

MCT USE ONLY. STUDENT USE PROHIBITED



**2433B: Microsoft® Visual Basic®, Scripting
Edition and Microsoft Windows® Script Host
Essentials**

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

Product Number: 2433B

Part Number: X18-43364

Released: 10/2012

MICROSOFT LICENSE TERMS

OFFICIAL MICROSOFT LEARNING PRODUCTS COURSEWARE – STUDENT EDITION

These license terms are an agreement between Microsoft Corporation and you. Please read them. They apply to the licensed content named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

- updates,
- supplements,
- Internet-based services, and
- support services

for this licensed content, unless other terms accompany those items. If so, those terms apply.

By using the licensed content, you accept these terms. If you do not accept them, do not use the licensed content.

If you comply with these license terms, you have the rights below.

1. OVERVIEW.

Licensed Content. The licensed content includes software, printed materials, academic materials (online and electronic), and associated media.

License Model. The licensed content is licensed on a per copy per device basis.

2. INSTALLATION AND USE RIGHTS.

- a. **Licensed Device.** The licensed device is the device on which you use the licensed content. You may install and use one copy of the licensed content on the licensed device.
- b. **Portable Device.** You may install another copy on a portable device for use by the single primary user of the licensed device.
- c. **Separation of Components.** The components of the licensed content are licensed as a single unit. You may not separate the components and install them on different devices.
- d. **Third Party Programs.** The licensed content may contain third party programs. These license terms will apply to your use of those third party programs, unless other terms accompany those programs.

3. ADDITIONAL LICENSING REQUIREMENTS AND/OR USE RIGHTS.

- a. **Media Elements and Templates.** You may use images, clip art, animations, sounds, music, shapes, video clips and templates provided with the licensed content solely for your personal training use. If you wish to use these media elements or templates for any other purpose, go to www.microsoft.com/permission to learn whether that use is allowed.
- b. **Academic Materials.** If the licensed content contains academic materials (such as white papers, labs, tests, datasheets and FAQs), you may copy and use the academic materials. You may not make any modifications to the academic materials and you may not print any book (either electronic or print version) in its entirety. If you reproduce any academic materials, you agree that:
 - The use of the academic materials will be only for your personal reference or training use
 - You will not republish or post the academic materials on any network computer or broadcast in any media;
 - You will include the academic material's original copyright notice, or a copyright notice to Microsoft's benefit in the format provided below:

Form of Notice:

© 2007 Reprinted for personal reference use only with permission by Microsoft Corporation. All rights reserved.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the US and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

MCT USE ONLY. STUDENT USE PROHIBITED

c. **Distributable Code.** The licensed content may contain code that you are permitted to distribute in programs you develop if you comply with the terms below.

i. **Right to Use and Distribute.** The code and text files listed below are "Distributable Code."

- REDIST.TXT Files. You may copy and distribute the object code form of code listed in REDIST.TXT files.
- Sample Code. You may modify, copy, and distribute the source and object code form of code marked as "sample."
- Third Party Distribution. You may permit distributors of your programs to copy and distribute the Distributable Code as part of those programs.

ii. **Distribution Requirements.** For any Distributable Code you distribute, you must

- add significant primary functionality to it in your programs;
- require distributors and external end users to agree to terms that protect it at least as much as this agreement;
- display your valid copyright notice on your programs; and
- indemnify, defend, and hold harmless Microsoft from any claims, including attorneys' fees, related to the distribution or use of your programs.

iii. **Distribution Restrictions.** You may not

- alter any copyright, trademark or patent notice in the Distributable Code;
- use Microsoft's trademarks in your programs' names or in a way that suggests your programs come from or are endorsed by Microsoft;
- distribute Distributable Code to run on a platform other than the Windows platform;
- include Distributable Code in malicious, deceptive or unlawful programs; or
- modify or distribute the source code of any Distributable Code so that any part of it becomes subject to an Excluded License. An Excluded License is one that requires, as a condition of use, modification or distribution, that
 - the code be disclosed or distributed in source code form; or
 - others have the right to modify it.

4. **INTERNET-BASED SERVICES.** Microsoft may provide Internet-based services with the licensed content. It may change or cancel them at any time. You may not use these services in any way that could harm them or impair anyone else's use of them. You may not use the services to try to gain unauthorized access to any service, data, account or network by any means.

5. **Scope of License.** The licensed content is licensed, not sold. This agreement only gives you some rights to use the licensed content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the licensed content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the licensed content that only allow you to use it in certain ways. You may not

- disclose the results of any benchmark tests of the licensed content to any third party without Microsoft's prior written approval;
- work around any technical limitations in the licensed content;
- reverse engineer, decompile or disassemble the licensed content, except and only to the extent that applicable law expressly permits, despite this limitation;
- make more copies of the licensed content than specified in this agreement or allowed by applicable law, despite this limitation;
- publish the licensed content for others to copy;
- rent, lease or lend the licensed content; or
- use the licensed content for commercial licensed content hosting services.
- Rights to access the server software that may be included with the Licensed Content, including the Virtual Hard Disks does not give you any right to implement Microsoft patents or other Microsoft intellectual property in software or devices that may access the server.

6. **BACKUP COPY.** You may make one backup copy of the licensed content. You may use it only to reinstall the licensed content.
7. **TRANSFER TO ANOTHER DEVICE.** You may uninstall the licensed content and install it on another device for your use. You may not do so to share this license between devices.
8. **TRANSFER TO A THIRD PARTY.** The first user of the licensed content may transfer it and this agreement directly to a third party. Before the transfer, that party must agree that this agreement applies to the transfer and use of the licensed content. The first user must uninstall the licensed content before transferring it separately from the device. The first user may not retain any copies.
9. **EXPORT RESTRICTIONS.** The licensed content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the licensed content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
10. **NOT FOR RESALE SOFTWARE/LICENSED CONTENT.** You may not sell software or licensed content marked as "NFR" or "Not for Resale."
11. **ACADEMIC EDITION.** You must be a "Qualified Educational User" to use licensed content marked as "Academic Edition" or "AE." If you do not know whether you are a Qualified Educational User, visit www.microsoft.com/education or contact the Microsoft affiliate serving your country.
12. **ENTIRE AGREEMENT.** This agreement, and the terms for supplements, updates, Internet-based services and support services that you use, are the entire agreement for the licensed content and support services.
13. **APPLICABLE LAW.**
 - a. **United States.** If you acquired the licensed content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
 - b. **Outside the United States.** If you acquired the licensed content in any other country, the laws of that country apply.
14. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the licensed content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
15. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS." YOU BEAR THE RISK OF USING IT. MICROSOFT GIVES NO EXPRESS WARRANTIES, GUARANTEES OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT EXCLUDES THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
16. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO U.S. \$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the licensed content, software, services, content (including code) on third party Internet sites, or third party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this licensed content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut

modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence , aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers ; et
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE LICENSED CONTENT, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE LICENSED CONTENT, OR OTHERWISE ARISING OUT OF THE USE OF THE LICENSED CONTENT. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE LICENSED CONTENT. THE ENTIRE RISK AS TO THE QUALITY, OR ARISING OUT OF THE USE OR PERFORMANCE OF THE LICENSED CONTENT, AND ANY SUPPORT SERVICES, REMAINS WITH YOU.

16. **EXCLUSION OF INDIRECT DAMAGES.** TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, PUNITIVE, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE LICENSED CONTENT, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE LICENSED CONTENT, OR OTHERWISE ARISING OUT OF THE USE OF THE LICENSED CONTENT, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), MISREPRESENTATION, STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

17. **LIMITATION OF LIABILITY.** NOTWITHSTANDING ANY DAMAGES THAT YOU MIGHT INCUR FOR ANY REASON WHATSOEVER (INCLUDING, WITHOUT LIMITATION, ALL DAMAGES REFERENCED HEREIN AND ALL DIRECT OR GENERAL DAMAGES IN CONTRACT OR ANYTHING ELSE), THE ENTIRE LIABILITY OF MICROSOFT AND ANY OF ITS SUPPLIERS UNDER ANY PROVISION OF THIS EULA AND YOUR EXCLUSIVE REMEDY HEREUNDER SHALL BE LIMITED TO THE GREATER OF THE ACTUAL DAMAGES YOU INCUR IN REASONABLE RELIANCE ON THE LICENSED CONTENT UP TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE LICENSED CONTENT OR US\$5.00. THE FOREGOING LIMITATIONS, EXCLUSIONS AND DISCLAIMERS SHALL APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, EVEN IF ANY REMEDY FAILS ITS ESSENTIAL PURPOSE.

18. **APPLICABLE LAW.** If You acquired this Licensed Content in the United States, this EULA is governed by the laws of the State of Washington, and, in respect of any dispute which may arise hereunder, You consent to the jurisdiction of the federal and state courts located in King County, Washington. If You acquired this Licensed Content in Canada, unless expressly prohibited by local law, this EULA is governed by the laws in force in the Province of Ontario, Canada; and, in respect of any dispute which may arise hereunder, You consent to the jurisdiction of the federal and provincial courts sitting in Toronto, Ontario. If You acquired this Licensed Content in the European Union, Iceland, Norway, or Switzerland, then the local law of such jurisdictions applies. If You acquired this Licensed Content in any other country, then local law may apply.

19. **ENTIRE AGREEMENT; SEVERABILITY.** This EULA (including any addendum or amendment to this EULA which is included with the Licensed Content) is the entire agreement between You and Microsoft relating to the Licensed Content and the support services (if any) and supersedes all prior or contemporaneous oral or written communications, proposals and representations with respect to the Licensed Content or any other subject matter covered by this EULA. To the extent the terms of any Microsoft policies or programs for support services conflict with the terms of this EULA, the terms of this EULA shall control. If any provision of this EULA is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.

Should You have any questions concerning this EULA, or if You desire to contact Microsoft for any reason, please use the address information enclosed in this Licensed Content to contact the Microsoft subsidiary serving Your country or visit Microsoft on the World Wide Web at <http://www.microsoft.com>.

