluded are not displayed when using AUTOCNT reports. Also, they cannot be disabled or enabled.

This slide shows several examples of conditionally included statements. More information follow:

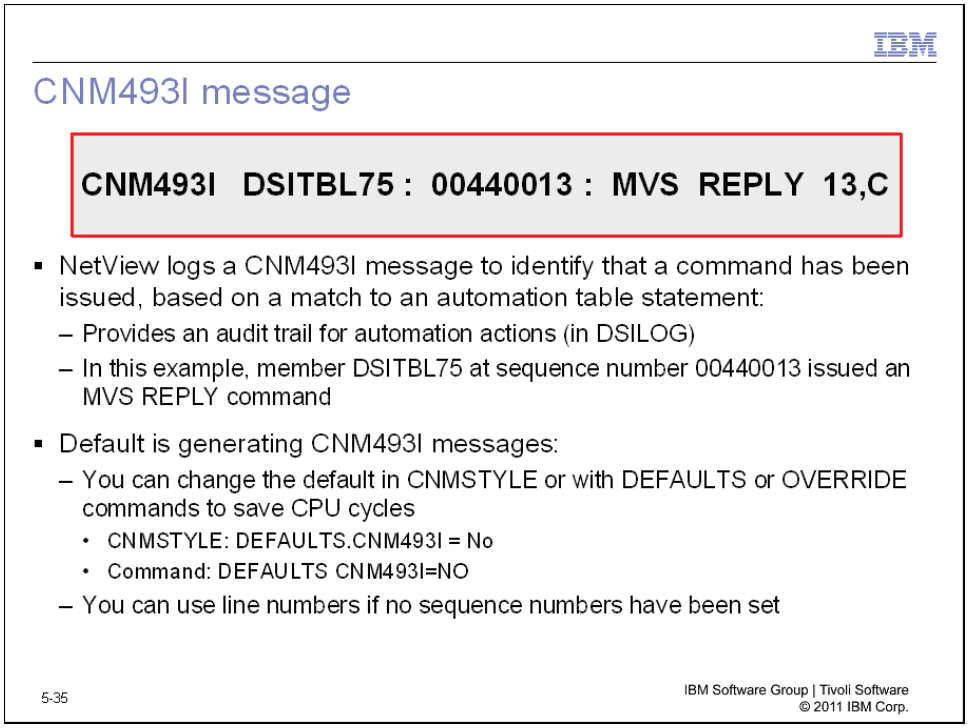
- If the SA z/OS tower is enabled, include a test for message IEF404I.
- If the SA z/OS tower is enabled, include a BEGIN-END section.
- If the current system is SYSA, include the SYSAMSGS member. You could also use the CURSYS() function as a variable when including SYSAMSGS.

All uses of these REXX functions require that the member read from DSIPARM and have a first line as follows that identifies the member as a *Data-REXX* program:

```
/*%DATA REXX
```

Note: The use of Data-REXX is not restricted to automation tables. All NetView DSIPARM members except CNMSTYLE can use this feature.

CNM493I message



A CNM493I message is written to the NetView log when a command is scheduled as a result of automation table processing. In the NetView log, the CNM493I message precedes the message that was matched. The CNM493I message includes information, such as the command that was scheduled, and the automation table segment and statement number where the match occurred. The statement number can be a sequence number or line number within the member. CNM493I messages are useful when checking that intended (or unintended) automation matches occurred.

In this example, an MVS REPLY command was issued from the DSITBL75 member. The command is at the line or sequence number 00440013.

If customers do not want the CNM493I message logged, they can use CNMSTYLE or the DEFAULTS command to disable logging of the CNM493I message. Use CNM493I when you initially develop and test your automation table. Turn off CNM493I when you put your table into production.

You can disable the logging of CNM493I to reduce the amount of messages written to the NetView log. Be careful about disabling. Doing so reduces the information that is available for problem determination, and it can cause confusion.

