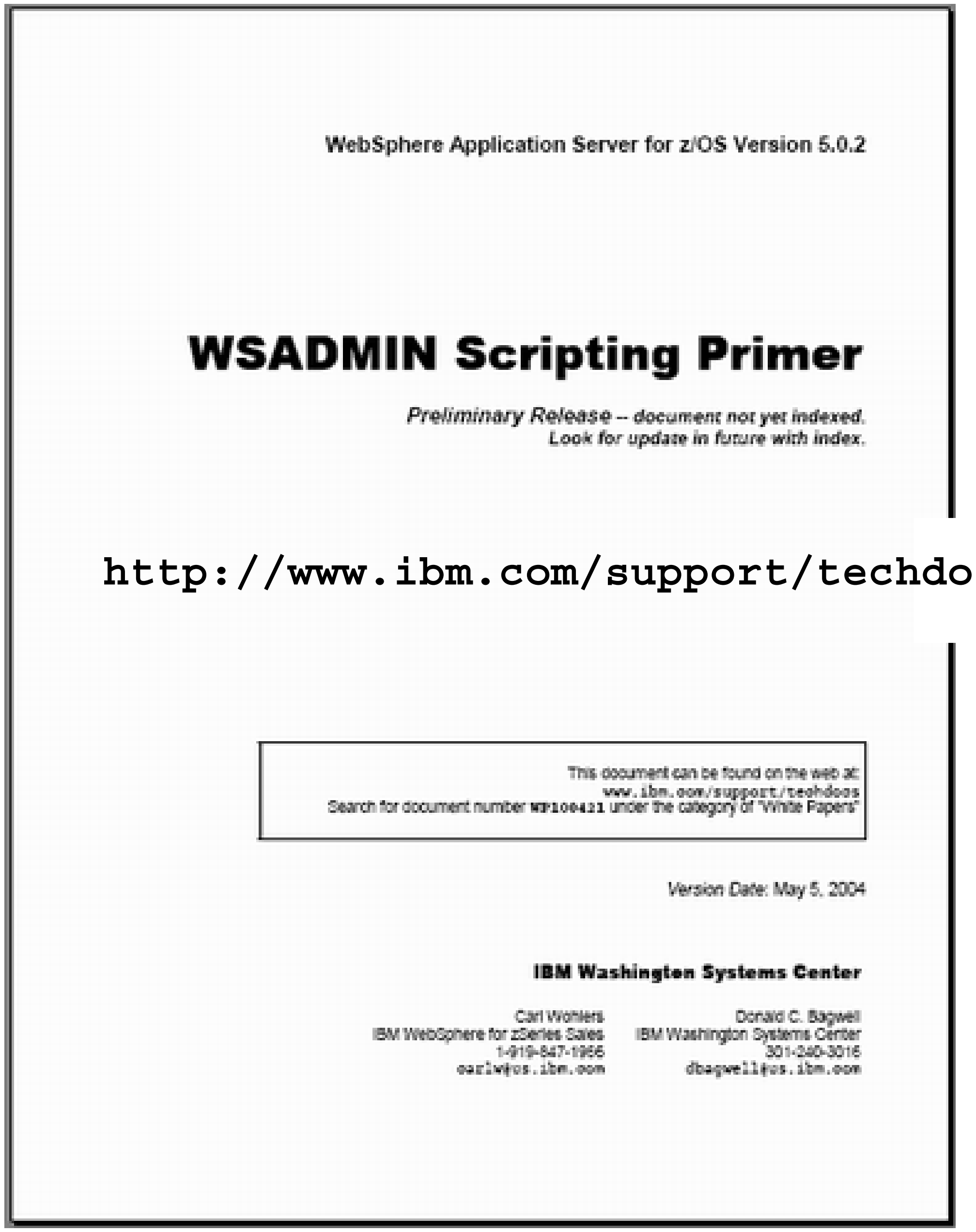
IBM WebSphere
Software

WebSphere Application Server
for z/OS and OS/390

WSADMIN
Scripting Interface

IBM Americas Advanced Technical Support -- Washington Systems Center
Gaithersburg, MD, USA

Presentation Based on White Paper



WebSphere Application Server for z/OS Version 5.0.2

WSADMIN Scripting Primer

Preliminary Release -- *document not yet indexed. Look for update in future with index.*

This document can be found on the web at: www.ibm.com/support/techdocs
Search for document number WP100421 under the category of "White Papers"

Version Date: May 5, 2004

IBM Washington Systems Center

Carl Wohlers
IBM WebSphere for zSeries Sales
1-919-847-1966
carlw@us.ibm.com

Donald C. Bagwell
IBM Washington Systems Center
301-240-3016
dbagwell@us.ibm.com

If you're interested in going deeper still, refer to white paper WP100421 on the "Techdocs" website

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100421`

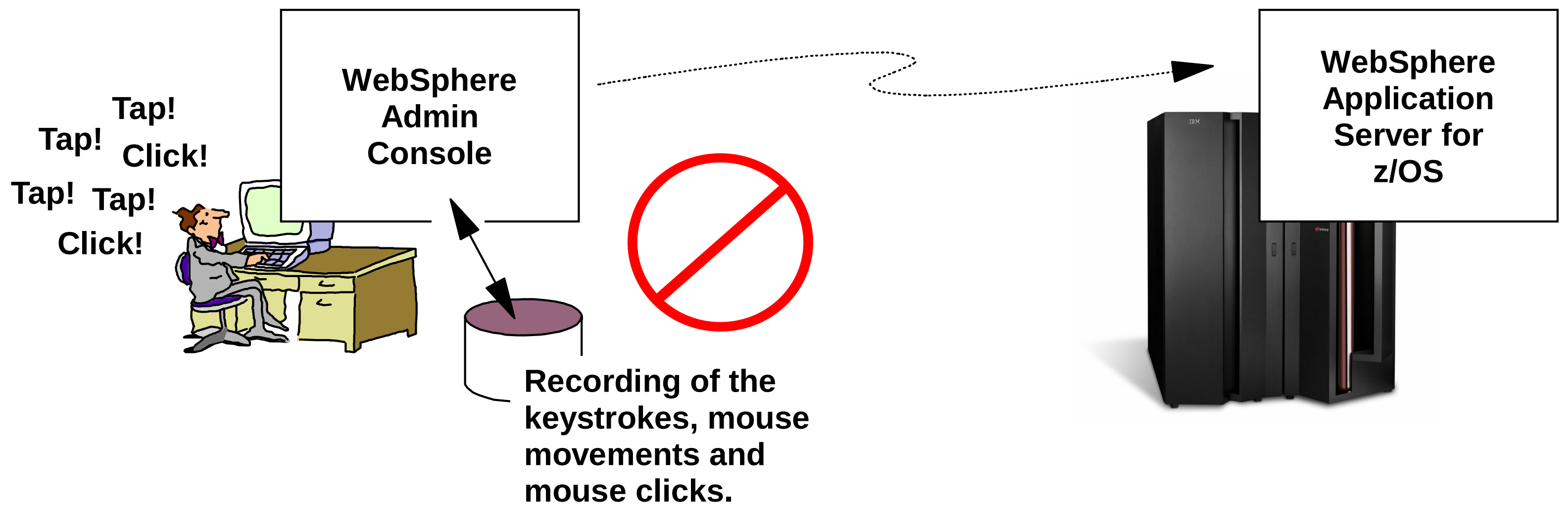
Includes a ZIP file with dozens of exercises.

This presentation won't go into nearly as much detail; rather it'll pick up the key points.

If you want more detail, pull the white paper off Techdocs.

What WSADMIN is NOT

WSADMIN is not a keyboard activity record-and-playback mechanism.


Tap!
Tap!
Click!
Tap!
Tap!
Click!
WebSphere Admin Console
Recording of the keystrokes, mouse movements and mouse clicks.
WebSphere Application Server for z/OS

A common question is whether it's possible to record Admin Console work and use it to create a WSADMIN script.

The answer is "no" ... but that's not necessarily a bad thing. As you'll see, many WSADMIN commands are far simpler than the steps you'd take in the Admin Console to achieve the same thing.

Some, however, are more complex. It's a tradeoff.

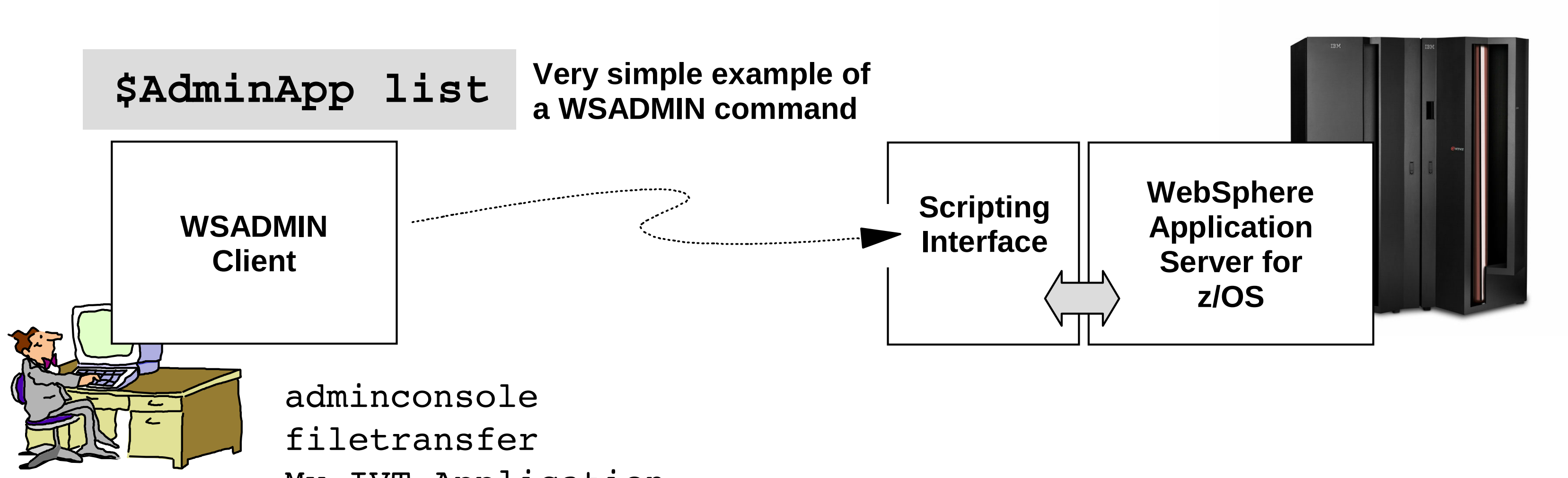
What is WSADMIN?