Si vous avez acquis votre Contenu Sous Licence Microsoft au CANADA :

DÉNI DE GARANTIES. Dans la mesure maximale permise par les lois applicables, le Contenu Sous Licence et les services de soutien technique (le cas échéant) sont fournis *TELS QUELS ET AVEC TOUS LES DÉFAUTS* par Microsoft et ses fournisseurs, lesquels par les présentes dénient toutes autres garanties et conditions expresses, implicites ou en vertu de la loi, notamment, mais sans limitation, (le cas échéant) les garanties, devoirs ou conditions

implicites de qualité marchande, d'adaptation à une fin usage particulière, de fiabilité ou de disponibilité, d'exactitude ou d'exhaustivité des réponses, des résultats, des efforts déployés selon les règles de l'art, d'absence de virus et d'absence de négligence, le tout à l'égard du Contenu Sous Licence et de la prestation des services de soutien technique ou de l'omission de la 'une telle prestation des services de soutien technique ou à l'égard de la fourniture ou de l'omission de la fourniture de tous autres services, renseignements, Contenus Sous Licence, et contenu qui s'y rapporte grâce au Contenu Sous Licence ou provenant autrement de l'utilisation du Contenu Sous Licence. PAR AILLEURS, IL N'Y A AUCUNE GARANTIE OU CONDITION QUANT AU TITRE DE PROPRIÉTÉ, À LA JOUSSANCE OU LA POSSESSION PAISIBLE, À LA CONCORDANCE À UNE DESCRIPTION NI QUANT À UNE ABSENCE DE CONTREFAÇON CONCERNANT LE CONTENU SOUS LICENCE.

EXCLUSION DES DOMMAGES ACCESSOIRES, INDIRECTS ET DE CERTAINS AUTRES DOMMAGES. DANS LA MESURE MAXIMALE PERMISE PAR LES LOIS APPLICABLES, EN AUCUN CAS MICROSOFT OU SES FOURNISSEURS NE SERONT RESPONSABLES DES DOMMAGES SPÉCIAUX, CONSÉCUTIFS, ACCESSOIRES OU INDIRECTS DE QUELQUE NATURE QUE CE SOIT (NOTAMMENT, LES DOMMAGES À L'ÉGARD DU MANQUE À GAGNER OU DE LA DIVULGATION DE RENSEIGNEMENTS CONFIDENTIELS OU AUTRES, DE LA PERTE D'EXPLOITATION, DE BLESSURES CORPORELLES, DE LA VIOLATION DE LA VIE PRIVÉE, DE L'OMISSION DE REMPLIR TOUT DEVOIR, Y COMPRIS D'AGIR DE BONNE FOI OU D'EXERCER UN SOIN RAISONNABLE, DE LA NÉGLIGENCE ET DE TOUTE AUTRE PERTE PÉCUNIAIRE OU AUTRE PERTE DE QUELQUE NATURE QUE CE SOIT) SE RAPPORTANT DE QUELQUE MANIÈRE QUE CE SOIT À L'UTILISATION DU CONTENU SOUS LICENCE OU À L'INCAPACITÉ DE S'EN SERVIR, À LA PRESTATION OU À L'OMISSION DE LA 'UNE TELLE PRESTATION DE SERVICES DE SOUTIEN TECHNIQUE OU À LA FOURNITURE OU À L'OMISSION DE LA FOURNITURE DE TOUS AUTRES SERVICES, RENSEIGNEMENTS, CONTENUS SOUS LICENCE, ET CONTENU QUI S'Y RAPPORTÉ GRÂCE AU CONTENU SOUS LICENCE OU PROVENANT AUTREMENT DE L'UTILISATION DU CONTENU SOUS LICENCE OU AUTREMENT AUX TERMES DE TOUTE DISPOSITION DE LA U PRÉSENTE CONVENTION EULA OU RELATIVEMENT À UNE TELLE DISPOSITION, MÊME EN CAS DE FAUTE, DE DÉLIT CIVIL (Y COMPRIS LA NÉGLIGENCE), DE RESPONSABILITÉ STRICTE, DE VIOLATION DE CONTRAT OU DE VIOLATION DE GARANTIE DE MICROSOFT OU DE TOUT FOURNISSEUR ET MÊME SI MICROSOFT OU TOUT FOURNISSEUR A ÉTÉ AVISÉ DE LA POSSIBILITÉ DE TELS DOMMAGES.

LIMITATION DE RESPONSABILITÉ ET RE COURS. MALGRÉ LES DOMMAGES QUE VOUS PUISSIEZ SUBIR POUR QUELQUE MOTIF QUE CE SOIT (NOTAMMENT, MAIS SANS LIMITATION, TOUS LES DOMMAGES SUSMENTIONNÉS ET TOUS LES DOMMAGES DIRECTS OU GÉNÉRAUX OU AUTRES), LA SEULE RESPONSABILITÉ 'OBLIGATION INTÉGRALE DE MICROSOFT ET DE L'UN OU L'AUTRE DE SES FOURNISSEURS AUX TERMES DE TOUTE DISPOSITION DEU LA PRÉSENTE CONVENTION EULA ET VOTRE RE COURS EXCLUSIF À L'ÉGARD DE TOUT CE QUI PRÉCÈDE SE LIMITE AU PLUS ÉLEVÉ ENTRE LES MONTANTS SUIVANTS : LE MONTANT QUE VOUS AVEZ RÉELLEMENT PAYÉ POUR LE CONTENU SOUS LICENCE OU 5,00 \$US. LES LIMITES, EXCLUSIONS ET DÉNIS QUI PRÉCÈDENT (Y COMPRIS LES CLAUSES CI-DESSUS), S'APPLIQUENT DANS LA MESURE MAXIMALE PERMISE PAR LES LOIS APPLICABLES, MÊME SI TOUT RE COURS N'ATTEINT PAS SON BUT ESSENTIEL.

À moins que cela ne soit prohibé par le droit local applicable, la présente Convention est régie par les lois de la province d'Ontario, Canada. Vous consentez Chacune des parties à la présente reconnaît irrévocablement à la compétence des tribunaux fédéraux et provinciaux siégeant à Toronto, dans de la province d'Ontario et consent à instituer tout litige qui pourrait découler de la présente auprès des tribunaux situés dans le district judiciaire de York, province d'Ontario.

Au cas où Vous auriez des questions concernant cette licence ou que Vous désiriez vous mettre en rapport avec Microsoft pour quelque raison que ce soit, veuillez utiliser l'information contenue dans le Contenu Sous Licence pour contacter la filiale de succursale Microsoft desservant Votre pays, dont l'adresse est fournie dans ce produit, ou visitez écrivez à : Microsoft sur le World Wide Web à <http://www.microsoft.com>

MCT USE ONLY. STUDENT USE PROHIBITED

Table of Contents

Introduction

Introduction	i
Course Materials.....	ii
Microsoft Learning Product Types.....	iv
Microsoft Learning Product Types (continued).....	v
Microsoft Learning	vii
Microsoft Certification Program	viii
Facilities.....	xii
About This Course	xiii
Prerequisites.....	xv
Course Outline.....	xvi
Course Outline (continued).....	xvii
Virtual Machine Environment.....	xviii
Demonstration: Using Microsoft Virtual PC	xx

Module 1: Overview of Windows Scripting Technologies

Module Overview	1-1
Lesson 1: Windows Scripting Technologies.....	1-2
Lesson 2: Introducing WSH.....	1-7
Lesson 3: Running Scripts.....	1-16
Lesson 4: Working with Scripts	1-23
Lab: Configuring and Using WSH.....	1-30

Module 2: Objects in VBScript and WSH

Module Overview	2-1
Lesson 1: Object Terminology	2-2
Lesson 2: Creating and Manipulating Objects.....	2-11
Lesson 3: Object Models	2-16
Lesson 4: Common Object Models	2-30
Lab: Objects in VBScript and WSH	2-44

Module 3: Script Logic

Module Overview	3-1
Lesson 1: Fundamental VBScript Rules.....	3-2
Lesson 2: Variables, Constants, and Data Types	3-6
Lesson 3: Operators	3-23
Lesson 4: Conditions and Loops	3-31
Lesson 5: Procedures.....	3-43
Lesson 6: Script Layout	3-50
Lab: Script Logic.....	3-57

Module 4: Error Handling and Debugging

Module Overview.....	4-1
Lesson 1: Error Handling.....	4-2
Lesson 2: Debugging.....	4-9
Lab: Error Handling and Debugging	4-15

Module 5: ADSI

Module Overview.....	5-1
Lesson 1: Overview of ADSI.....	5-2
Lesson 2: Binding with ADSI	5-10
Lesson 3: ADSI Objects	5-19
Lesson 4: Searching Active Directory	5-27
Lab A: ADO Search	5-33
Lesson 5: Creating New ADSI Objects	5-38
Lesson 6: Managing Security, Shares, and Services by Using ADSI	5-44
Lab B: Scripting Administrative Tasks by Using ADSI	5-53

Module 6: Creating Logon Scripts

Module Overview.....	6-1
Lesson 1: Verifying the WSH Environment	6-2
Lesson 2: Common Logon Script Tasks	6-10
Lab A: Creating Logon Scripts.....	6-27
Lesson 3: Managing Logon Scripts	6-31
Lesson 4: Troubleshooting and Best Practices	6-41
Lab B: Assigning Logon Scripts.....	6-46

Module 7: Administrative Scripts

Module Overview.....	7-1
Lesson 1: Administrative Scripts	7-2
Lesson 2: Producing Event Logs and E-Mail Messages.....	7-10
Lab A: Administrative Scripts.....	7-16
Lesson 3: Managing the Registry	7-21
Lesson 4: Controlling Drives, Folders, and Files.....	7-26
Lab B: File and E-mail Scripts	7-39

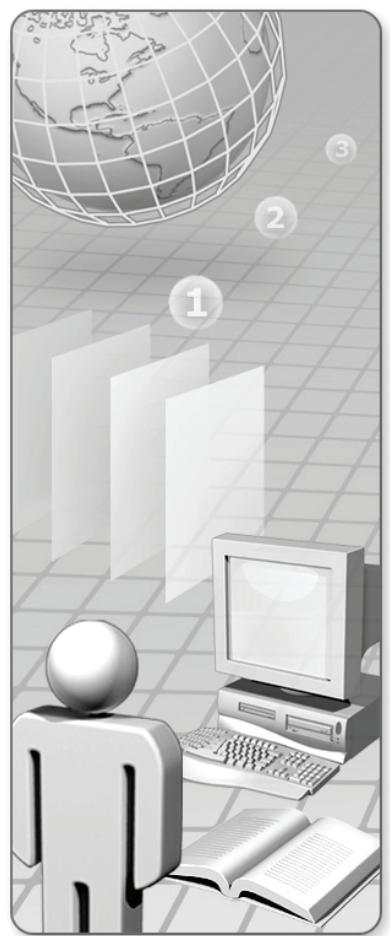
Module 8: WMI

Module Overview.....	8-1
Lesson 1: WMI Overview.....	8-2
Lesson 2: WMI Scripting Basics	8-8
Lesson 3: Common Issues in WMI Scripts.....	8-17
Lesson 4: Scripting Common AdministrativeTasks by Using WMI	8-29
Lab: Writing WMI Scripts	8-38
Course Evaluation	8-43