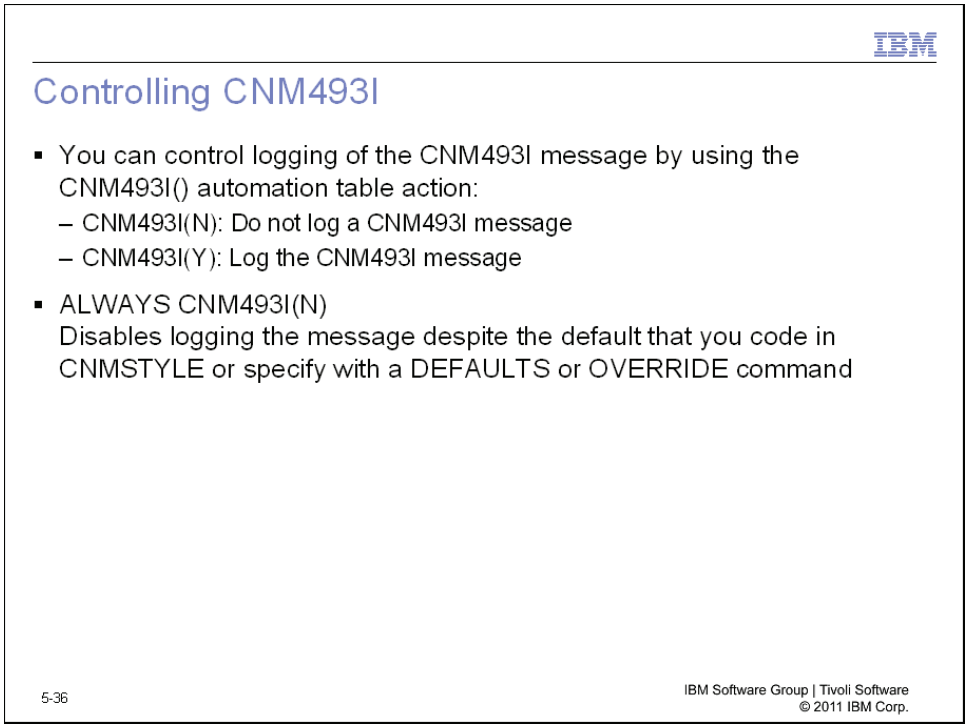
The format of the CNM493I message is as follows:

```
CNM493I member-name : sequence number : command text
```

The command text is shown with variable substitution performed.

Note: If NUM ON has not been issued for the member, the relative line number within the member is shown. In previous releases of NetView, the sequence number would be displayed as N/A if NUM ON had not been set.

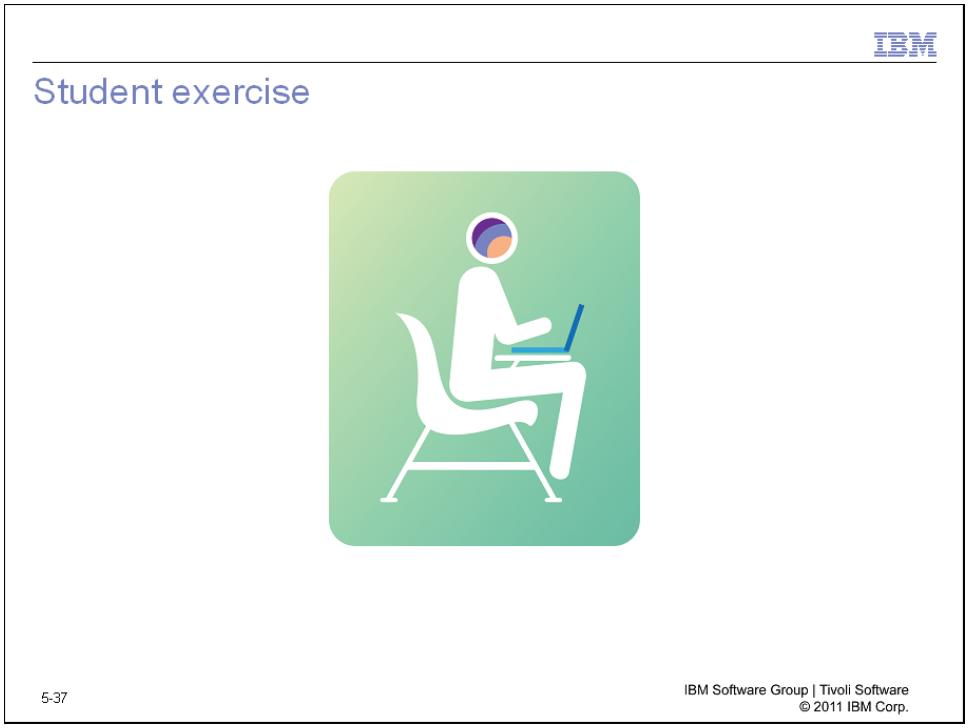
Controlling CNM493I



The CNM493I message action indicates if one of the following CNM493I messages is to be written to the NetView log.