WSADMIN is Scripting Interface

WSADMIN is an interface to WebSphere that allows commands issued to modify some aspect of the runtime environment:

```
$AdminApp list
```

Very simple example of a WSADMIN command



```
adminconsole
filetransfer
My_IVT_Application
SuperSnoop
  :
  :
```

What sort of things can be accomplished?

- Install or uninstall applications
- Modify an existing application
- Start or stop servers
- Initiate node synchronization
- Create new servers, clusters, virtual hosts, etc.

Without realizing it, you may have already used WSADMIN ...

You've Probably Used WSADMIN

CNTL

BBOWIAPP

Submitted before you started the server ... don't need server running to install applications

```
//INST1 EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 BPXBATCH SH +
   /wasv5config/g5cell+
   /AppServer+
   /bin/wsadmin.sh -conntype none +
   -c '$AdminApp install +
   /wasv5config/g5cell+
   /AppServer+
   /installableApps/adminconsole.ear +
  {-appname adminconsole  +
   -MapRolesToUsers {{"administrator" ...
   {"monitor" No No G5ADMIN G5CFG} +
   {"operator" No No G5ADMIN G5CFG} +
   {"configurator" No No G5ADMIN G5CFG}} +
   -server g5sr01c +
   -node g5nodec +
   -cell g5cellc +
   -copy.sessionmgr.
   g5sr01c}' +
   1> /tmp/bbowiapp_26921.out +
   2> /tmp/bbowiapp_26921.err
/*
```

BPXBATCH invocation of shell script

WSADMIN command, attributes and options

When configuring WebSphere initially, the BBOWIAPP job installed the Admin Console into your new server using WSADMIN

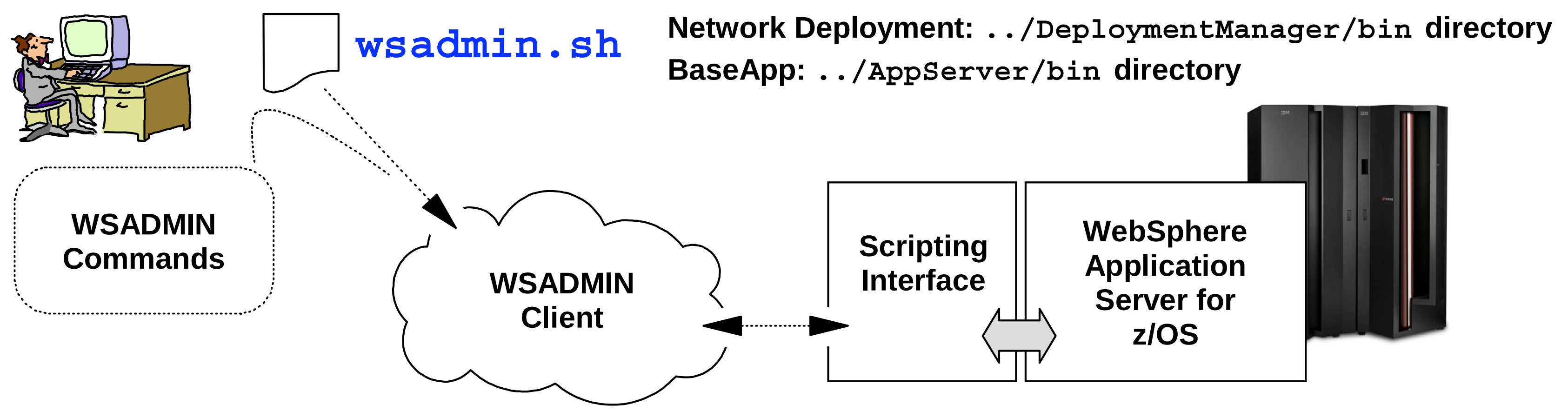
Some interesting things:

- **Server wasn't up when you installed application**
- **Simple BPXBATCH invocation of `wsadmin.sh` shell script**
- **WSADMIN command and its attributes/options contained in the JCL**

We'll explore all of these things in this presentation.

WSADMIN "Client"

To exercise the scripting interface you use the WSADMIN client. On z/OS the client comes in the form of a shell script:



Network Deployment: `../DeploymentManager/bin` directory
BaseApp: `../AppServer/bin` directory

This is the bare-bones basics of it. There are a lot of variations on how this is done, which we'll cover. For now, understand three key points:

- WSADMIN client is shell script
- WSADMIN commands passed into client
- Client operates against WebSphere

Where can you run client?

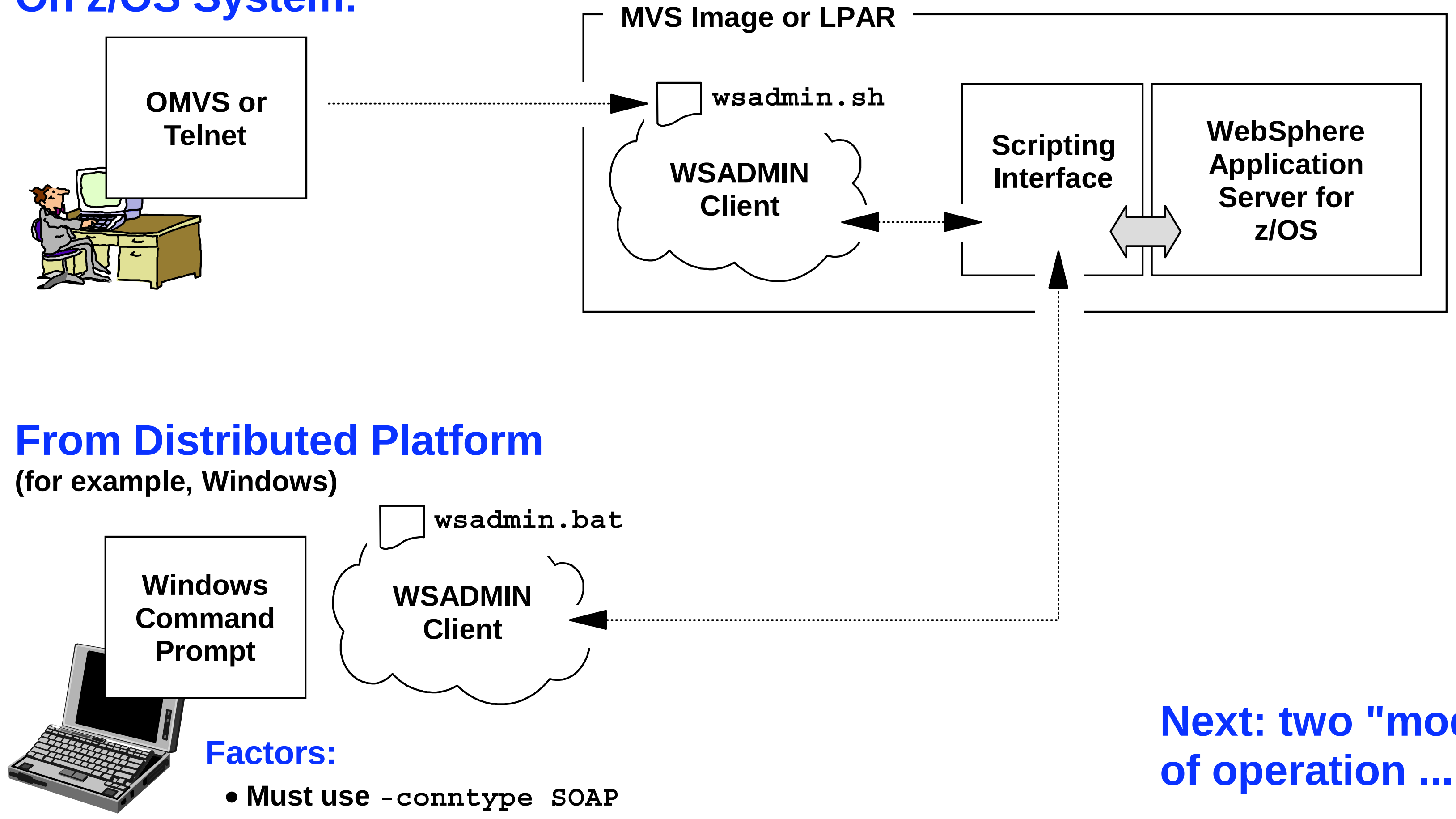
Where You May Run Client

WSADMIN is provided on all WebSphere Application Server platforms. So it's possible to run the WSADMIN client in different places:

On z/OS System:

From Distributed Platform

(for example, Windows)