Introduction

Table of Contents

Introduction	i
Course Materials	ii
Microsoft Learning Product Types	iv
Microsoft Learning Product Types (<i>continued</i>)	v
Microsoft Learning	vii
Microsoft Certification Program	viii
Facilities	xii
About This Course	xiii
Prerequisites	xv
Course Outline	xvi
Course Outline (<i>continued</i>)	xvii
Virtual Machine Environment	xviii
Demonstration: Using Microsoft Virtual PC	xx



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

MCT USE ONLY. STUDENT USE PROHIBITED

Introduction

- Name
- Company affiliation
- Title/function
- Job responsibility
- Networking, programming, or scripting experience
- Expectations

Course Materials

- **Name card**
- **Student workbook**
- **Student Companion Content**
- **Student Course Files**
- **Course evaluation**

The following materials are included with your kit:

- *Name card.* Write your name on both sides of the name card.
- *Student workbook.* The student workbook contains the material covered in class, in addition to the hands-on lab exercises.
- *Student Companion Content on the* [http://www.microsoft.com/learning/companionmoc/ Site](http://www.microsoft.com/learning/companionmoc/). The Student Companion Content is provided in an easy-to-navigate digital format with integrated premium on-line resources designed to supplement the Student workbook. It contains questions and answers, multimedia files, and lab answer keys. Additionally, it contains links to resources pertaining to this course.
- *Student Course Files on the* [http://www.microsoft.com/learning/companionmoc/ Site](http://www.microsoft.com/learning/companionmoc/) : Include the Allfiles.exe, a self-extracting executable file that contains all the files required for the labs and demonstrations.
- *Course evaluation.* Near the end of the course, you will have the opportunity to complete an online evaluation to provide feedback on the course, training facility, and instructor.

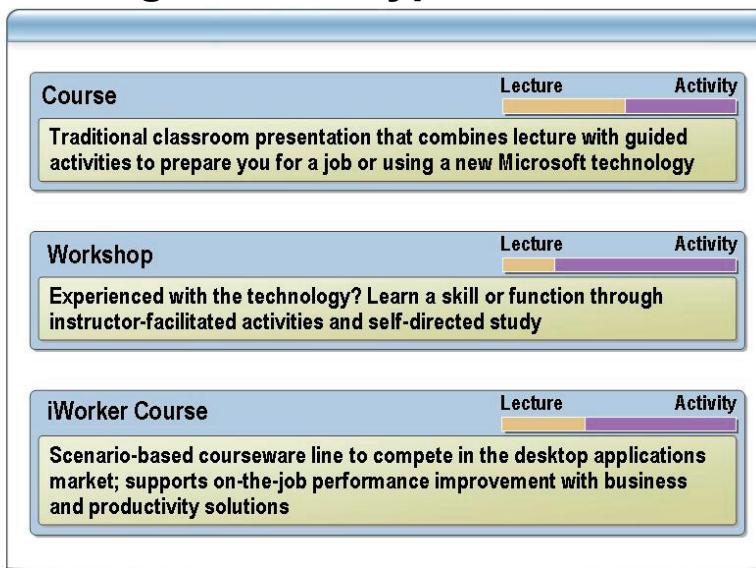
To provide additional comments or feedback on the course, send e-mail to support@mscourseware.com. To inquire about the Microsoft Certified Professional program, send e-mail to mcphelp@microsoft.com.

Document Conventions

The following conventions are used in course materials to distinguish elements of the text.

Convention	Use
Bold	Represents commands, command options, and syntax that must be typed exactly as shown. It also indicates commands on menus and buttons, dialog box titles and options, and icon and menu names.
<i>Italic</i>	In syntax statements or descriptive text, indicates argument names or placeholders for variable information. Italic is also used for introducing new terms, for book titles, and for emphasis in the text.
Title Capitals	Indicate domain names, user names, computer names, directory names, and folder and file names, except when specifically referring to case-sensitive names. Unless otherwise indicated, you can use lowercase letters when you type a directory name or file name in a dialog box or at a command prompt.
ALL CAPITALS	Indicate the names of keys, key sequences, and key combinations—for example, ALT+SPACEBAR.
monospace	Represents code samples or examples of screen text.
[]	In syntax statements, enclose optional items. For example, [filename] in command syntax indicates that you can choose to type a file name with the command. Type only the information within the brackets, not the brackets themselves.
{ }	In syntax statements, enclose required items. Type only the information within the braces, not the braces themselves.
	In syntax statements, separates an either/or choice.
►	Indicates a procedure with sequential steps.
...	In syntax statements, specifies that the preceding item may be repeated.
.	Represents an omitted portion of a code sample.
.	

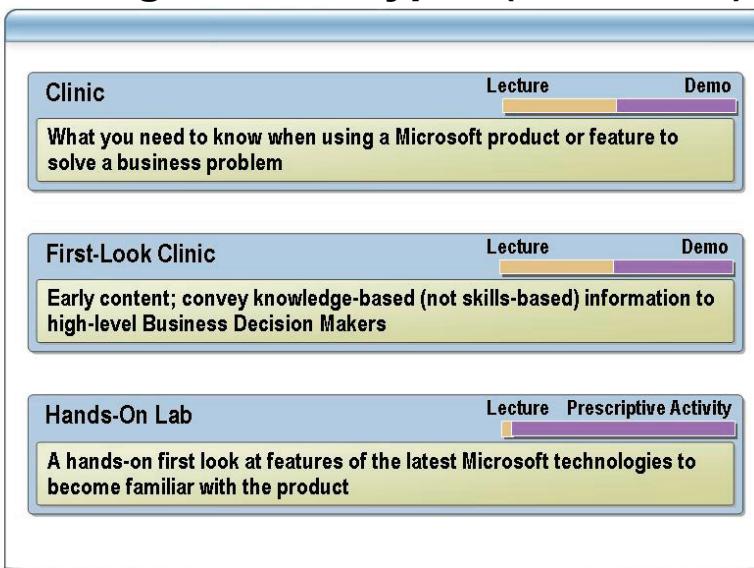
Microsoft Learning Product Types



Microsoft Learning offers the following instructor-led products. Each is specific to a particular audience type and level of experience. The different product types also tend to suit different learning styles. These types are as follows:

- **Courses** are for information technology (IT) professionals and developers who are new to a particular product or technology and for experienced individuals who prefer to learn in a traditional classroom format. Courses provide a relevant and guided learning experience that combines lecture and practice to deliver thorough coverage of a Microsoft product or technology. Courses are designed to address the needs of learners engaged in planning, design, implementation, management, and support phases of the technology adoption life-cycle. They provide detailed information by focusing on concepts and principles, reference content, and in-depth hands-on lab activities to ensure knowledge transfer. Typically, the content of a course is broad, addressing a wide range of tasks necessary for the job role.
- **Workshops** are for knowledgeable IT professionals and developers who learn best by doing and exploring. Workshops provide a hands-on learning experience in which participants use Microsoft products in a safe and collaborative environment based on real-world scenarios.

Microsoft Learning Product Types (*continued*)



- **iWorker Courses**, or Information Worker Courses, are scenario-based courseware lines to compete in the desktop applications market. This scenario-based courseware line will fill a need for applications training that supports on-the-job performance improvement with business and productivity solutions (rather than feature-based training). The purpose of an iWorker course is to promote skills/knowledge transfer in the context of business scenarios to accomplish business objectives by working individually or collaboratively to find answers. iWorker Courses are aimed at users who have working knowledge of the technology and are interested in applying that knowledge in specific business scenarios.
- **Clinics** are for IT professionals, developers and technical decision makers. Clinics offer a detailed “how to” presentation that describes the features and functionality of an existing or new Microsoft product or technology and that showcases product demonstrations and solutions. Clinics focus on how specific features will solve business problems.
- **First-Look Clinics** are products specifically designed to deliver early content or critical information that Product Groups or other internal customers need communicated quickly and broadly. The First Look products convey knowledge-based (not skills-based) information to an audience profile identified as high-level Business Decision Makers.
- **Hands-On Labs** provide IT professionals and developers with hands-on experience with an existing or new Microsoft product or technology. Hands-On Labs provide a realistic and safe environment to encourage knowledge transfer by learning through doing. The labs provided are completely prescriptive so that no lab answer keys are

required. There is very little lecture or text content provided in hands-on labs, aside from lab introductions, context setting, and lab reviews.

Microsoft Learning



Microsoft Learning develops Official Microsoft Learning Product (OMLP) courseware for computer professionals who design, develop, support, implement, or manage solutions by using Microsoft products and technologies. These learning products provide comprehensive, skills-based training in instructor-led and online formats.

Additional Recommended Learning Products

Each learning product relates in some way to other learning products. A related product may be a prerequisite, a follow-up course, clinic, or workshop in a recommended series; or a learning product that offers additional training.

There are no related learning products currently recommended for this course. However, other related learning products may become available in the future, so for up-to-date information about recommended learning products, visit the Microsoft Learning Web site.

Microsoft Learning Information

For more information, visit the Microsoft Learning Web site at
<http://www.microsoft.com/learning/>.

Microsoft Certification Program



Microsoft Learning offers a variety of certification credentials for developers and IT professionals. The Microsoft Certification Program (MCP) is the leading certification program for validating your experience and skills, keeping you competitive in today's changing business environment.

MCP Certifications

The MCP includes the following certifications.

MCITP

The new Microsoft Certified IT Professional (MCITP) credential allows IT professionals to distinguish themselves as experts in their specific area of focus. There is a straightforward upgrade path from the MCDBA certification to the new MCITP credentials. There are currently three IT Professional certifications:

- Microsoft Certified IT Professional: Database Developer
- Microsoft Certified IT Professional: Database Administrator
- Microsoft Certified IT Professional: Business Intelligence Developer

MCPD

The Microsoft Certified Professional Developer (MCPD) credential highlights developer job roles, featuring specific areas of expertise. There is a straightforward upgrade path from the MCAD and MCSA for Microsoft .NET certifications to the new MCPD credential. There are three MCPD certification paths:

- Microsoft Certified Professional Developer: Web Developer
- Microsoft Certified Professional Developer: Windows Developer
- Microsoft Certified Professional Developer: Enterprise Applications Developer

MCTS

The Microsoft Certified Technology Specialist (MCTS) credential enables professionals to target specific technologies and distinguish themselves by demonstrating in-depth knowledge of and expertise in the technologies with which they work. There are currently five MCTS certifications:

- Microsoft Certified Technology Specialist: .NET Framework 2.0 Web Applications
- Microsoft Certified Technology Specialist: .NET Framework 2.0 Windows Applications
- Microsoft Certified Technology Specialist: .NET Framework 2.0 Distributed Applications
- Microsoft Certified Technology Specialist: SQL Server™ 2005
- Microsoft Certified Technology Specialist: BizTalk® Server

MCDST on Microsoft® Windows®

The Microsoft Certified Desktop Support Technician (MCDST) certification is designed for professionals who successfully support and educate end users and troubleshoot operating system and application issues on desktop computers running the Windows operating system.