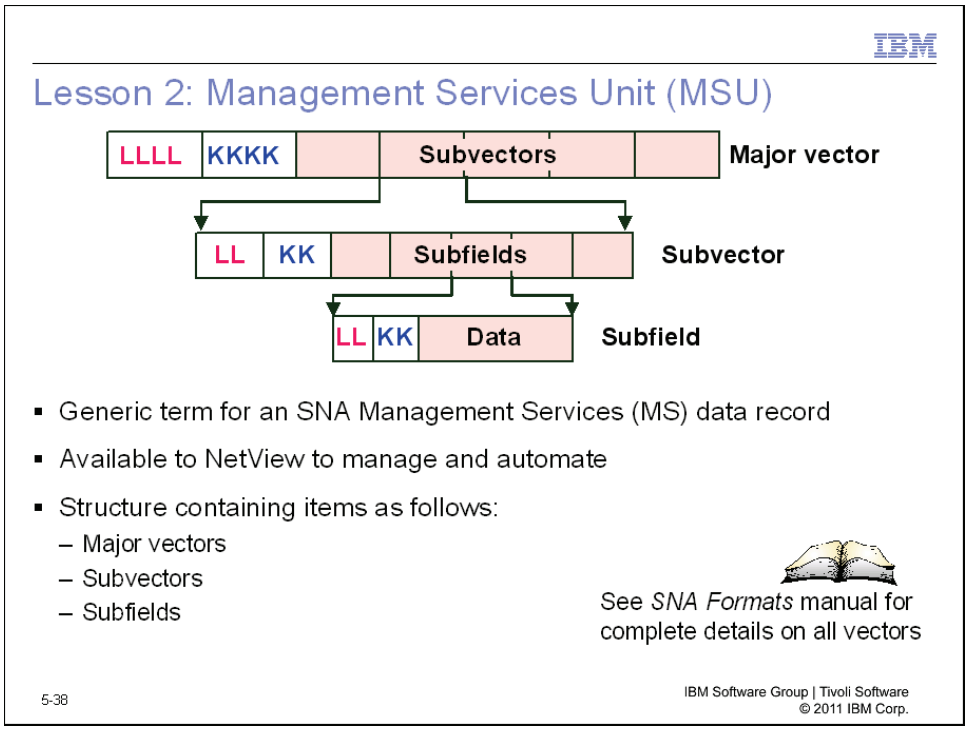
- If a CNM493I action is not specified in the automation table (or by a DEFAULTS or OVERRIDE command), NetView generates CNM493I messages.
- If a CNM493I action is specified, but the appropriate statement has no EXEC action with a CMD keyword, the CNM493I action is ignored.

Student exercise



Open your *Student Exercises* book and perform Exercise 5-1.

Lesson 2: Management Services Unit (MSU)



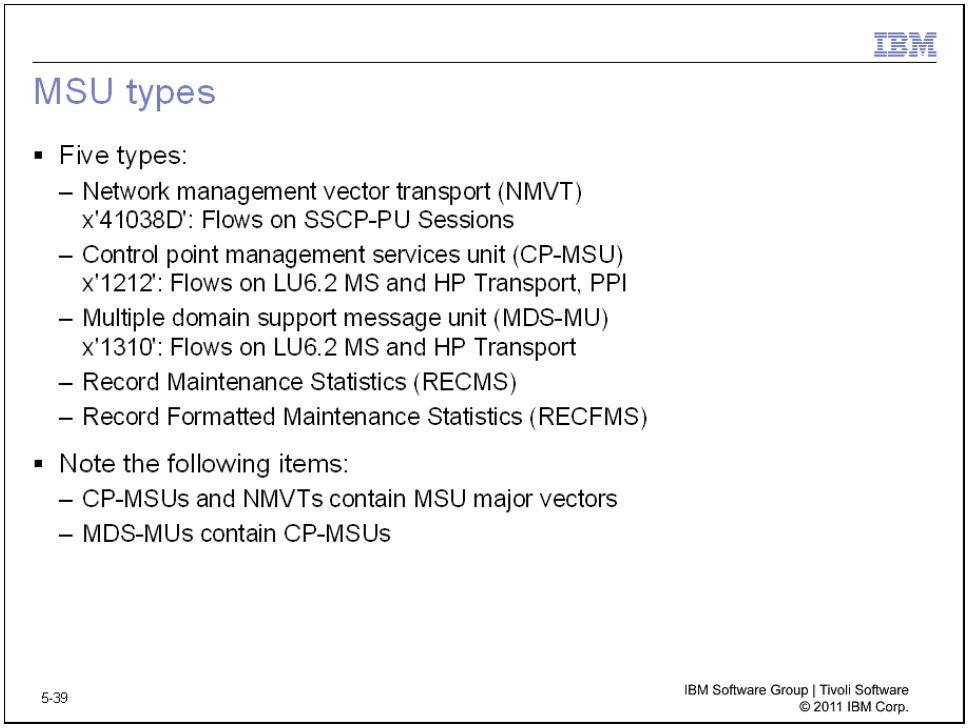
This lesson is optional.