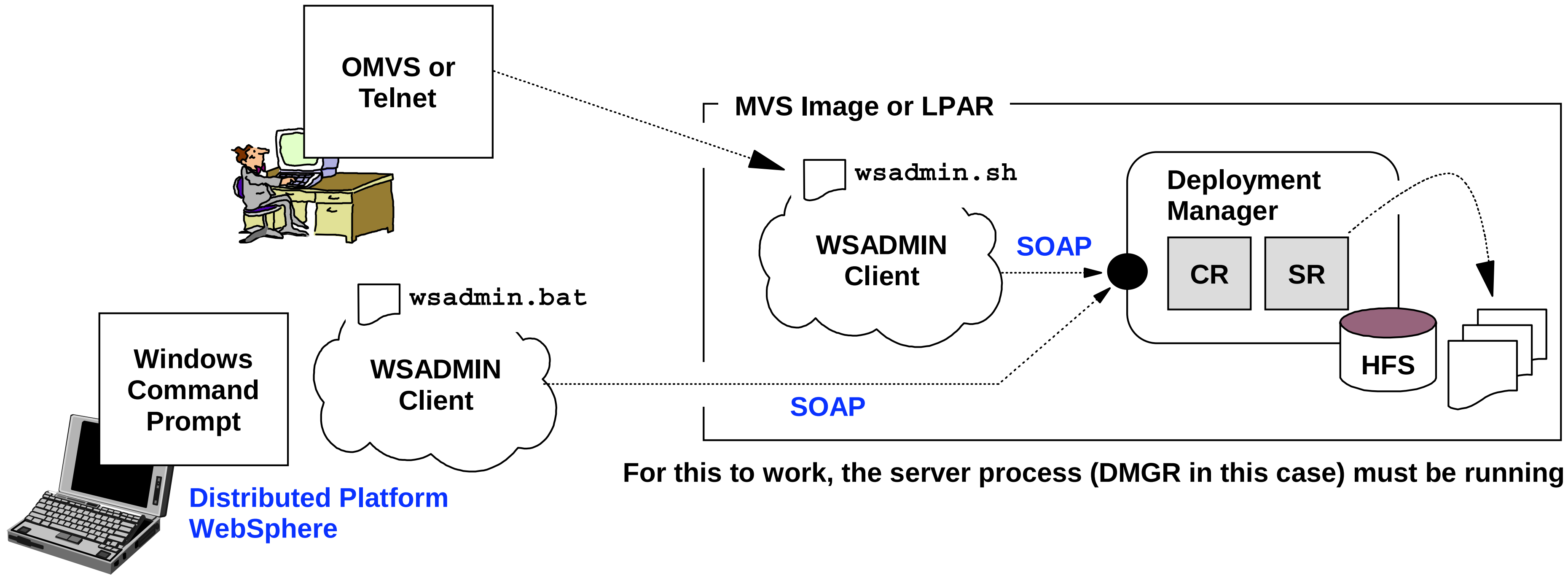
Factors:

- Must use `-conntype SOAP`
- Target server process must be up
- When security on then need to coordinate certificates

Next: two "modes" of operation ...

"Local" Mode vs. "Remote" Mode

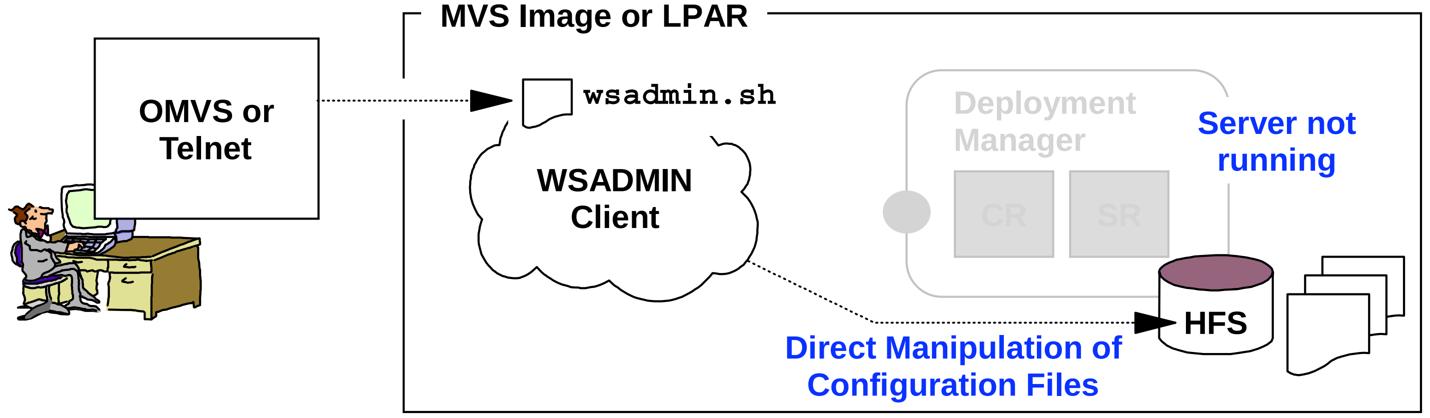
"Remote" Mode -- Connect via SOAP to server; let server modify configuration files



For this to work, the server process (DMGR in this case) must be running

"Local" Mode -- WSADMIN changes configuration files directly

- WSADMIN must run on z/OS
- BBOWIAPP did this
- Some functions not available (`$AdminControl`)



When "Local" vs. "Remote"

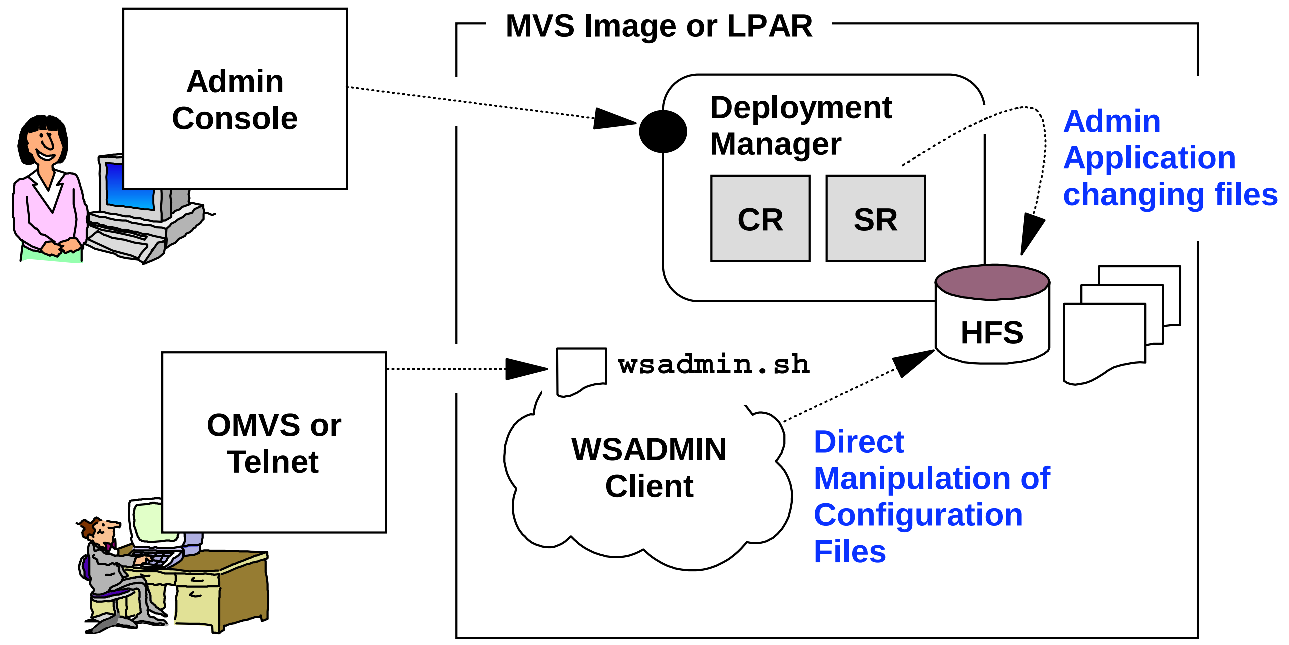
Rule of Thumb: If Deployment Manager (or AppServer if BaseApp) is available, connect to it ("Remote"). If server process not available, then use "Local."

Avoid this

Why? Admin Console will detect change in underlying repository. Changes you made must then either:

- Be discarded
- Overwrite WSADMIN changes

It can be very confusing.



If you come in "remote," the server running the administrative service can handle (to some degree) two different forces working against the configuration repository. But it has to know about WSADMIN doing it, and it can't if WSADMIN is operating in "local" mode.

"...to some degree..." -- Some configuration buffering does occur. Based on timing, it's possible changes in one environment won't be "seen" in the other.

Generally speaking, even in remote mode you should avoid having the Admin Console working against repository at the same time WSADMIN is doing it.