MCSA on Microsoft Windows Server® 2003

The Microsoft Certified Systems Administrator (MCSA) certification is designed for professionals who implement, manage, and troubleshoot existing network and system environments based on the Windows Server 2003 platform. Implementation responsibilities include installing and configuring parts of systems. Management responsibilities include administering and supporting systems.

MCSE on Microsoft Windows Server 2003

The Microsoft Certified Systems Engineer (MCSE) credential is the premier certification for professionals who analyze business requirements and design and implement infrastructure for business solutions based on the Windows Server 2003 platform.

Implementation responsibilities include installing, configuring, and troubleshooting network systems.

MCAD for Microsoft .NET

The Microsoft Certified Application Developer (MCAD) for Microsoft .NET credential provides industry recognition for professional developers who use Microsoft Visual Studio® .NET and Web services to develop and maintain department-level applications, components, Web or desktop clients, or back-end data services, or who work in teams developing enterprise applications. The credential covers job tasks ranging from developing to deploying and maintaining these solutions.

MCSD for Microsoft .NET

The Microsoft Certified Solution Developer (MCSD) for Microsoft .NET credential is the top-level certification for advanced developers who design and develop leading-edge enterprise solutions by using Microsoft development tools and technologies as well as the Microsoft .NET Framework. The credential covers job tasks ranging from analyzing business requirements to maintaining solutions.

MCDBA on Microsoft SQL Server 2000

The Microsoft Certified Database Administrator (MCDBA) credential is the premier certification for professionals who implement and administer SQL Server 2000 databases. The certification is appropriate for individuals who derive physical database designs, develop logical data models, create physical databases, create data services by using Transact-SQL, manage and maintain databases, configure and manage security, monitor and optimize databases, and install and configure SQL Server.

MCP

The Microsoft Certified Professional (MCP) credential is for individuals who have the skills to successfully implement a Microsoft product or technology as part of a business solution in an organization. Hands-on experience with the product is necessary to successfully achieve certification.

MCT

Microsoft Certified Trainers (MCTs) demonstrate the instructional and technical skills that qualify them to deliver Official Microsoft Learning Products through a Microsoft Certified Partner for Learning Solutions (CPLS).

Certification Requirements

Certification requirements differ for each certification category and are specific to the products and job functions addressed by the certification. To earn a certification credential, you must pass rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise.

Additional Information: See the Microsoft Learning Web site at <http://www.microsoft.com/learning/>. You can also send e-mail to mcphelp@microsoft.com if you have specific certification questions.

Acquiring the Skills Tested by MCP Exams

Official Microsoft Learning Products can help you develop the skills that you need to do your job. They also complement the experience that you gain while working with Microsoft products and technologies. However, no one-to-one correlation exists between Official Microsoft Learning Products and MCP exams. Microsoft does not expect or intend for the courses to be the sole preparation method for passing MCP exams.

Practical product knowledge and experience are also necessary to pass MCP exams.

To help prepare for MCP exams, use the preparation guides that are available for each exam. Each Exam Preparation Guide contains exam-specific information, such as a list of the topics on which you will be tested. These guides are available on the Microsoft Learning Web site at <http://www.microsoft.com/learning/>.

Facilities



- Class hours
- Building hours
- Parking
- Restrooms
- Meals
- Phones
- Messages
- Smoking
- Recycling

About This Course

- **Description**
- **Course Objectives**
- **Audience**

This section provides you with a brief description of the course, objectives, and target audience.

Description

This course teaches you how to write administrative scripts for your Windows-based enterprises. It introduces Windows Scripting Host (WSH) and Microsoft Visual Basic, Scripting Edition (VBScript). It also shows you how to use common scripting techniques and outlines technologies that you can use with WSH and Visual Basic, Scripting Edition to manage resources on your network. Topics include error handling and debugging, using Visual Basic, Scripting Edition to access Active Directory Service Interfaces (ADSI) and Windows Management Instrumentation (WMI), and common administrative tasks that you can perform with scripts.

Objectives

After completing this course, you will be able to:

- Describe WSH and associated scripting technologies.
- Use objects in code written in Visual Basic, Scripting Edition.
- Master the essentials of the Visual Basic, Scripting Edition language.
- Master error handling and debugging by using Visual Basic, Scripting Edition.
- Use Visual Basic, Scripting Edition to interact with ADSI.
- Develop logon, logoff, startup, and shutdown scripts.
- Develop scripts that perform common administrative tasks.
- Use Visual Basic, Scripting Edition to interact with WMI.

Audience

The audience for this course is systems administrators for Windows Server 2003 who must learn how to develop administrative scripts for their enterprise networks.

Prerequisites

- Practical experience of using and administering Windows Server 2003
- Practical experience of using and administering the Active Directory directory service
- Practical experience of using and administering system security
- Practical experience of using and administering services
- Practical experience of using systems management information
- Awareness of the potential uses of logon scripts

This course requires that you meet the following prerequisites:

- Practical experience of using and administering Windows Server 2003.
- Practical experience of using and administering the Active Directory directory service.
- Practical experience of using and administering system security.
- Practical experience of using and administering services.
- Practical experience of using systems management information.
- Awareness of the potential uses of logon scripts.

Important: This learning product will be most useful to people who intend to use their new skills and knowledge on the job immediately after training.

Course Outline

- **Module 1: Overview of Windows Scripting Technologies**
- **Module 2: Objects in VBScript and WSH**
- **Module 3: Script Logic**
- **Module 4: Error Handling and Debugging**
- **Module 5: ADSI**
- **Module 6: Creating Logon Scripts**
- **Module 7: Administrative Scripts**
- **Module 8: WMI**

Module 1, “Overview of Windows Scripting Technologies” describes the technologies available for scripting on Windows operating systems, focusing on WSH and its requirements. After completing this module, you will be able to run scripts, in addition to writing, debugging, and troubleshooting them.

Module 2, “Objects in VBScript and WSH” discusses what objects are and how they can be used in scripts. After completing this module, you will be able to write scripts that access objects and use an object browser.

Module 3, “Script Logic” introduces the basic concepts of Visual Basic, Scripting Edition and shows you how to use that scripting language. After completing this module, you will understand how to use Visual Basic, Scripting Edition to create effective and efficient administrative scripts.

Module 4, “Error Handling and Debugging” introduces the techniques that are used to handle errors that scripting code encounters. After completing this module, you will be able to create error-handling routines and use a debugger to locate and fix logic problems in scripts.

Course Outline (*continued*)

- **Module 1: Overview of Windows Scripting Technologies**
- **Module 2: Objects in VBScript and WSH**
- **Module 3: Script Logic**
- **Module 4: Error Handling and Debugging**
- **Module 5: ADSI**
- **Module 6: Creating Logon Scripts**
- **Module 7: Administrative Scripts**
- **Module 8: WMI**

Module 5, “ADSI” discusses Active Directory Service Interfaces, and how you can write scripts that bind to ADSI objects and use ADSI methods and properties. After completing this module, you will be able to use Visual Basic, Scripting Edition to interact with ADSI.

Module 6, “Creating Logon Scripts” explains the uses of logon scripts, some issues that you may encounter when you create logon scripts, and how to create and manage logon scripts. After completing this module, you will be able to call logon scripts from batch files, perform common tasks using logon scripts, and assign logon scripts to users.

Module 7, “Administrative Scripts” discusses how scripts that are written by using WSH and Visual Basic, Scripting Edition can perform many common administrative tasks.

After completing this module, you will be able to use scheduling in scripts, send e-mails, manage the registry, and work with drives, folders, and files using scripts.

Module 8, “WMI” describes WMI and how you can use it to query and manage resources on local and remote computers. After completing this module, you will be able to write scripts that gather information and automate common management tasks using WMI.

Virtual Machine Environment

Virtual machine	Used as the:
2433B-LON-DC1	Windows Server 2003 SP1 Domain Controller for the FOURTHCOFFEE domain
2433B-VISTA-CL1- <i>nn</i> *	Windows Vista client for the FOURTHCOFFEE domain

* A virtual machine is provided for each module. *nn* indicates the module number.

This section provides the information for setting up the classroom environment to support the business scenario of the course.

Virtual Machine Configuration

In this course, you will use Microsoft Virtual PC 2007 to perform the labs.

If, when performing the hands-on activities, you make any changes to the virtual machine and do not want to save them, you can close the virtual machine without saving the changes. This will take the virtual machine back to the most recently saved state. To close a virtual machine without saving the changes, perform the following steps: 1. On the virtual machine, on the **Action** menu, click **Close**. 2. In the **Close** dialog box, in the **What do you want the virtual machine to do?** list, click **Turn off and delete changes**, and then click **OK**.

The following table shows the role of each virtual machine used in this course.

Virtual machine	Role
2433B-LON-DC1	Windows Server 2003 SP1 Domain Controller for the FOURTHCOFFEE domain
2433B-VISTA-CL1- <i>nn</i> *	Windows Vista client for the FOURTHCOFFEE domain

* A virtual machine is provided for each module. *nn* indicates the module number.

Software Configuration

The following software is installed on each VM:

- Windows Server 2003 with SP1
- Windows Vista

Course Files

There are files associated with the demonstrations and labs in this course. The files for each module are located on drive E of the module-specific virtual machine.