The NetView automation table can process both messages and SNA Management Services Units (MSUs). With SNA structure, management information can exchange between Control Points (for example, CP-to-CP, SSCP-to-PU). Such data flows by using generic Management Services Units (MSUs).

A Management Services Unit (MSU) is an SNA data record that is used for providing management information to appropriate points of the network. One point is the NetView program. With such information, the SNA network can be managed at the SNA level (activating control units). NetView uses the data for managing more complexity and automating as necessary.

An MSU is a structured data unit that contains one or more major vectors, which, in turn, contain one or more subvectors. Subfields within a subvector contain the actual data. The size and content of an MSU depends on the Control Point that formats the data.

MSU types



Five types of MSUs, which are envelopes for transporting the actual data, can be automated by NetView as follows:

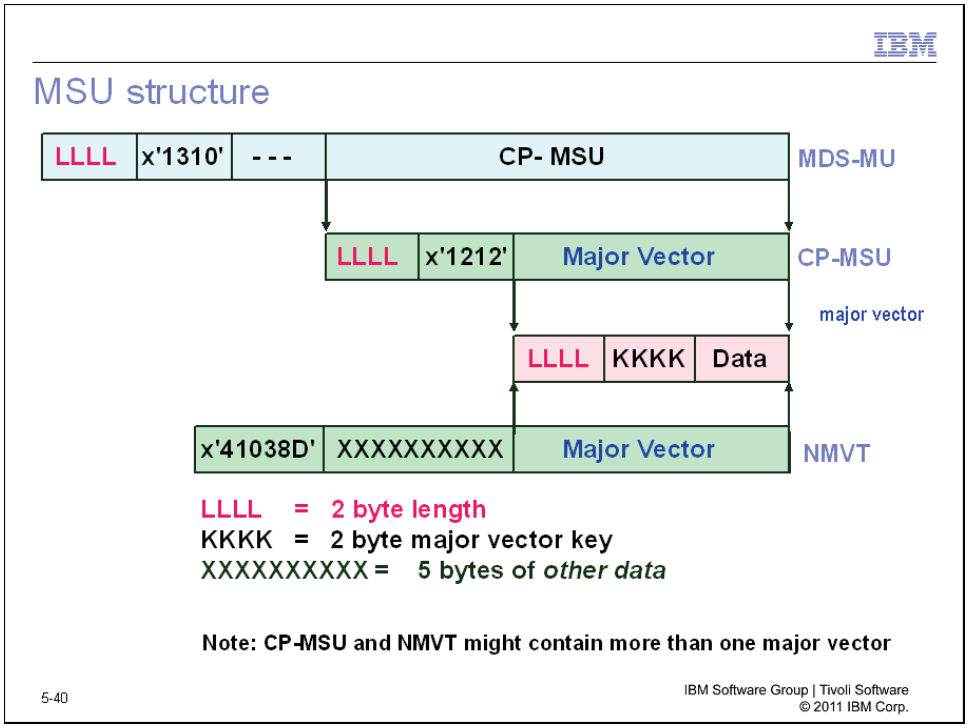
- Network management vector transport (NMVT): Flow on SSSCP-PU sessions
- Control point MSU (CP-MSU): Flow on LU 6.2 MS and HP transport and the PPI
- Multiple-domain support message unit (MDS-MU): Flow on LU 6.2 MS and HP transport between domains
- RECMS
- RECFMS

For automation testing, you can generate alerts with GENALERT.