On z/OS, Run Under "WAS Admin ID"

In order to have access to the configuration directory structure, `wsadmin.sh` must run under the authority of the "WebSphere Administrator ID"

If Telnet or OMVS:

```
EZYTE27I login: USER1
EZYTE28I user1 Password: xxxxxxxx
 :
 :
USER1:/u/user1-> su g5admin
Enter the password for g5admin: xxxxxxxx
USER1:/u/user1-> cd /wasv5config/g5cell/DeploymentManager/bin
USER1:/wasv5config/g5cell/DeploymentManager/bin-> ./wsadmin.sh ...
```

Switch users to the WebSphere Admin ID

If JCL

```
**************************************
//WSADMIN   JOB (ACCTNO,ROOM),'USER1',
//   USER=G5ADMIN,PASSWORD=xxxxxxxx
//************************************
//INST1 EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 BPXBATCH SH +
   /wasv5config/g5cell+
   /DeploymentManager/bin/wsadmin.sh ...
```

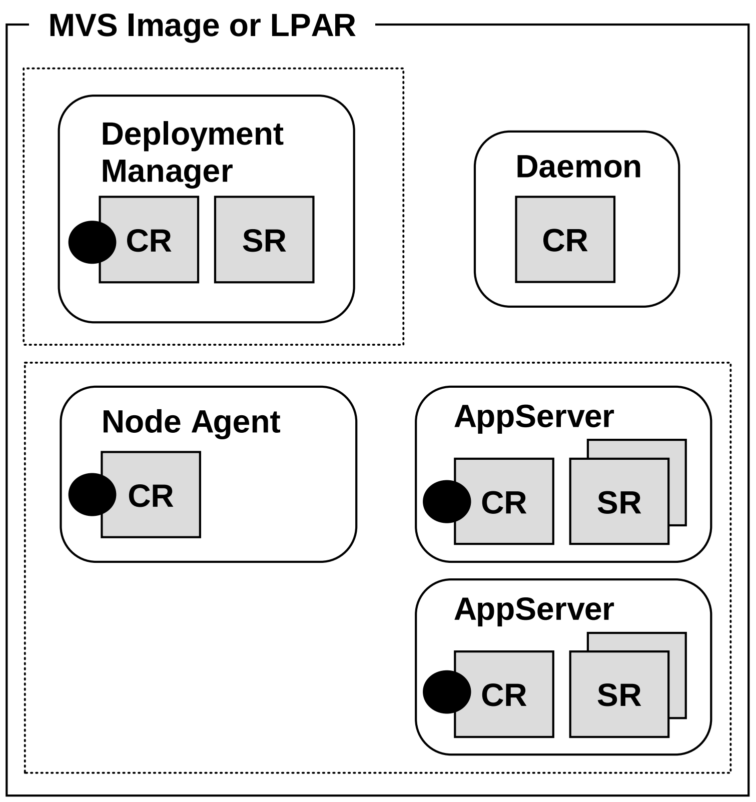
Provide authority via JOB card

Any UID=0 ID? It'll work, but it may affect file ownership. Better to use WAS Admin ID.

This is different from the issue of authentication when "Global Security" enabled. More on that at end of presentation.

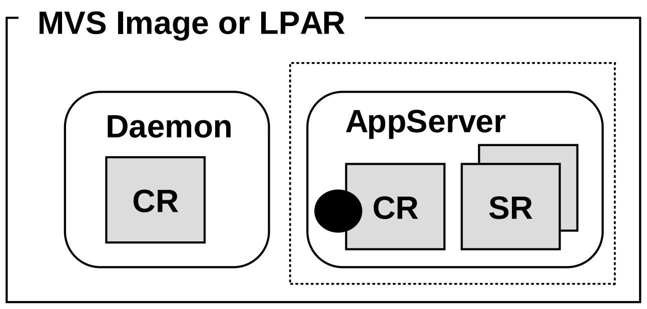
Network Deployment vs. BaseApp

In a Network Deployment configuration, there are many different SOAP ports to which WSADMIN could connect:



Rule of Thumb: Connect to Deployment Manager. That'll then update "master configuration"

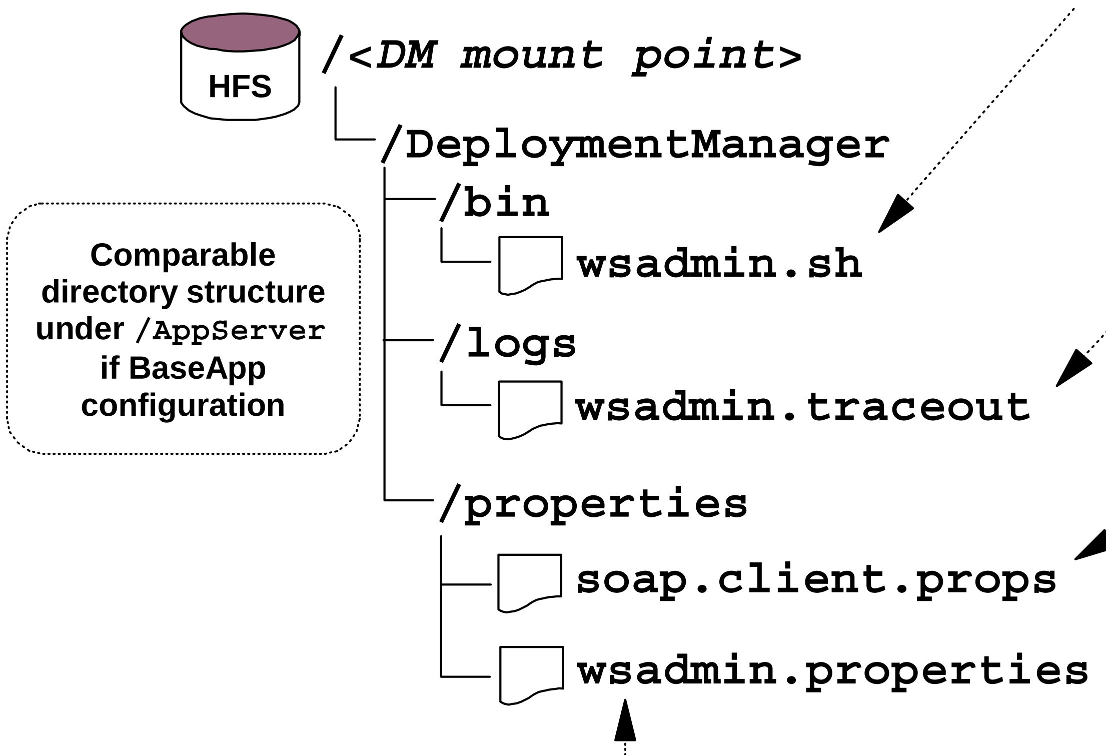
A Base Application Server node has only the application server, so that's what you'd connect to in "Remote" mode:



Message:

- If ND, connect to DMGR
- When it comes to basics of WSADMIN, Network Deployment or BaseApp are essentially the same
- Going forward in this presentation we'll assume ND

Other Files To Be Aware Of



WSADMIN shell script

- This is what you invoke either from OMVS/Telnet or within JCL

Where WSADMIN client-side tracing will go by default

- Location specified in `wsadmin.properties` and you may change it if you wish

Update needed if security enabled:

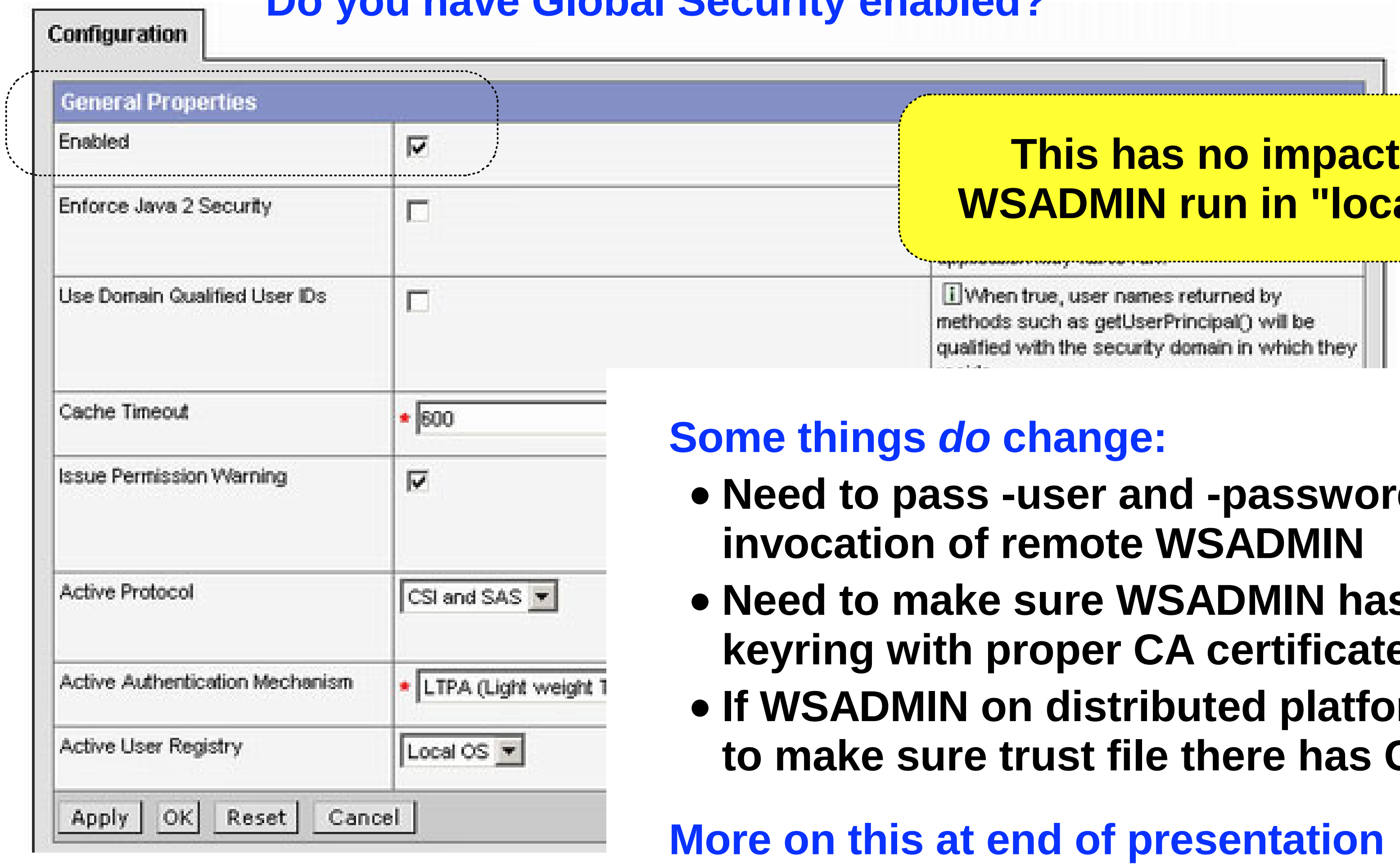
- Need to set `com.ibm.SOAP.securityEnabled` property from "false" to "true"
- May need to change pointer to keyring where CA certificate stored

Key things in here:

- Tracing on/off switch
- "temp" file location
- Default script type (Jacl vs. Jython)
- tracing file output directory and file

What About Security?