Classroom Setup

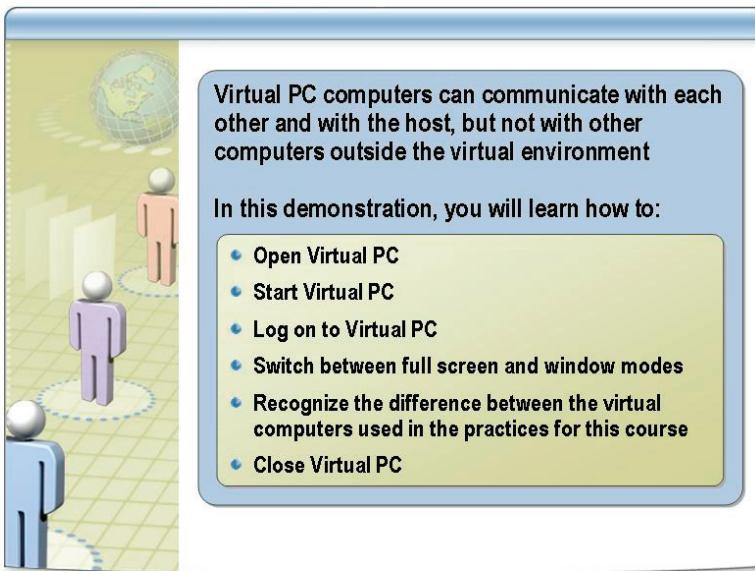
Each classroom computer will have the same virtual machine configured in the same way. Each module-specific virtual machine is configured as a server named LON-DC1 and a client computer running Windows Vista. Network connectivity in the virtual machines is limited to a private virtual network that includes all virtual machines on the same host computer—the virtual machines are not configured to access the external network or the Internet.

Course Hardware Level

To ensure a satisfactory student experience, Microsoft Learning requires a minimum equipment configuration for trainer and student computers in all Microsoft Certified Partner for Learning Solutions (CPLS) classrooms in which Official Microsoft Learning Product courseware is taught.

This course requires that you have a computer that meets or exceeds hardware level 5, which specifies a minimum of two 2.4-gigahertz (GHz) (minimum) Pentium 4 or equivalent CPUs, at least 2 gigabytes (GB) of RAM, 16 megabytes (MB) of video RAM, and a 7200 RPM 40-GB hard disk.

Demonstration: Using Microsoft Virtual PC



In this demonstration, your instructor will help familiarize you with the Virtual PC environment in which you will work to complete the labs in this course. You will learn:

- How to open Virtual PC.
- How to start Virtual PC.
- How to log on to Virtual PC.
- How to switch between full screen and window modes.
- How to tell the difference between the virtual machines that are used in the practices for this course.
- That the virtual machines can communicate with each other and with the host, but they cannot communicate with other computers that are outside of the virtual environment. (For example, no Internet access is available from the virtual environment.)
- How to close Virtual PC.

Keyboard Shortcuts

While working in the Virtual PC environment, you may find it helpful to use keyboard shortcuts. All Virtual PC shortcuts include a key that is referred to as the HOST key or the RIGHT-ALT key. By default, the HOST key is the ALT key on the right side of your keyboard. Some useful shortcuts include:

- RIGHT-ALT+DELETE to log on to the Virtual PC
- RIGHT-ALT+ENTER to switch between full screen mode and window modes
- RIGHT-ALT+RIGHT ARROW to display the next Virtual PC

For more information about Virtual PC, see Virtual PC Help.

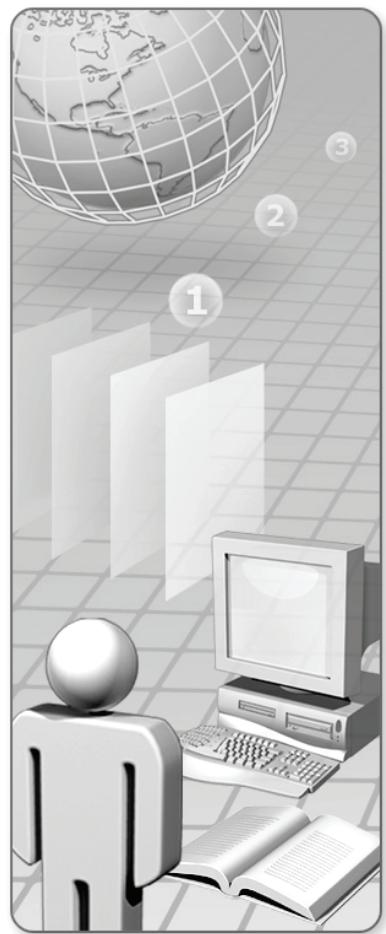
MCT USE ONLY. STUDENT USE PROHIBITED

MCT USE ONLY. STUDENT USE PROHIBITED

Module 1: Overview of Windows Scripting Technologies

Table of Contents

Module Overview	1-1
Lesson 1: Windows Scripting Technologies	1-2
Lesson 2: Introducing WSH	1-7
Lesson 3: Running Scripts	1-16
Lesson 4: Working with Scripts	1-23
Lab: Configuring and Using WSH	1-30



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Windows Scripting Technologies**
- **Introducing WSH**
- **Running Scripts**
- **Working with Scripts**

To use Windows® scripting technologies, you must understand some basic concepts such as how Windows Script Host (WSH) functions and what types of scripts you can run by using WSH. In this module, you will learn about WSH and how to work with script files.

Objectives

After completing this module, you will be able to:

- Describe the Windows scripting technologies.
- Describe WSH.
- Run scripts.
- Work with scripts, including writing, debugging, and troubleshooting.

Lesson 1: Windows Scripting Technologies

- **Windows Script Host**
- **Windows PowerShell**
- **Comparing VBScript and Visual Basic**

You can use several scripting technologies to help to automate system administration tasks on computers running Windows. This lesson will teach you about the various scripting technologies and how they can be used.

Objectives

After completing this lesson, you will be able to:

- Describe WSH.
- Describe Windows PowerShell™ command-line interface.
- Compare Microsoft® Visual Basic®, Scripting Edition (VBScript) and Visual Basic.

Windows Script Host

- **WSH components:**
 - Provide a script environment
 - Are also used by other environments such as Internet Information Services
- **WSH is supported on all Windows operating systems, including:**
 - Windows Vista
 - Windows XP
 - Windows Server 2003
- **WSH scripts can be written in Visual Basic, Scripting Edition or Microsoft JScript**

WSH is a set of components that provides an environment in which to run scripts. WSH is sometimes referred to as Windows Script.

WSH components are present whether you use the Windows Script Host environment or other script environments such as Internet Information Services (IIS).

WSH is supported on all of the 32-bit and 64-bit operating systems that run Windows, such as Windows Vista®, Windows® XP, Windows Server® 2003, Microsoft Windows® 2000, Windows NT® version 4.0, and Windows® 9.x.

The WSH environment does not require that scripts are written in a particular scripting language, so WSH scripts can be written in Visual Basic, Scripting Edition or Microsoft JScript®. Most administrative scripts are written in Visual Basic, Scripting Edition.

Windows PowerShell

- **Windows PowerShell is:**
 - A command-line shell
 - A scripting language
- **Windows PowerShell script-related features:**
 - Cmdlets
 - Scripting language
 - Data access
 - Navigation

Windows PowerShell is both a command-line shell and a scripting language.

Windows PowerShell provides a range of script-related features:

- **Command-line tools.** The Windows PowerShell command-line tools (called cmdlets) are used for performing common system administration tasks such as managing the registry, services, processes, event logs, certificates, and Windows Management Instrumentation (WMI). You can use these cmdlets easily in Windows PowerShell scripts.
- **Scripting language.** The Windows PowerShell scripting language has tight integration with the command-line shell, and also supports existing scripts, existing command-line tools, and multiple operating systems, including Windows Vista, Windows XP, Windows Server 2003, and Windows Server® 2008.
- **Data access.** Windows PowerShell provides standard tools for accessing the full range of Windows-based data access technologies, including data for Active Directory Service Interfaces (ADSI), WMI, HTML, and XML; and objects for Component Object Model (COM), and ActiveX Data Objects (ADO).
- **Navigation.** Windows PowerShell provides simplified, command-based navigation of the operating system that lets you navigate the registry, certificate store, and other data by using the same commands you use to navigate the file system.

Windows PowerShell is an emerging technology and is beyond the scope of this course. For more information about Windows PowerShell, see the Windows PowerShell Technology Center Web site.

Comparing VBScript and Visual Basic

- **VBScript Is an Interpreted Language**
 - Not compiled into an executable
- **Syntax Differences**
 - Code must be in procedures in Visual Basic
 - The only data type is variant
 - No Debug.Print
 - Cannot access type library references directly
 - Constants have to be explicitly declared

Visual Basic, Scripting Edition is a subset of the Visual Basic for Applications language that you can use in the Microsoft Office System and the Visual Basic development system. If you are already familiar with the Visual Basic language, then you will find it easy to write scripts by using Visual Basic, Scripting Edition. However, note that there are some differences between the two languages. Some of the differences are described below.

VBScript Is an Interpreted Language

Visual Basic, Scripting Edition is an interpreted language, which means that code is processed and parsed for syntax accuracy at run time and not prepackaged as an executable file before it is run. Scripts written in Visual Basic, Scripting Edition are slower to execute than scripts written in Visual Basic.

Syntax Differences

Unlike Visual Basic, Visual Basic, Scripting Edition does not require that the main body of the script be enclosed in a **Sub ()** or **Function ()** procedure.

Visual Basic, Scripting Edition has only one data type, called a **Variant**, which can contain many different kinds of data. For example, a **Variant** can contain strings and integers. You cannot declare an explicit data type in Visual Basic, Scripting Edition.

There is no support in Visual Basic, Scripting Edition for the **Debug.Print** statement used in Visual Basic. The equivalent in WSH is **WScript.Echo**.

One limitation of Visual Basic, Scripting Edition is the lack of native support for referencing type library information, which particularly affects constants. If you want to

MCT USE ONLY. STUDENT USE PROHIBITED

use a constant name to reference a value, you must explicitly declare the constant at the start of the script.

This is not a definitive list of the differences between the languages. It simply illustrates that there are some important differences that will affect the ability to reuse code examples from one language in another language in the code.

Lesson 2: Introducing WSH

- **WSH Environment**
- **Features of WSH**
- **Types of Script Files**

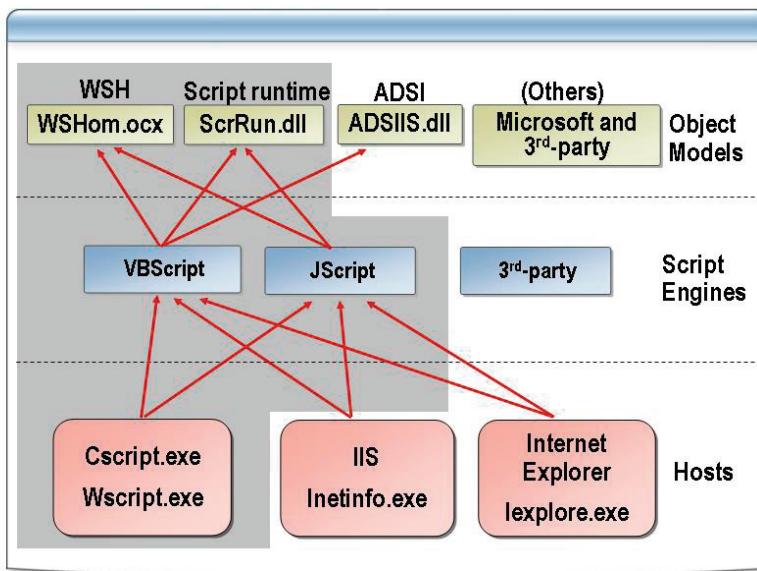
WSH includes several component technologies. In this lesson, you will learn about the WSH environment and the features of WSH. The lesson will compare the types of script files and describe how they are used.