For online information, enter the following text:

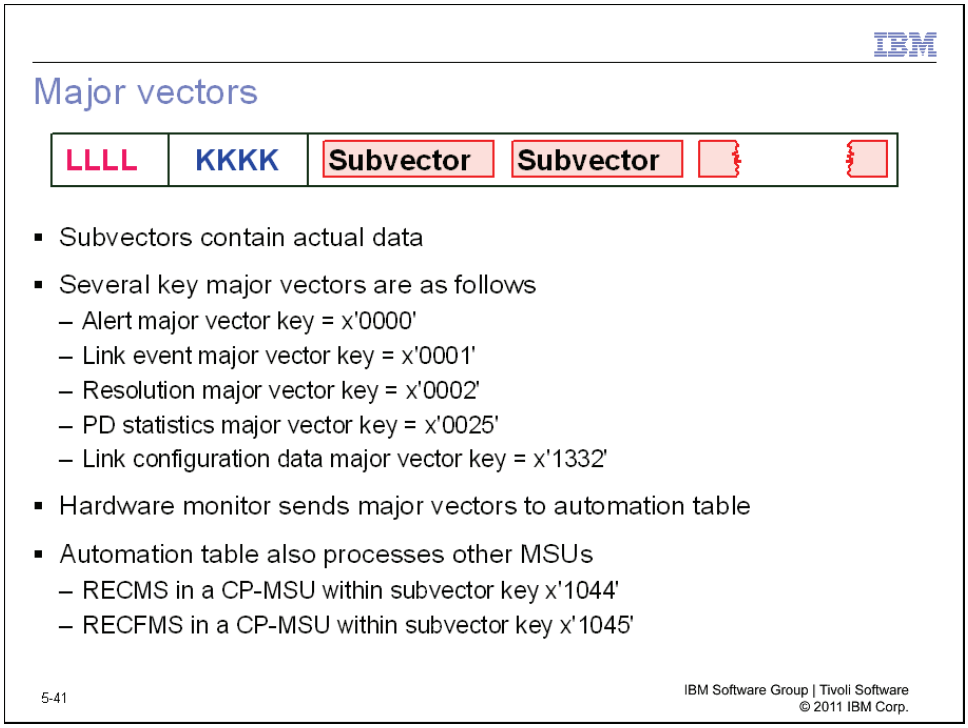
```
HELP GENALERT GENERIC FORMAT
HELP GENALERT NONGENERIC FORMAT
HELP GENALERT RECFMS FORMAT
HELP GENALERT RESOLVED FORMAT
```

MSU structure



An MDS-MU is an envelope that carries CP-MSUs. The CP-MSUs and NMVTs are envelopes that transport data such as major vectors. Detailed descriptions of MSUs are in the *SNA Formats* and the *SNA Management Services Reference* manuals.

Major vectors



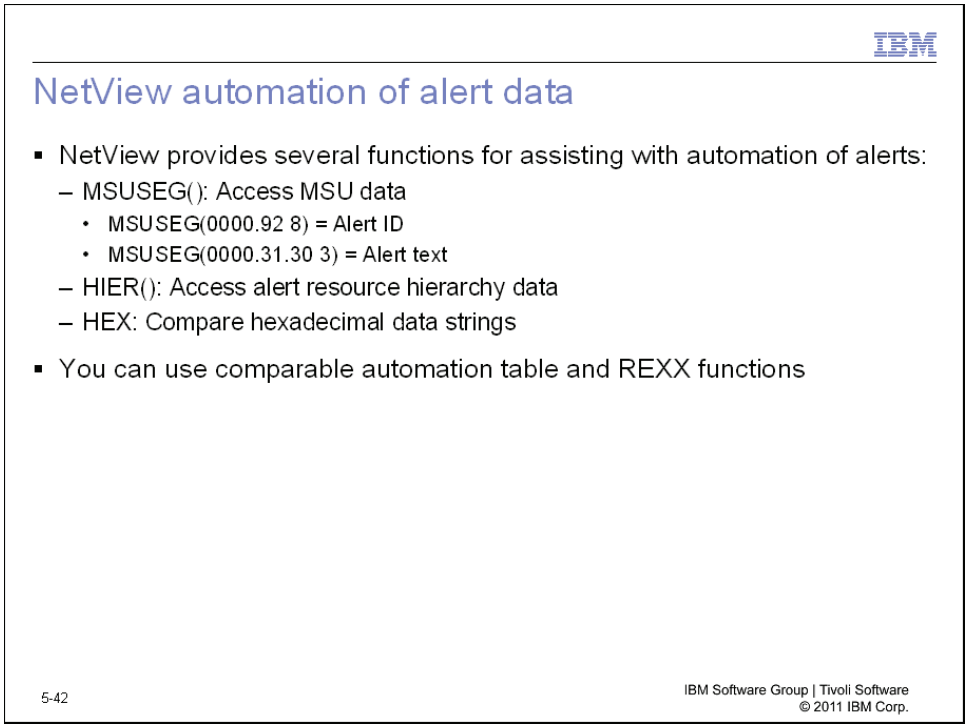
Certain types of MSUs that Hardware Monitor (NPDA) forwards to the automation table are as follows:

- x'0000' Alerts
- x'0001' Link events
- x'0002' Resolution vectors
- x'0025' - PD statistics
- x'1332' Link configuration data

The automation table has access to any MDS-MU or CP-MSU. The original SNA MSU record types are also available to the automation table if they are encapsulated in one of the following subvectors within a CP-MSU:

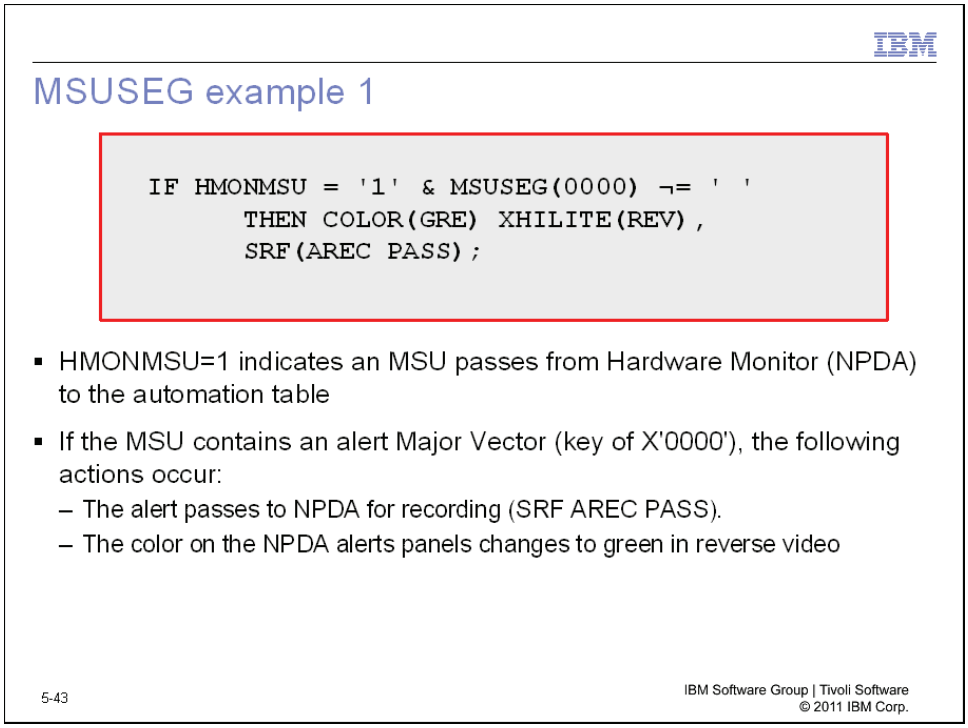
- Subvector with key X'1044' for RECMS records
- Subvector with key X'1045' for RECFMS records

NetView automation of alert data



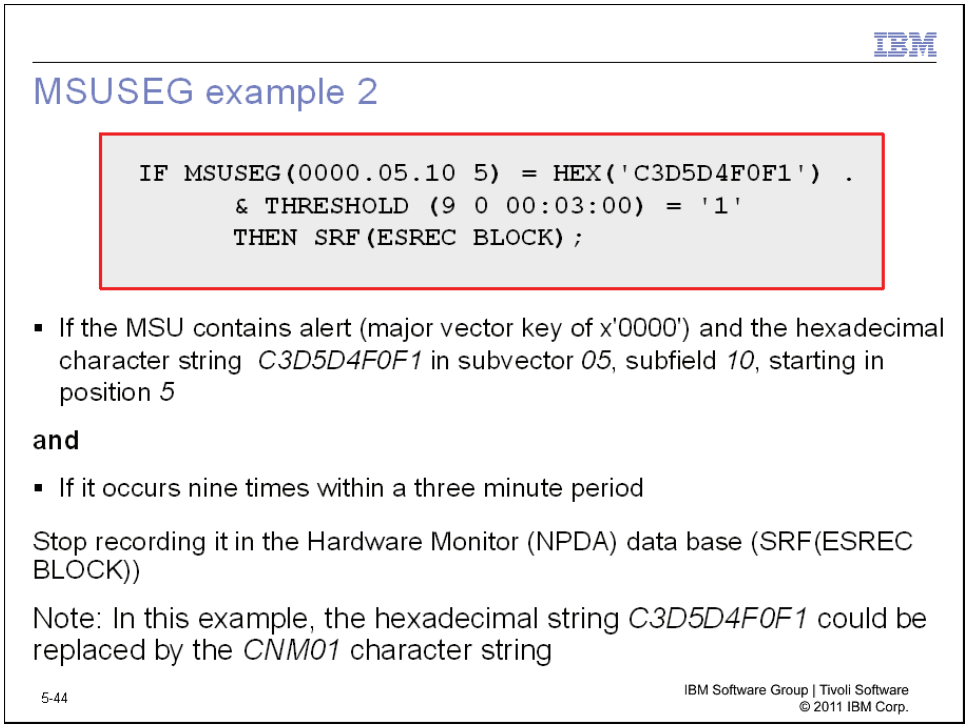
With the NetView automation table and REXX, you can index into an MSU to test data in its subvectors and subfields.