Do you have Global Security enabled?



Enabled	☑
Enforce Java 2 Security	☐
Use Domain Qualified User IDs	☐
Cache Timeout	* 600
Issue Permission Warning	☑
Active Protocol	CSI and SAS
Active Authentication Mechanism	* LTPA (Light weight T
Active User Registry	Local OS

When true, user names returned by methods such as getUserPrincipal() will be qualified with the security domain in which they

This has no impact when WSADMIN run in "local" mode

Some things *do* change:

- **Need to pass -user and -password in on invocation of remote WSADMIN**
- **Need to make sure WSADMIN has access to keyring with proper CA certificate**
- **If WSADMIN on distributed platform, you'll need to make sure trust file there has CA certificate**

More on this at end of presentation

Key Message: scripting *itself* is not affected when security enabled -- only access to scripting interface

Syntax of WSADMIN Invocation

```
./wsadmin.sh -?
```

```
wsadmin
        [ -h(elp)  ]
        [ -?  ]
        [ -c <command> ]
        [ -p <properties_file_name>]
        [ -profile <profile_script_name>]
        [ -f <script_file_name>]
        [ -javaoption java_option]
        [ -lang  language]
        [ -wsadmin_classpath  classpath]
        [ -conntype
                SOAP
                        [-host host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password] |
                RMI
                        [-host host_name]
                        [-port port_number]
                        [-user userid]
                        [-password password]  |
                JMS <jms parms> |
                NONE
        ]
        [ script parameters ]
```

Used to indicate WSADMIN commands follow. We illustrate that in a few charts.

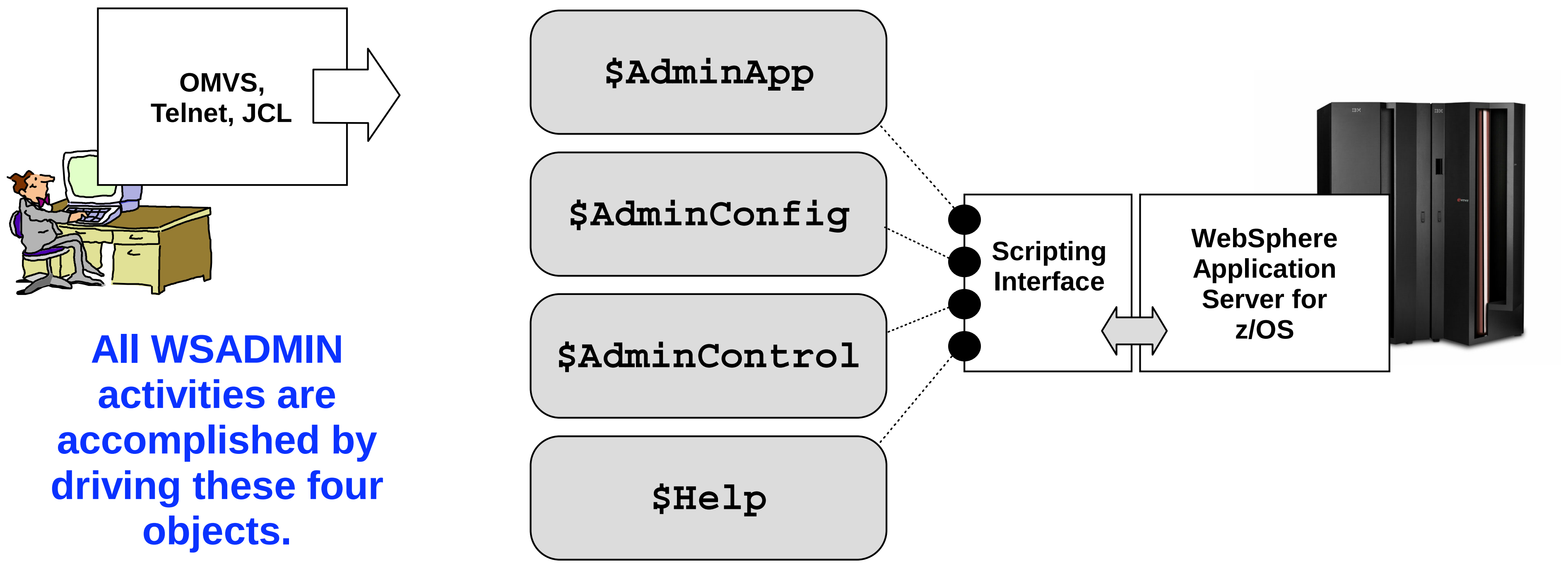
Used to tell WSADMIN that script file is in EBCDIC rather than default ASCII

Used to point to a file in which the commands are held. We illustrate that after `-c` switch.

Note: if connecting via SOAP <u>and</u> global security is enabled, provide the WebSphere Admin ID and password on the invocation.

Used to indicate the type of connection -- "Remote" (`-conntype SOAP`) or "Local" (`-conntype NONE`)

Four WSADMIN Program "Objects"



All WSADMIN activities are accomplished by driving these four objects.

Each has many different "methods," attributes and options:

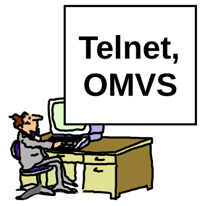
`$AdminApp`	`uninstall`	`My_IVT_Application`	
Object	Method	Attribute	Options
`$AdminApp`	`install`	`/u/user1/MyIVT.ear`	`{-server G5SR01C -node G5NODEC}`

Good deal of the learning curve is discovering the syntax of these methods

More examples coming

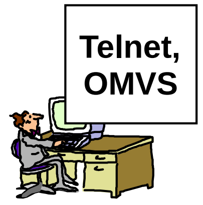
"Inline" Commands

Interactively at WSADMIN prompt



```
./wsadmin.sh
   :
WASX7029I: For help, enter: "$Help help"
wsadmin> $AdminApp list
```

Passed in as parameter on shell script invocation



```
./wsadmin.sh -c '$AdminApp list'
```

Parameter for shell script, but processed in JCL

```
BPXBATCH SH +
   /wasv5config/g5cell+
   /AppServer+
   /bin/wsadmin.sh +
   -c '$AdminApp list +
     :
/*
```

All three of these are more or less the same thing

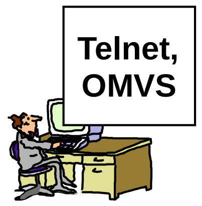
Great for relatively simple things, such as:

- **listing installed applications**
- **uninstalling an application**
- **installing an application with a small set of options**
- **Exploring the "help" option -- getting information about an option, etc.**

But as the input gets more complex, you want to keep things in a separate file ...

Files Containing Script

It's called a "scripting interface" because scripting languages like "Jacl" and "Jython" can be used to drive the WSADMIN commands:



```
../bin/-> ./wsadmin.sh -conntype none -f /u/user1/install.jacl
```

Fairly simple script ... not a lot of fancy stuff going on here ...

```
install.jacl
set ear      "/u/user1/MyIVT.ear"
set node     "g5nodec"
set server   "g5sr01c"
# ---------------------------------------
set options [list -node $node -server $server]
# ---------------------------------------
$AdminApp install $ear $options
$AdminConfig save
```

Script processing allows:

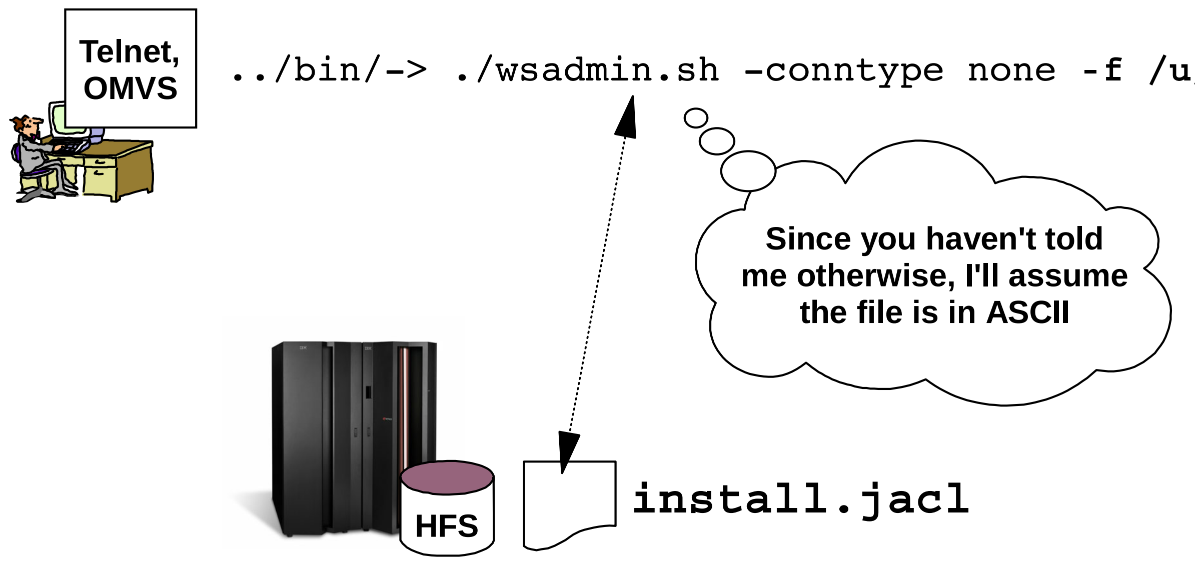
- passing in parameters
- logic tests (if-then-else)
- built-in functions (count, length, string, etc.)
- error checking and handling

Notes:

- "Jacl" is Java-based version of "Tcl" scripting language
- "Jacl" is default script-type expected
- Support for "Jython" in WebSphere for z/OS Version 5.1

WSADMIN Expects ASCII Script File

Be aware that WSADMIN client on z/OS expects -- by <u>default</u> -- for script files to be in ASCII encoding:


Telnet, OMVS
Since you haven't told me otherwise, I'll assume the file is in ASCII
HFS
install.jacl

```
../bin/-> ./wsadmin.sh -conntype none -f /u/user1/install.jacl
```

You're free to store the files in ASCII in the HFS

It works just fine when you do

If file is really in EBCDIC and WSADMIN expects ASCII, it'll fail. But there is a way to tell WSADMIN that the file is in EBCDIC:

```
./wsadmin.sh -javaoption -Dscript.encoding=Cp1047 -conntype none -f /u/user1/install.jacl
```

The `-javaoption` switch is used to pass in the type of encoding used by the script file

The $AdminApp Object

```
$AdminApp help

edit
editInteractive
export
exportDDL
help
install
installInteractive
list
listModules
options
publishWSDL
taskInfo
uninstall
updateAccessIDs
deleteUserAndGroupEntries
```

```
$AdminApp help install

WASX7096I: Method: install

        Arguments: filename, options

        Description: Installs the application in the file
        specified by "filename" using the options specified
        by "options." All required information must be
        supplied in the options string; no prompting is
        performed.

        The AdminApp "options" command may be used to get a
        list of all possible options for a given ear file.
        The AdminApp "help" command may be used to get more
        information about each particular option.
```

How can you know what options are valid?