Objectives

After completing this lesson, you will be able to:

- Describe the WSH environment.
- Describe the features of WSH.
- Compare the types of script files.

WSH Environment



WSH has several components. These components work together to make the operating system run and interoperate with scripts.

Hosts

Hosts are the programs that run your scripts. Before WSH was developed, IIS and the Microsoft Internet Explorer® internet browser were the only hosts available from Microsoft. As a result, scripting was restricted to the Web. However, by adding CScript.exe and WScript.exe, administrators had the tools to run scripts outside the Web environment. The WSH hosts accept scripts and determine which script engine you need to parse, interpret, and execute those scripts.

By using WSH 5.6, hosts can be remote computers. You can load scripts onto several remote computer systems and start them all running simultaneously. While a remote script runs, you can check its progress. After the script completes, you can ensure that the script ran correctly or determine the cause of any premature termination.

Note: For more information about using WSH with remote hosts, see “What’s New in WSH 5.6” on the MSDN Web site.

Script Engines

The WSH environment natively supports two scripting languages: Visual Basic, Scripting Edition and JScript.

After the scripting host determines the language used in your scripts, it loads the appropriate script engine and passes your script to it for execution. WSH includes two different script engines, one for VBScript and one for JScript, to run scripts that you write in these languages.

Object Models

Script programming usually makes extensive use of *objects*. Objects represent packaged functionality that you can reuse in your scripts. You use objects in your scripts to perform many tasks. The advantage of using objects is that they perform complex tasks without requiring you to understand how to implement the code they contain.

Related objects are defined in structures called object models. The object models are defined by *type libraries* contained in dynamic-link libraries (DLLs) or ActiveX control files.

When you install WSH, you install the hosts; two native script engines and two object models; Windows Script Host; and the Script runtime object models.

Note: For more information about objects and specific object models, see Module 2, “Working with Objects,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Extensibility

The WSH environment is extensible. You can install additional script engines to use new script languages. You can also install new type libraries to gain access to additional object models.

Features of WSH

- **Small Memory Footprint**
- **Language-Independent**
 - You can extend WSH to other script engines
- **Script Reuse**
- **Command-Line Reference**

WSH is ideal for non-interactive scripting requirements such as batch, logon, and administrative scripting.

In addition to non-interactive processing, WSH scripts can involve some user interaction such as confirming prompts and entering variable information.

Small Memory Footprint

The low system overhead of running scripts by using WSH makes it ideal for running administrative scripts.

Language-Independent

Although WSH natively supports the Visual Basic, Scripting Edition and JScript languages, you can extend the environment so that it can run scripts written in other languages such as PerlScript.

Note: PerlScript is a third-party script language developed by ActiveState. For more information, see the ActiveState Web site at www.activestate.com.

Script Reuse

By using WSH, you can save your scripts as operating system files. The content of your script files is plain text. Therefore, you can write and save your script files by using a simple text editor such as Notepad. After you have saved the script files, you can run your scripts many times; you do not need to rewrite code every time you want to run a certain set of actions. This is a useful feature if you have scripts that you must run regularly.

Command-Line Reference

Prior to WSH, the only native script language that the Windows operating system supported was the command language supported through Cmd.exe.

Administrators often construct batch files (.bat) that contain multiple commands. These commands are still supported and are an important part of the administrator's toolset.

In addition, specially written command utilities, such as those provided in the Windows Server 2003 Resource Kit, can be the most efficient way to get a task done.

A full command-line reference is provided in Windows Help and Support.

Types of Script Files

- **Script Files**
 - .vbs and .vbe
 - .js and .jse
- **Script Control Files**
 - .wsh
- **XML Script Files**
 - .wsf
 - .wsc

Working in the WSH environment involves using many file types. WSH identifies the type of file by its file extension. It is essential to understand the role that each of these file types can play within the WSH environment.

The following table describes the common file types used in the WSH environment.

File suffix	Role
.vbs and .vbe	Script file written in VBScript.
.js and .jse	Script file written in JScript.
.wsh	Script control file. This type of file controls the execution of a script.
.wsf	Windows script file written in XML code.
.wsc	Script component file. This type of file represents a COM component written in script.

Script Files

The .vbs and .vbe files are script files written in the Visual Basic, Scripting Edition language. A .vbe file is an encoded version of the script.

Similarly, the .js and .jse files are scripts written in JScript. A .jse file is an encoded version of the script.

Note: For more information about .vbe files, see Module 6, “Creating Logon Scripts,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Script Control Files

For each individual script that you run, you can record specific settings by using a control file. These files have a .wsh file extension. A .wsh file is a text file that uses a format similar to the format of .ini files. It is created automatically when you set the properties for a supported script file in Windows Explorer.

► To create a .wsh file for your script

1. Open Windows Explorer.
2. Right-click the script file.
3. On the context menu, click **Properties**.
4. On the Properties page, click the **Script** tab, select the settings you want for the script, and then click **OK** or **Apply**.

This creates a .wsh file with the same name as the script file and in the same folder as the script file.

Note: The way that the .wsh file works resembles how shortcuts (.lnk) work in Windows. It is a pointer to the file that you want to execute. It also configures the environment settings that the script will inherit.

The following is an example of a simple .wsh file.

```
[ScriptFile]
Path=C:\Scripting\Reference Scripts>ShowVar.VBS
[Options]
Timeout=20
DisplayLogo=0
```

In the above example, the Path setting identifies the script file that this .wsh file executes. The Timeout setting sets a maximum script execution time of 20 seconds. The DisplayLogo setting is set to 0, which means that a run-time logo will not be displayed.

Windows Script Files

A .wsf file is a specific type of script file that uses XML code to enhance the features available to script. This code uses tags similar to those in HTML but incorporates several features that offer more scripting flexibility.

Note: Before the release of WSH 2.0, the .ws extension was used for XML format script files and is still referred to by many reference books.

Support for Multiple Script Engines

You can use multiple script languages in a single .wsf file. The following example shows a .wsf file that includes script written in Visual Basic, Scripting Edition and JScript.

```
<job id="WSFDemo">
<script language="VBScript">
```

```
WScript.Echo "This came from VBScript"
</script>
<script language="JScript">
    var strTxt1;
    var strTxt2;
    strTxt1 = "This came from JScript";
    strTxt2 = "XML is very Powerful! ";
    WScript.Echo (strTxt1 + "\n" + strTxt2);
</Script>
</Job>
```

WSH is extensible using third-party ActiveX scripting engines. As a result, .wsf files are not restricted to Visual Basic, Scripting Edition or JScript. You can use other script languages such as PerlScript.

Multiple Jobs in a Single Script

Rather than keep related scripts in separate files, you can incorporate multiple files into a single .wsf file. Enclosing each script in an XML tag defines the scripts in a file as a collection of jobs, as follows.

```
<Package>
    <Job id="JobName1">
        'First Script Goes Here
    </Job>
    <Job id="JobName2">
        'Second Script Goes Here
    </Job>
</Package>
```

Note: The `<Package>` element is optional when a .wsf file contains only one job.

You must identify each job by using a unique job identifier. To run that portion of script, you can then reference these identifiers by using a command similar to the following.

```
CScript //Job:JobName1 Allmyscripts.wsf
```

In this example, **JobName1** is the name, or identifier, of the job contained in the Allmyscripts.wsf file.

Support for Including Other Script Files

If an existing script already provides a function that is required, a .wsf file enables you to use it without duplicating the script. This approach is similar to using **include** statements found in other environments such as Microsoft Visual C++® and Active Server Pages (ASP pages).

A .wsf file encapsulates a library of functions that multiple .wsf files can use. The following example shows the contents of a .wsf file that includes a JScript file (fso.js). It also includes a Visual Basic, Scripting Edition function that calls the **GetFreeSpace** function in the included JScript file, as follows.

```
<Job id="IncludeExample">
<script language="JScript" src="FSO.JS"/>
<script language="VBScript"> ' Get the free space for drive C.
s = GetFreeSpace("c:")
WScript.Echo s
</Script>
</Job>
```

The contents of fso.js is as follows.

```
function GetFreeSpace(drvPath) {
    var fs, d, s;
    fs = new ActiveXObject("Scripting.FileSystemObject");
    d = fs.GetDrive(fs.GetDirectoryName(drvPath));
    s = "Drive " + drvPath + " - ";
    s += d.VolumeName;
    s += " Free Space: " + d.FreeSpace/1024 + " Kbytes";
    return s;
}
```

Windows Script Component Files

A .wsc file is also implemented in XML. It registers the contents of the script file as a COM component. This technology was formerly known as Scriptlets and is often used to extend the functionality of Web servers. The XML format that is required to register your code as a COM component is very strict and beyond the scope of this course.

Lesson 3: Running Scripts

- **Running Scripts by using WScript.exe**
- **Running Scripts by using CScript.exe**
- **Script Languages**

The execution of scripts is controlled by the environment in which they run. This lesson examines the two hosts available to WSH: WScript.exe and CScript.exe. You can use both hosts to run your scripts, but there are differences between the two. These differences may affect which host you choose to run your scripts. This lesson describes the similarities and differences of these two environments, and the Microsoft script languages that you can use with these hosts.

Objectives

After completing this lesson, you will be able to:

- Run scripts by using WScript.exe.
- Run scripts by using CScript.exe.
- Describe the WSH script languages.

Running Scripts by Using WScript.exe

- **Running Scripts from Windows**
 - Double-click files or icons
 - Use the Windows Run command
 - Use a file drag-and-drop operation
 - Create WSH file
- **Wscript.exe is the Default Host**

One of the hosts you can use to run your scripts is WScript.exe.

Running Scripts from Windows

You can run scripts by using WScript.exe in the following ways:

- Double-click the script files in Windows Explorer.
By default, double-clicking any script file will cause WScript.exe to run the script.
- Use the Windows **Run** command.
If you use the **Run** command from the **Start** menu, WScript.exe runs the script.
- Drag the script file to WScript.exe.
You can drag a script file to WScript.exe, or you can create a shortcut to it.
- Create a WSH file.
You can create a WSH file that can then be used to run the script file.