MSUSEG example 1



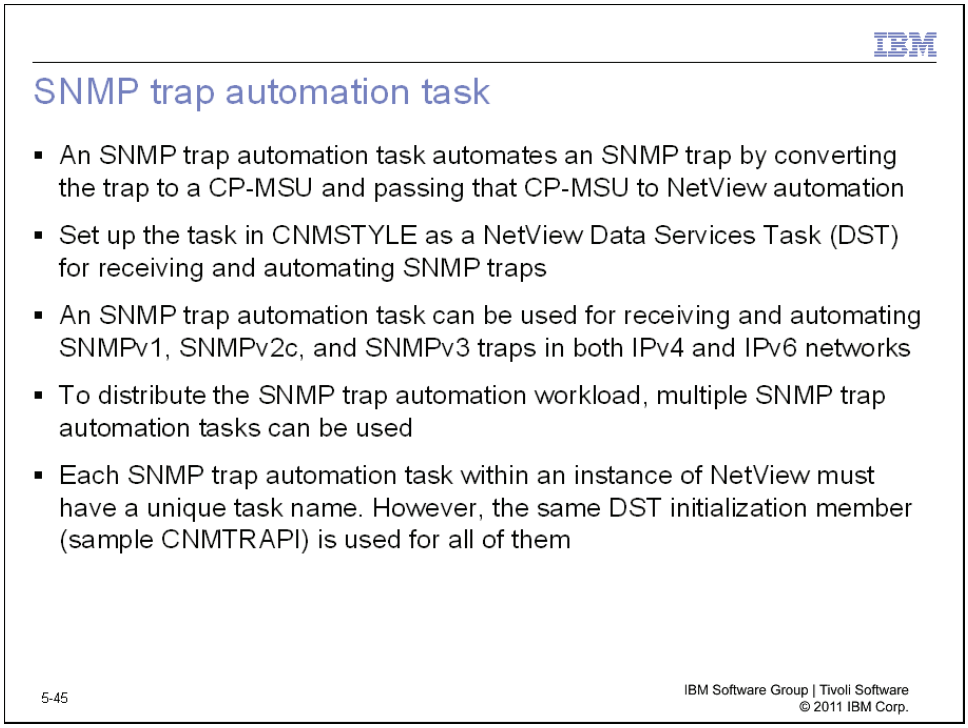
This example shows a test for a Management Services Unit (MSU).