Simple Example:

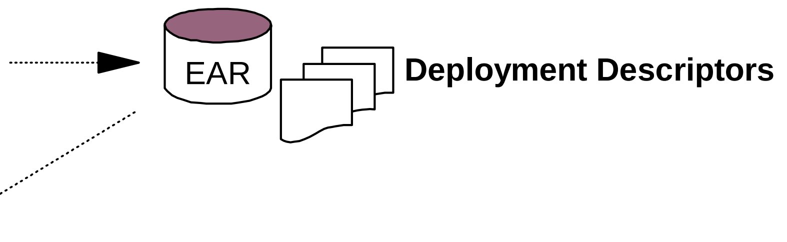
```
$AdminApp install /u/user1/MyIVT.ear {-node g5nodec -server g5sr01c}
```

Object **Method** **Filename** **Options**

$AdminApp Options

The `options` method of `$AdminApp` can be used to list back the tasks (or options) that are valid for a given EAR file:

```
$AdminApp options /u/user1/MyIVT.ear
```



```
WASX7112I: The following tasks are valid for "/u/user1/MyIVT.ear"
BindJndiForEJBNonMessageBinding
MapEJBRefToEJB
MapWebModToVH
MapModulesToServers
  :
server
cluster
cell
node
  :
appname
verbose
contextroot
  :
defaultbinding.force
defaultbinding.strategy.file
```

Two shown on the previous chart

You can use `help` to list back general information on each of these:

```
$AdminApp help appname
```

```
WASX7232I: "appname" option; use this option to specify
the name of the application.  The default is to use the
display name of the application.
```

The InfoCenter is helpful in determining syntax of these options.

Let's look at a simple example and start the discussion on Jacl scripting

Jacl Scripting Basics

Let's show how simple `$AdminApp` command can be coded in Jacl:

```
Object        Method         Filename               Options
$AdminApp install /u/user1/MyIVT.ear {-node g5nodec -server g5sr01c}
```

=

install.jacl

```
set ear      "/u/user1/MyIVT.ear"
set node     "g5nodec"
set server   "g5sr01c"
# ------------------------------------------
set options [list -node $node -server $server]
# ------------------------------------------
$AdminApp install $ear $options
$AdminConfig save
```

Simply setting variables with values for EAR file location and name, node and server

Using Jacl `list` function to create the options, including the enclosing braces.

Variables `node` and `server`, set earlier, included here with substitution -- `$var_name`

More variable substitution, including options list

Use of `$AdminConfig save` to save to the master configuration

Now simply point to this file either on command line or from JCL. Change variables to install different application or install into different server ...

Passing Arguments into Jacl Script

Shell Script

Pointer to Jacl File

Parameters Passed Into Script

```
./wsadmin.sh ... -f /u/user1/install.jacl g5nodec g5sr01c /u/user1/MyIVT.ear
```

Special variable `$argv`

```
g5nodec    g5sr01c    /u/user1/MyIVT.ear
```

Variables same as before

```
set node    [lindex $argv 0]
set server  [lindex $argv 1]
set ear     [lindex $argv 2]
# ----------------------------------------
set options [list -node $node -server $server]
# ----------------------------------------
$AdminApp install $ear $options
$AdminConfig save
```

Jacl function `lindex` parses elements out of a list. It has a zero-offset.

Jacl script is now "generic" and can be used to install any EAR file into any server ... simply by passing in parameters.

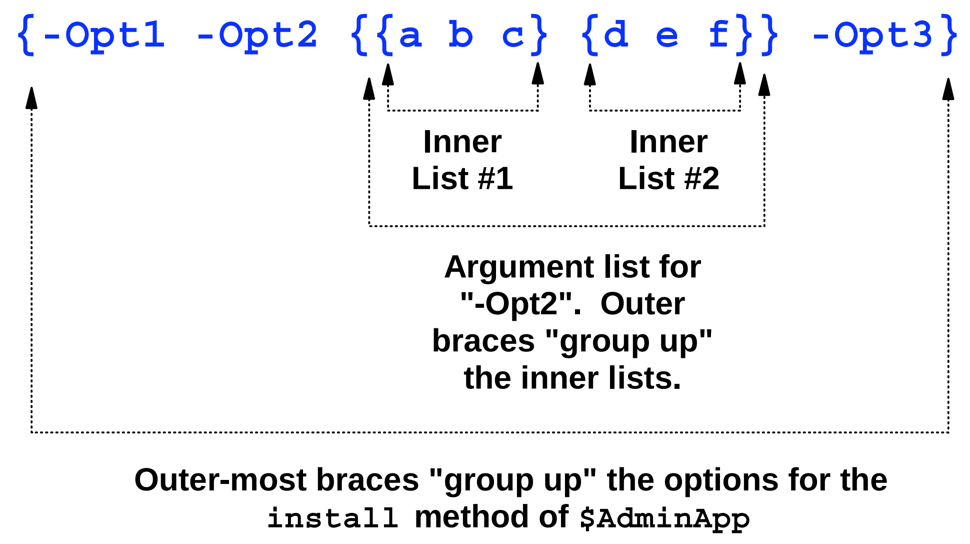
Nested Options

Some options have their own options ... which means it becomes necessary to nest option lists inside of other options:

BBODIAPP
(Installs Admin Console into DMGR)

```
 :
{-appname adminconsole
 -MapRolesToUsers {
   {"administrator" No No G5ADMIN G5CFG}
   {"monitor" No No G5ADMIN G5CFG}
   {"operator" No No G5ADMIN G5CFG}
   {"configurator" No No G5ADMIN G5CFG}
  }
-server dmgr
-node g5dm
-cell g5cell
}
 :
```

Simplified Schematic Diagram:



```
{-Opt1 -Opt2 {{a b c} {d e f}} -Opt3}
```

Inner List #1
Inner List #2
Argument list for "-Opt2". Outer braces "group up" the inner lists.
Outer-most braces "group up" the options for the install method of $AdminApp

Two ways you can build this with Jacl:

1

```
set inner_1   [list a b c]
set inner_2   [list d e f]
set Opt2_arg  [list $inner_1 $inner_2]
set options   [list -Opt1 -Opt2 $Opt2_arg -Opt3]
```

Build up the nested lists, starting from the inner-most and working outwards

2

```
set options [list -Opt1 -Opt2 [list [list a b c] [list d e f]] -Opt3]
```

Nesting `list` functions inside one another

This is one of the most challenging aspects of WSADMIN and Jacl -- understanding exact structure of option syntax, and matching up the braces.

$AdminConfig Object

`$AdminConfig` is used to create, modify or delete things in the configuration. This object has quite a few methods:

```
$AdminConfig help

  attributes                 required
  checkin                    reset
  convertToCluster           save
  create                     setCrossDocumentValidationEnabled
  createClusterMember        setSaveMode
  createDocument             setValidationLevel
  installResourceAdapter     show
  createUsingTemplate        showall
  defaults                   showAttribute
  deleteDocument             types
  existsDocument             validate
  extract
  getCrossDocumentValidationEnabled
  getid
  getObjectName
  getSaveMode
  getValidationLevel
  getValidationSeverityResult
  hasChanges
  help
  list
  listTemplates
  modify
  parents
  queryChanges
  remove
```

Further, these methods operate against configuration "types" -- specific configuration objects such as server, clusters and many more.

```
$AdminConfig types

   AdminService
   Agent
    :
   WASQueueConnectionFactory
   WASTopic
   WASTopicConnectionFactory
   WebContainer
   WebModuleConfig
   WebModuleDeployment
   WorkloadManagementServer
```

255 Total!

When you create or modify part of the configuration, you'll be working against a "type"

Exploring `VirtualHost` Type

First, use `attributes` method to list out the possible attributes for `VirtualHost`:

```
$AdminConfig attributes VirtualHost
  "aliases HostAlias*"
  "mimeTypes MimeEntry*"
  "name String"
```

Three attributes:

- `aliases` -- asterisk on "HostAlias" indicates there's more to this
- `mimeTypes` -- asterisk indicates there's more
- `name` -- no asterisk: this is lowest level. "name" is attribute, a text string is its value

Next, drill down on the HostAlias type with `attributes`:

```
$AdminConfig attributes HostAlias
  "hostname String"
  "port String"
```

Two attributes:

- `hostname` -- a string value
- `port` -- a string value

Finally, use required to determine minimum settings:

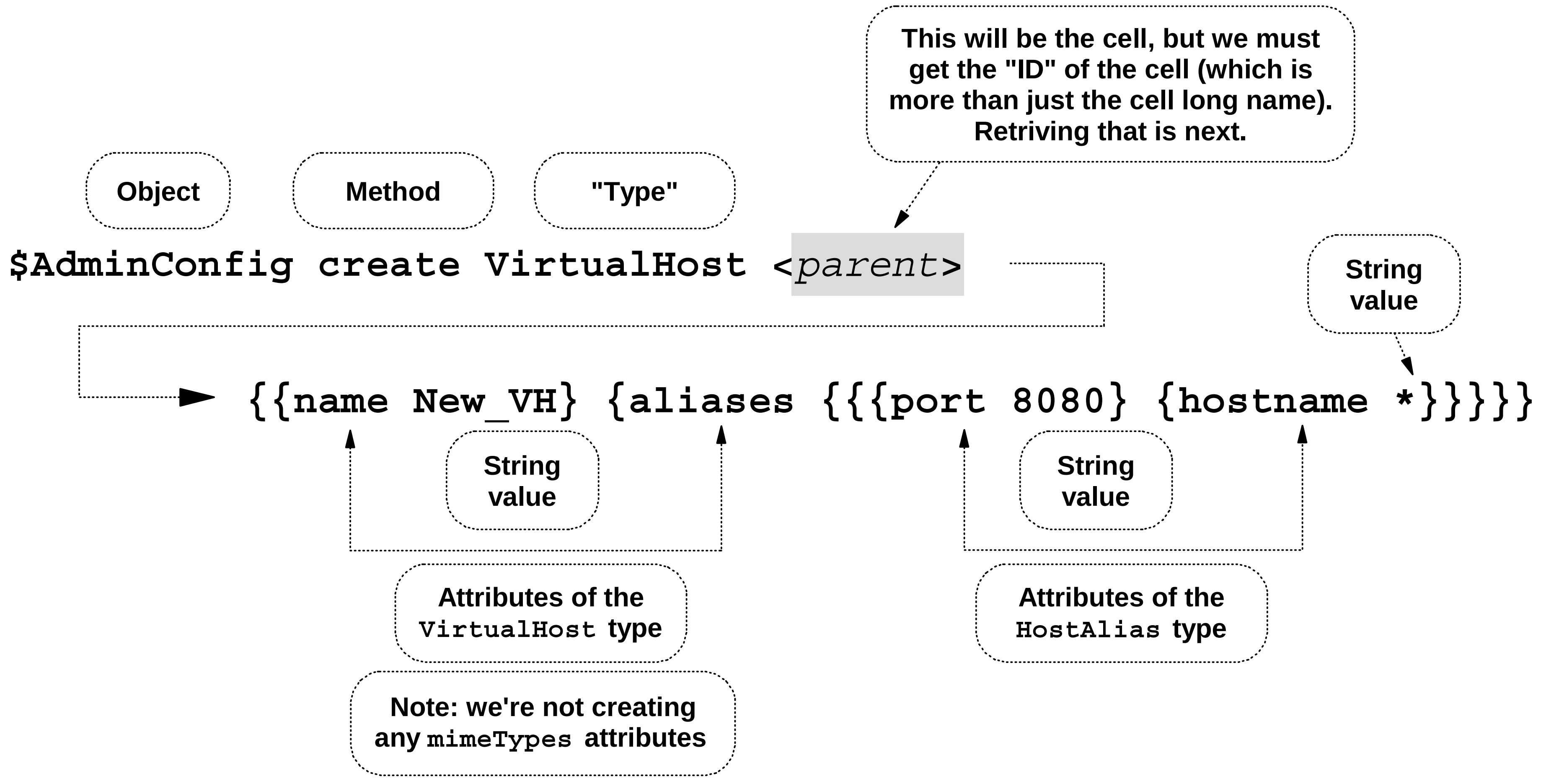
```
$AdminConfig required VirtualHost
  Attribute          Type
  name               String
```

You can get away with only the "name" attribute. VirtualHost won't actually work, but WSADMIN will allow it be created.

Let's see example of actual `$AdminConfig` command to create a new `VirtualHost` ...

Creating a New Virtual Host

The following shows the syntax of a `$AdminConfig` command used to create a new virtual host called `New_VH`. It'll have one port (`8080`) with a hostname of " `*` "



```
$AdminConfig create VirtualHost <parent>
  {{name New_VH} {aliases {{{port 8080} {hostname *}}}}}
```

Object
Method
"Type"
This will be the cell, but we must get the "ID" of the cell (which is more than just the cell long name). Retriving that is next.
String value
String value
String value
Attributes of the VirtualHost type
Attributes of the HostAlias type
Note: we're not creating any mimeTypes attributes

Yes, it would be easier to do this through the Admin Console. One-off things like this won't be what you use WSADMIN for. Repeatable things ... yes.

Using `getid` to Get <*parent*> Value

The `getid` method will return the unique "ID" value for a configuration object. You must supply a "containment path":

```
g5cell(cells/g5cell:cell.xml#Cell_1)
```

"Containment Path"
of the cell long name

```
set cell_id [$AdminConfig getid /Cell:g5cell/]
$AdminConfig create VirtualHost $cell_id ...
```

```
set cell    "g5cell"
set vh_name "New_VH"
set host1   "*"
set port1   "8081"
# ---------------------------------------------------
set cell_id [$AdminConfig getid /Cell:$cell/]
# ---------------------------------------------------
set name         [list "name" $vh_name]
set p1           [list port $port1]
set h1           [list hostname $host1]
set pair1        [list $p1 $h1]
set alias_attrs  [list $pair1]
set aliases      [list aliases $alias_attrs]
set VH_attrs     [list $name $aliases]
# ---------------------------------------------------
$AdminConfig create VirtualHost $cell_id $VH_attrs
$AdminConfig save
```

Notion of "ID" of configuration object becomes critical when `$AdminConfig` is used to modify an existing object.

Let's now turn to the `$AdminControl` object ...

The `$AdminControl` Object

`$AdminControl help`

```
getMBeanInfo_jmx
getNode
getPort
getType
help
invoke_jmx
invoke
isRegistered_jmx
isRegistered
makeObjectName
queryNames_jmx
queryNames
reconnect
setAttribute_jmx
setAttribute
setAttributes_jmx
startServer
stopServer
testConnection
trace
```

The `$AdminControl` object is useful only in "Remote" mode where WSADMIN is connected to a server process

If `-conntype NONE` used, `$AdminControl` considerably hobbled

Further, WSADMIN must be connected to a server in which the Admin Application is running

Possible to connect to Node Agent or AppServer in ND configuration, but `$AdminControl` won't work.

Examples:

```
$AdminControl startServer g5sr01c g5nodec

$AdminControl stopServer g5sr01c g5nodec
```

A very important `$AdminControl` method is `invoke` ... that's used to synchronize to the nodes in a Network Deployment configuration ...

Using `invoke` Method to Sync Nodes

Updates made to the "master configuration" are not usable until they are "synchronized" to the nodes. This is done with the `invoke` method:

Synchronizing with a single, specific node

```
WebSphere:platform=common,cell=g5cell,version=5.0,name=nodeSync,
  mbeanIdentifier=nodeSync,type=NodeSync,node=g5nodec,process=nodeagent
```

```
set var [$AdminControl completeObjectName type=NodeSync,node=g5nodec,*]
$AdminControl invoke $var sync
```

Synchronizing with multiple nodes

All nodes in a cell:

```
set node_ids [$AdminConfig list Node]
foreach node $node_ids {
 set node_name [$AdminConfig showAttribute $node name]
 set nodeSync  [$AdminControl completeObjectName type=NodeSync,node=$node_name,*]
 if { !($nodeSync=="") } then {
  $AdminControl invoke $nodeSync sync
 }
}
```

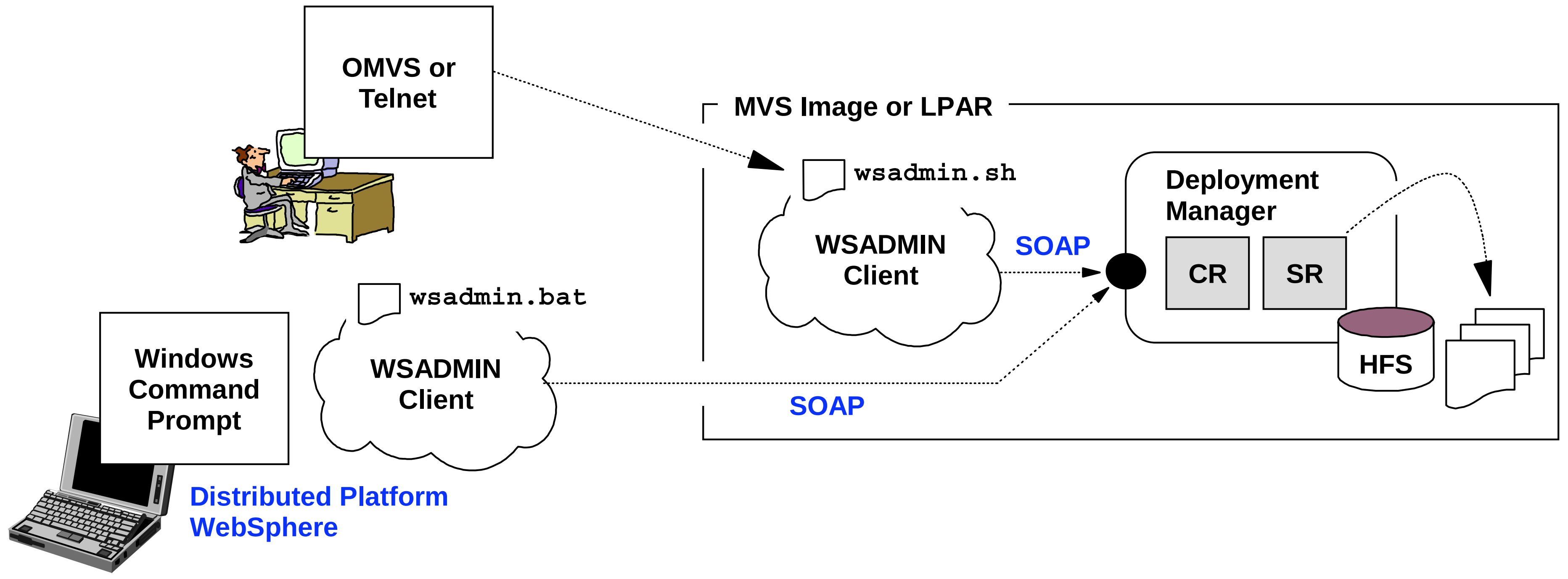
"If" structure checks to make sure node is not DMGR node. If not, then synchronize.