WScript.exe Is the Default Host

All of the methods described previously of using WScript.exe to run your scripts are presented on the assumption that WScript.exe is the default script host. If you change the default to CScript.exe, all of these methods will run the scripts in CScript.exe.

Caution: Visual Basic, Scripting Edition–based viruses can be a major problem if the user does not fully understand this behavior. For more information about the steps that you can take to prevent these viruses, see Module 6, “Creating Logon Scripts,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Running Scripts by Using CScript.exe

- **CScript.exe**
 - Use the Run dialog box or command prompt
 - Drag the file
- **CScript Parameters**
 - // modifies host settings
 - / passes data to the script itself
- **Changing the Default Script Host**
 - //h:CScript or //h:Wscript
- **Saving WSH Settings**

CScript.exe is the command-line version of WScript.exe. Similar to WScript.exe, it supports script execution by using the Windows **Run** command or by using the drag-and-drop operation. However, CScript is not the default script host. If you want to use CScript to run your scripts, then you must either explicitly call CScript or set it as the default script host.

CScript.exe

To use CScript.exe explicitly, you can type a command line at the command prompt or in the **Run** dialog box by using the following syntax.

```
CScript [host options...][script name][options and parameters]
```

Note: Windows Vista uses User Account Control (UAC) so that, by default, applications do not run using Administrator-level credentials even when you are logged on using an administrator account. However, some scripts require administrator credentials. To start a command prompt using Administrator credentials, press the Windows logo key, type **cmd** and then press **CTRL+SHIFT+ENTER**. Finally, press **ALT+C** to confirm the elevation prompt.

The terms are defined as follows:

- Host options

Host options enable or disable various WSH features. The following example stops the WSH from displaying its logo when it runs the MyScript.vbs script.

```
CScript //nologo myscript.vbs
```

- Script name

The script name is the name of the script file, complete with extension and any necessary path information. The following example runs the script file Qbackup.vbs in the C:\Scripts folder.

```
CScript C:\Scripts\Qbackup.vbs
```

- Script options and parameters

Script options and parameters are parameters, or arguments, that are to be passed to the script itself. The following example passes the value C: as an argument to the Myscript.vbs script. The script can then use the parameter to display some information about drive C.

```
CScript myscript.vbs /C:
```

Be aware of the difference between using a single forward slash and a double forward slash to pass parameters at the command line. The former indicates data that is to be passed into the script, and the latter is used as a setting for the CScript environment.

CScript Parameters

Each parameter is optional. If you type **CScript** at the command prompt, the CScript syntax and the valid host parameters are displayed.

CScript.exe supports the host parameters listed in the following table.

Parameter	Description
//?	Shows command usage.
//I	Interactive mode. Displays the user prompts and script errors. This is the default.
//B	Batch mode. Suppresses command-line display of user prompts and script errors. This is the opposite of //I.
//T:nn	Enables time-out option. This is the maximum number of seconds the script can run before it is forcibly terminated by WSH, as specified by nn. The default is unlimited.
//LOGO	Displays a banner. This is the default setting.
//NOLOGO	This switch prevents the banner from appearing at run time.
//H:CScript	Registers CScript.exe as the default host for running scripts.
//H:WScript	Registers WScript.exe as the default host for running scripts.
//S	Saves the current command-line options, for logo and time-out, for this user.
//E:engine	Executes the script with the specified scripting engine. This enables script to be run even if it has a different file extension.
//JOB:<JobID>	Runs the specified JobID from the .wsf file (where JobID is...).
//U	Enables you to use Unicode for redirected input/output (I/O) from the console.

Note: The host parameters described in the previous table are not case sensitive.

Changing the Default Script Host

You can change the default host for a user by using the command-line options of WSH, as the following example shows.

```
CScript.exe //H:cscript
```

This changes the default host to CScript.exe. This setting is automatically saved for the current user and then becomes the default for this user, until the next time the //H: option is used.

Saving WSH Settings

You can use the //S option to save the logo and time-out settings for the user who is currently logged on, as the following example shows.

```
CScript.exe //NOLOGO //T:15 //S
```

This changes the default settings to prevent a banner appearing at run time, and to enable a time-out of 15 seconds. Using the //S option, these settings are saved for the current user and become the default settings every time the user uses WSH in the future.

Note: The saved logo and time-out settings are stored in the registry at the following location: HKEY_CURRENT_USER\Software\Microsoft\Windows Script Host\Settings.

Script Languages

- **Choosing a Script Language**
- **VBScript**
 - Use VBScript if you have no programming experience, or if you are familiar with Visual Basic or Visual Basic for Applications
- **JScript**
 - Use JScript if you are familiar with C, Visual C++, or Java

Two different languages are natively supported by WSH: Visual Basic, Scripting Edition and JScript.

Choosing a Script Language

Although both languages have similar capabilities, you might find that your background makes learning one of the languages easier than learning the other language.

VBScript

If you are new to programming, then Visual Basic, Scripting Edition is easier to learn. Visual Basic, Scripting Edition is also easier to learn if you have a background in programming using Visual Basic or Visual Basic for Applications.

JScript

If you have a background in C, Visual C++, or Java, then JScript is more appropriate. You will find that it has more familiar programming constructs than Visual Basic, Scripting Edition.

With each update of the scripting engines, Microsoft is bringing the feature sets of the languages closer together. Also, by using Windows Script Files (WSFs), you can use the language that is best suited to a particular task, even within the same script. You can develop scripts that have a mix of languages contained within them. You can write one section in Visual Basic, Scripting Edition, and write another section in JScript, all within the same script file.

Note: There are a few areas in which one scripting language is more powerful than the other. For example, JScript has comprehensive error-handling routines, whereas VBScript has excellent run-time expression evaluation. Detailed comparison of the two languages is outside the scope of this course.

For more information about the differences between VBScript and JScript, see the MSDN Scripting Web site.

Lesson 4: Working with Scripts

- Writing Scripts
- Debugging Scripts
- Troubleshooting

This lesson describes the process of writing scripts. To create scripts, you must have a development tool to write the script and a host in which to run the script. There are a number of development tools available.

Objectives

After completing this lesson, you will be able to:

- Write a Windows script.
- Debug a Windows script.
- Troubleshoot Windows script operation.

Writing Scripts

- **WSH 5.6**
 - Part of Windows Server 2003 and Windows XP
 - Download for previous versions of Windows
- **WSH 5.7**
 - Part of Windows Vista
- **Development Tools**
 - Notepad
 - Third-party IDEs
 - Visual Studio
- **Reference Documentation**

Before you can successfully write and run your scripts, you must make sure that the following two basic requirements are met:

- WSH must be present on the computer on which you intend to run the script.
- You must have a suitable script-creation environment for the development of your scripts.

WSH 5.6 and WSH 5.7

WSH 5.6 ships with the following versions of Windows:

- Windows Server 2003
- Windows XP

WSH 5.6 is also available for download for the following versions of Windows:

- Windows 2000 Professional and Server
- Windows NT 4.0
- Windows 9.x

The components of the WSH 5.6 download are:

- Visual Basic, Scripting Edition version 5.6
- JScript version 5.6
- Windows Script Components
- Windows Script Host 5.6
- Windows Script Runtime Version 5.6

WSH 5.7 ships with Windows Vista. Windows Vista includes the following WSH components:

- Visual Basic, Scripting Edition version 5.7
- JScript version 5.7
- Windows Script Components
- Windows Script Host 5.7
- Windows Script Runtime Version 5.7

Note: Updates to the script engines also ship with Internet Explorer updates. For example, version 5.7 of the VBScript and JScript engines ships with version 7.0 of Internet Explorer. Check the documentation of future updates of Internet Explorer to determine which script engine version they contain.

Development Tools

WSH does not ship with a dedicated development tool for scripting. However, you can obtain tools that enable you to develop scripts on your Windows platform.

The most basic development tool is Notepad.exe. Using Notepad as your scripting development tool has the following advantages:

- It is installed on all Windows machines by default.
- It supports cut-and-paste and other text-editing operations.
- It has low memory and processor requirements.

Using Notepad does not impose any specific limitations on the code that you develop. However, this tool provides no specific scripting functionality such as object browsing or integrated debugging.

There are other development tools available from third parties that are far more powerful than Notepad. There are also the Integrated Development Environments (IDEs) of Microsoft Visual Studio® and the Script Editor component of the Microsoft Office System (which is designed for Visual Basic for Applications coding). These programs offer features that include the object browser and mature, integrated debugging tools. However, these tools are not designed exclusively for Visual Basic, Scripting Edition. As a result, there are compatibility and programming issues that you need to understand before you can use these tools to develop scripts. The Microsoft TechNet Script Center provides example scripts, and also free scripting tools such as Scriptomatic, which can be helpful when you are creating scripts that need to use WMI or ADSI interfaces.

Note: For more information about Scriptomatic, see Module 8, “Understanding WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Reference Documentation

It is a good idea to read the reference documentation for Visual Basic, Scripting Edition and WSH, which is available on the Web. These documents provide an invaluable guide to the objects, methods, and properties that are available in Visual Basic, Scripting Edition and WSH, as well as many examples. The Visual Basic, Scripting Edition documentation also provides a reference to the Visual Basic, Scripting Edition language syntax. The WSH and Visual Basic, Scripting Edition documentation is available from the Microsoft TechNet Script Center Web site.

Debugging Scripts

- **Script Errors**
 - Syntax errors
 - Run-time errors
 - Logic errors
- **Microsoft Script Debugger**
 - Features of Microsoft Script Debugger

Even the most experienced programmers will create bugs and errors in their code. Your scripts will also be vulnerable to errors. Finding and fixing errors is an essential day-to-day task that you must master to develop robust, useful scripts.

Script Errors

You can categorize errors in your script into three groups:

- **Syntax errors**

A syntax error occurs when you have not followed all of the rules of the language. For example, omitting a quotation mark where one is required will generate a syntax error. Syntax errors are picked up as the script is compiled and executed. For this reason they are sometimes called compilation errors.

Compilation is the process of turning the code that you write into a set of instructions that the processor can execute. Syntax errors prevent your code from running. You will receive a message box that indicates the type of syntax error that has occurred.

- **Run-time errors**

A run-time error occurs when your code, although syntactically correct, attempts to perform an action that is not possible at the time it is attempted. For example, a run-time error occurs if your script attempts to open a file that does not exist or attempts to save a file to the floppy drive when no disk is present. Run-time errors cause your script to stop unexpectedly. You will receive a message box that indicates the type of action that was attempted but was not possible.