MSUSEG example 2



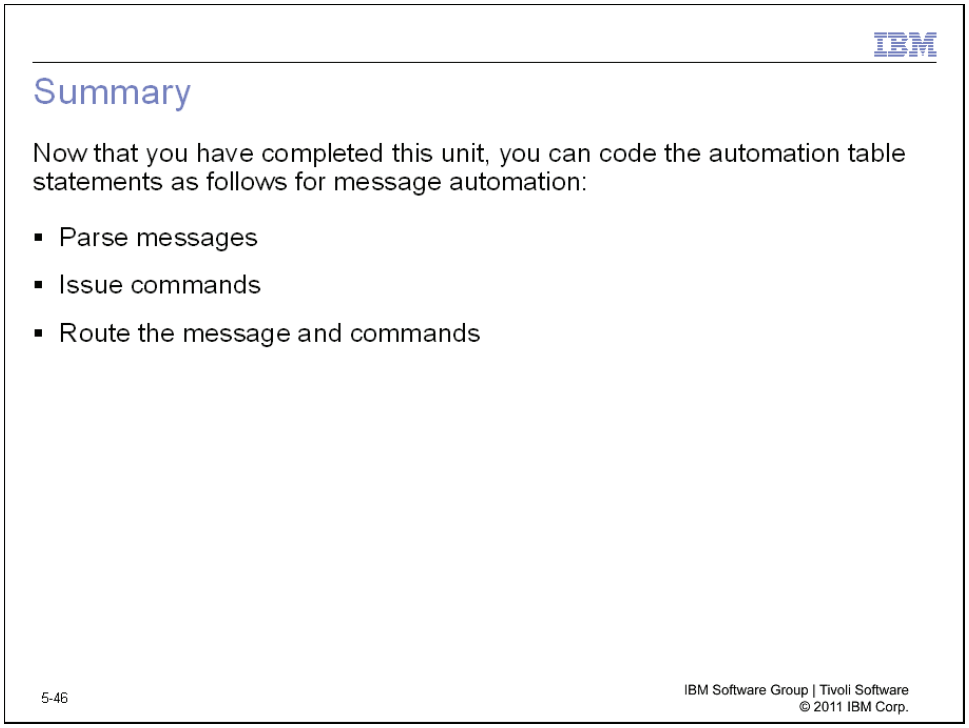
This example shows another test for a Management Services Unit (MSU). Use the period as a placeholder after the hex string to set the limit of the hex string comparison. Days are numeric (zero to 365) and are represented by the value after the threshold count (9 in this case).

SNMP trap automation task



For more information, see the *IBM Tivoli NetView for z/OS Automation Guide*, Chapter 22, "Automating Messages and Management Services Units," and Chapter 37, "SNMP Trap Automation."

Summary



IBM Tivoli certification and training

In today's global business world, enhancing and maintaining skills is essential to keeping pace with rapidly changing technologies. Businesses need to maximize technology potential and employees need to keep up to date with the latest information. Training and professional certification are two powerful solutions.

Certification

There are many reasons for certification:

- You demonstrate value to your customer through increased overall performance with shorter time cycles to deliver applications.
- Technical certifications assist technical professionals to obtain more visibility to potential customers.
- You differentiate your skills and knowledge from other professionals and stand out as the committed technical professional in today's competitive global world.

Online certification paths are available to guide you through the process for achieving certification in many IBM Tivoli areas. See **ibm.com/tivoli/education** for more information.

Special offer for having taken this course

Now through 31 December 2011: For having completed this course, you are entitled to a 15% discount on your next examination at any Thomson Prometric testing center worldwide. Use this special promotion code when registering online or by telephone to receive the discount: **15CSWR**. (This offer might be withdrawn. Check with the testing center as described later in this section.)

Role-based certification

All IBM certifications are based on job roles. They focus on a job a person must do with a product, not just the product's features and functions. Tivoli Professional Certification uses the following job roles used:

- IBM Certified Advanced Deployment Professional
- IBM Certified Deployment Professional
- IBM Certified Administrator
- IBM Certified Solution Advisor
- IBM Certified Specialist
- IBM Certified Operator

Training

A broad spectrum of courses, delivery options, and tools helps keep your employees up to date with the latest IBM Tivoli information:

- *Instructor-led training (ILT)*
  Live interaction with an IBM instructor, hands-on lab exercises, and networking with your peers from other companies
  ibm.com/tivoli/education
- *Instructor-led online (ILO)*
  All the benefits of ILT, but savings on travel dollars and training costs
  ibm.com/training/ilo
- *Self-paced virtual classes (SPVC)*
  Interactive and hands-on exercises on your schedule
  ibm.com/training/us/spvc
- *Web-based training (WBT)*
  Training anywhere, any time, that saves you money and travel
  ibm.com/training/us/tivoli/wbt
- *Multimedia library*
  Modules supporting new and experienced learners with fully animated multimedia clips, step-by-step audio, and companion text
  ibm.com/software/tivoli/education/multimedialibrary
- *IBM Education Assistant*
  More specific, granular web-based training with individual presentations on specific topics
  www-01.ibm.com/software/info/education/assistant/
- *Corporate Education Licensing Program (CELP)*
  Solutions for large IBM customers who need to adopt IBM Tivoli's tools and technologies
  ibm.com/training/us/tivoli/celp
- *Tivoli training paths*
  Course maps with flow charts and course descriptions to help you find the right course
  ibm.com/training/us/tivoli/paths

Printed in Ireland

CT6E9EN