All nodes across which cluster is defined:

```
set c_id     [$AdminConfig getid /ServerCluster:g5sr02cluster/]
set c_membs  [$AdminConfig list ClusterMember $c_id]
foreach m_id $c_membs {
 set node_name [$AdminConfig showAttribute $m_id nodeName]
 set nodeSync [$AdminControl completeObjectName type=NodeSync,node=$node_name,*]
 set work [$AdminControl invoke $nodeSync sync]
}
```

If Global Security Enabled

Affects how you invoke WSADMIN in "remote" mode. ("Local" mode is unaffected by global security because it doesn't go through server.)



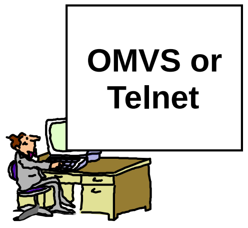
Two things:

- Pass `-user` and `-password` in on invocation of WSADMIN:

```
./wsadmin.sh -conntype SOAP ... -port 15510 -user g5admin -password ######
```

- Insure ID under which WSADMIN runs has proper CA Certificate in keyring
  Must have CA certificate used to sign default certificate of the DMGR controller ID's keyring

User/Password Passed In



```
./wsadmin.sh -conntype SOAP -host wsc3.washington.ibm.com
  -port 15510 -user G5ADMIN -password XXXXXXX -f /u/user1/test.jacl
```

Couple of points:

- **The userid and password you send in needs to have READ access to the EJBROLE profile defined for the WebSphere cell**
  - **Does not *have* to be the "WAS Admin ID," but that will by default have access**
- **You can hard-code this into the `soap.client.props` file and avoid having to send it in on each command line:**

```
# JMX SOAP connector identity
com.ibm.SOAP.loginUserid=G5ADMIN
com.ibm.SOAP.loginPassword=XXXXXX
```

No user passed in

```
BBOO0222I SECJ0305I: Role based authorization check failed for 506
security name <null>, accessId NO_CRED_NO_ACCESS_ID while invoking
method getProcessType on resource Server and module Server.
```

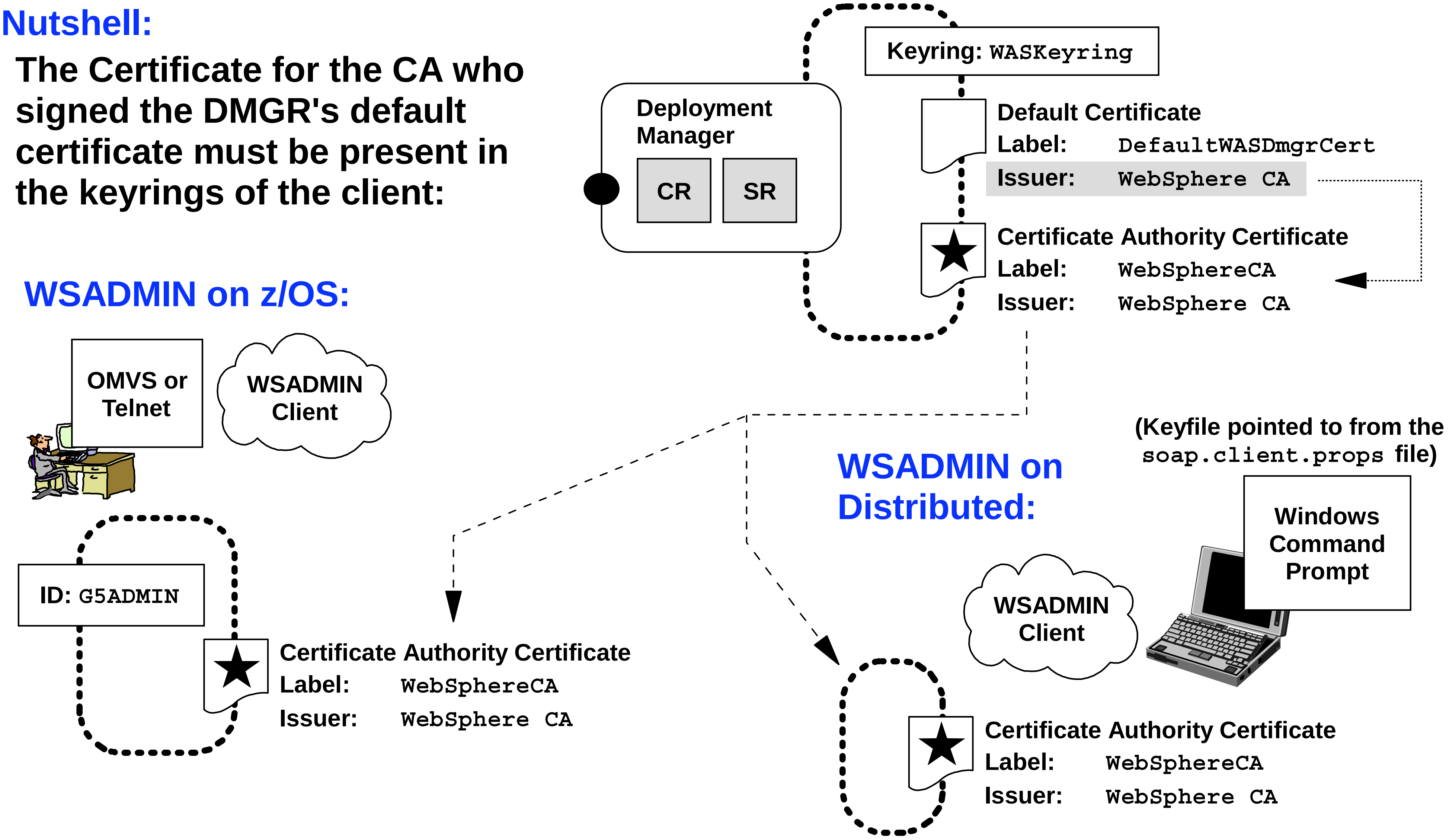
User passed in, not in EJBROLE

```
BBOO0222I SECJ0305I: Role based authorization check failed for 507
security name <plex/ID>, accessId user:<plex/ID> while
invoking method getRepositoryEpoch on resource ConfigRepository and
module ConfigRepository.
```

WSADMIN and CA Certificates

Nutshell:

The Certificate for the CA who signed the DMGR's default certificate must be present in the keyrings of the client:



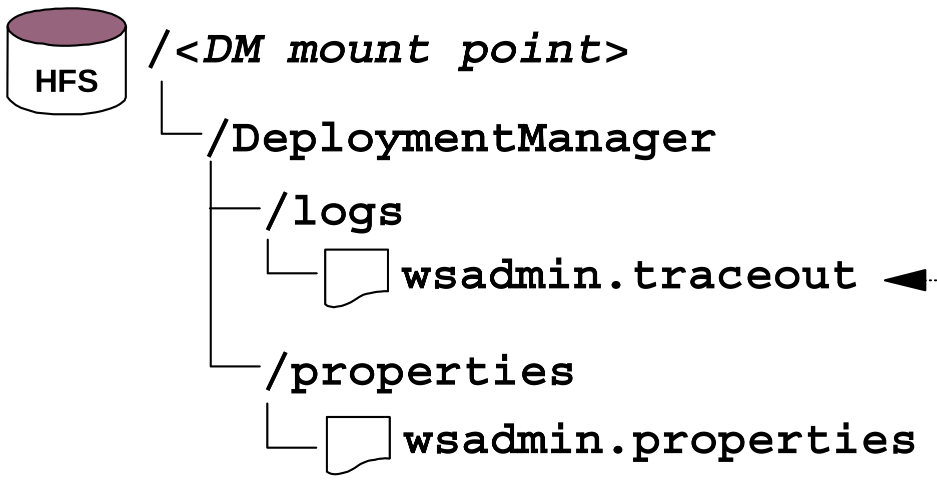
WSADMIN on z/OS:

WSADMIN on Distributed:

Error symptom:

```
WASX7023E: Error creating "SOAP" connection to host "<host>"; exception information:
com.ibm.websphere.management.exception.ConnectorNotAvailableException
```

Tracing of the WSADMIN Activities



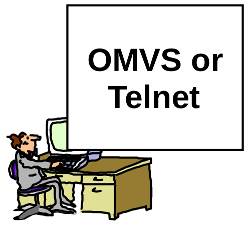
```
/<DM mount point>
  └─/DeploymentManager
      ├─/logs
      │   └─wsadmin.traceout
      └─/properties
          └─wsadmin.properties
```

Two ways to control tracing:

- Static from within properties file
- Dynamically with `$AdminControl`

```
:
com.ibm.ws.scripting.traceFile=/DeploymentManager/logs/wsadmin.traceout
:
#com.ibm.ws.scripting.traceString=com.ibm.*=all=enabled
:
```

Default state: off



```
$AdminControl trace com.ibm.*=all=enabled

$AdminControl trace com.ibm.*=all=disabled
```