- **Logic errors**

A logic error occurs when a script runs without syntax or run-time errors, but the results are unexpected or unintended. For example, a script may prompt the user for a password. The script is supposed to prevent the user from proceeding without a valid password, but it allows the user to process without a password. This could be due to a logic error in the script.

Logic errors are typically the most difficult ones to fix, because they are the most difficult ones to identify. Logic errors usually result in bugs in your script.

You can use Microsoft Script Debugger to run your code line-by-line to see why the script does not run the script the way it was intended. This is the most efficient approach to finding and fixing logic errors.

Microsoft Script Debugger

The Microsoft Script Debugger is available as a download for Windows XP and Windows Server 2003. This tool is designed to help you to find problems or bugs with the execution of scripts.

Note: Microsoft Script Debugger is deprecated in Windows Vista and Microsoft offers no support for the tool on this version of the operating system.

Features of Microsoft Script Debugger

Some of the features of Microsoft Script Debugger include the ability to run script commands directly, set breakpoints in your code, and view a list of procedures whose execution is pending.

Note: The script debugger documentation is written with the focus on Internet scripts rather than WSH. While this can be confusing for an administrator who has no Internet Explorer script experience, the majority of the information it provides can be applied to WSH scripting.

For information about how to use the features of Microsoft Script Debugger, see Module 4, “Error Handling and Debugging,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Troubleshooting

- **WScript.Echo Method**
 - Generates a message box in WScript
 - Outputs to the command line in CScript
- **The Debugger Launches Without a Loaded Script**
 - From the command prompt, check which line caused the failure
 - Log off and log back on to reset the debugger

When you are working with scripts, you must be aware of some basic troubleshooting information.

WScript.Echo Method

The following statement behaves differently in WScript and CScript.

```
WScript.Echo "This is a message"
```

With WScript.exe, this method generates a message box that interrupts the execution of the script until the user clicks OK.

With CScript.exe, this method outputs the text directly to the command line and then continues to execute the script. No user interaction is required.

The Debugger Launches Without a Loaded Script

If the debugger launches without a loaded script, it is usually due to a syntax error in your script. If this is the case, you can switch to the command prompt, where an error message will be displayed. The error message will highlight the line and character at which the error was found.

If this fails to help, and the script window is still missing, log off and log on again to clear the problem.

Lab: Configuring and Using WSH



- Exercise 1
Configuring WSH
- Exercise 2
Reviewing the WSH and VBScript Documentation
- Exercise 3
Using Windows Script Files
- Exercise 4
Using an IDE to Edit Script Files

After completing this lab, you will be able to:

- Manage the WSH environment on a computer, run scripts, and configure WSH files.
- Navigate the WSH and Visual Basic, Scripting Edition documentation.
- Use WSFs.
- Edit scripts by using an IDE.

Estimated time to complete this lab: 45 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine.
- Start the 2433B-VISTA-CL1-01 virtual machine.
- Log on to the 2433B-VISTA-CL1-01 virtual machine as **FOURTHCOFFEE\Administrator** using the password **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to run scripts written in Visual Basic, Scripting Edition by using WSH technologies. You must control and configure the script host that you will use to run your scripts, and make use of .wsh and .wsf files. In addition, you want to review the documentation for WSH and Visual Basic, Scripting Edition. Finally, you want to review and edit scripts by using an IDE.

Exercise 1

Configuring WSH

In this exercise, you will examine and configure the various settings for the WSH environment.

The principal tasks for this exercise are as follows:

- To list the script settings for the CScript.exe host.
- To run a script file.
- To change the default script host.
- To create a WSH file for Scr1.vbs.
- To open the WSH file for Scr1.vbs.
- To modify the WSH file for Scr1.vbs.

Tasks	Supporting information
1. To list the script settings for the CScript.exe host.	<ul style="list-style-type: none"> • Open a command prompt. • Run CScript and view the options.
2. To run a script file.	<ul style="list-style-type: none"> • At the command prompt, change to the E:\Labfiles\Starter folder. • Run the Scr1.vbs script using the default host. • Run the Scr1.vbs script using the CScript host.
3. To change the default script host.	<ul style="list-style-type: none"> • At the command prompt, change the default script host to CScript. • Run Scr1.vbs to verify the change of default host.
4. To create a WSH file for Scr1.vbs.	<ul style="list-style-type: none"> • Use Windows Explorer to view the contents of the E:\Labfiles\Starter folder. • View the properties of Scr1.vbs. • Select to stop script execution after 1 second. • View the contents of the E:\Labfiles\Starter folder to verify the creation of Scr1.wsh.
5. To open the WSH file for Scr1.vbs.	<ul style="list-style-type: none"> • Open Scr1.wsh in Notepad.
6. To modify the WSH file for Scr1.vbs.	<ul style="list-style-type: none"> • Modify Scr1.wsh to stop script execution after five seconds. • At the command prompt, run Scr1.wsh to verify the time-out.

Questions

Q: What are the script host executable files?

Q: Which host is the default host?

MCT USE ONLY. STUDENT USE PROHIBITED

Exercise 2

Reviewing the WSH and VBScript Documentation

In this exercise, you will review the WSH and Visual Basic, Scripting Edition documentation.

The principal tasks for this exercise are:

- To review the WSH documentation.
- To review the VBScript documentation.
- To search the Windows Scripting Technologies documentation.

Tasks	Supporting information
1. To review the WSH documentation.	<ul style="list-style-type: none">• On the desktop, open the Windows Script 5.6 Documentation help file.• Locate information about WSH.
2. To review the VBScript documentation.	<ul style="list-style-type: none">• Use the Windows Scripting Technologies help file to locate information about VBScript.
3. To search the Windows Scripting Technologies documentation.	<ul style="list-style-type: none">• Use the Windows Scripting Technologies help file to find information about the Echo method.

Questions

Q: What is the Visual Basic, Scripting Edition equivalent of the **Debug.Print** command in Visual Basic?

Q: What is the main difference between Visual Basic, Scripting Edition and WSH?

Exercise 3

Using Windows Script Files

In this exercise, you will use a .wsf file and explore some of the features of the XML format.

The principal tasks for this exercise are:

- Use WSF files.

Tasks	Supporting information
1. Use WSF files.	<ul style="list-style-type: none">• Use Windows Explorer to view the contents of the E:\Labfiles\Starter folder.• Run WSFIInAction.wsf.• Note the job name.• At the command prompt, run WSFIInAction.wsf specifying Job2.• Note the job name and output.• At a command prompt, run WSFIInAction.wsf specifying Job4.• Note the output.

Questions

Q: Which script engine will the Myscript.jse file invoke?

Q: Which script engine will the Myscript.wsf file invoke?

Exercise 4

Using an IDE to Edit Script Files

In this exercise, you will use the PrimalScript IDE.

The principal tasks for this exercise are:

- To prepare the PrimalScript IDE.
- To open a script file in PrimalScript.
- To use the auto-complete features of the IDE.
- To save and run the script.

Tasks	Supporting information
1. To prepare the PrimalScript IDE.	<ul style="list-style-type: none"> • Start PrimalScript 4.1 Professional. • In the Welcome dialog box, select VBScript System/Network Administrator as your role. • Complete the wizard, leaving all other settings at their defaults. • Close the Left Nexus and Right Nexus windows. • On the Tools menu, click Options, expand Environment, and then select Directories. • Configure your My Scripts directory to be E:\Labfiles.
2. To open a script file in PrimalScript.	<ul style="list-style-type: none"> • Open E:\Labfiles\Starter\Scr1.vbs.
3. To use the auto-complete features of the IDE.	<ul style="list-style-type: none"> • Add a new line to the end of Scr1.vbs. • Type WScript then a single period. • From the auto-complete list, double-click Echo. • Press SPACEBAR and note the on-screen help. • Type "Welcome to the VBScript and WSH course!" (including the quotation marks).
4. To save and run the script.	<ul style="list-style-type: none"> • Save Scr1.vbs. • Run the script, using the PrimalScript Run Script option. • Note the results in the Output window. • At the command prompt, run Scr1.vbs using the CScript host. • At the command prompt, run Scr1.vbs using the WScript host.

Questions

Q: What are some advantages of using Notepad as your script editor?

Q: What are some advantages to using an IDE to develop scripts?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

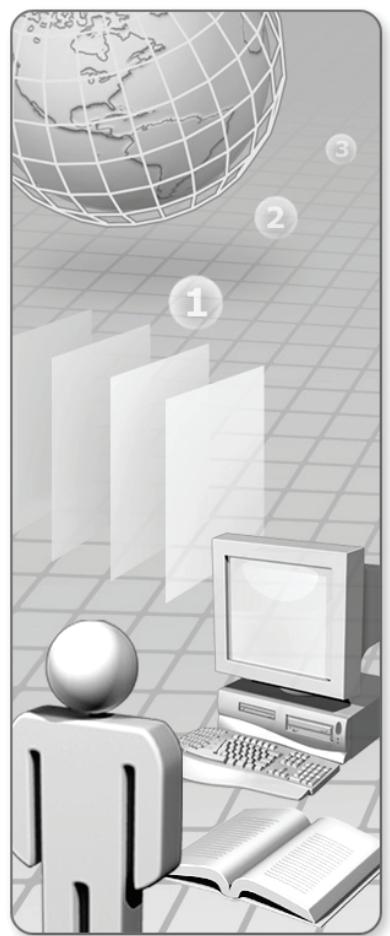
After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-01 virtual machines and discard any changes.

Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

Module 2: Objects in VBScript and WSH

Table of Contents

Module Overview	2-1
Lesson 1: Object Terminology	2-2
Lesson 2: Creating and Manipulating Objects	2-11
Lesson 3: Object Models	2-16
Lesson 4: Common Object Models	2-30
Lab: Objects in VBScript and WSH	2-44



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- Object Terminology
- Creating and Manipulating Objects
- Object Models
- Common Object Models

To write administrative scripts, you must understand what objects are. You must also know how to access objects and the functionality that they provide by using Microsoft® Visual Basic®, Scripting Edition (VBScript) and Windows® Script Host (WSH). This module will teach you about object models and objects, and how to interact with objects in your scripts.

Objectives

After completing this module, you will be able to:

- Describe how scripts use objects.
- Describe object terminology.
- Explain how to use an object browser.
- Describe how scripts interact with the Component Object Model (COM).
- Explain the use of various object models.

Lesson 1: Object Terminology

- **Object Terminology**
- **Methods and Properties**
- **Instantiation Terminology**

Using objects in your scripts gives you many advantages such as efficiency, reliability, and reuse of existing functionality. Before you can use objects in your scripts, you must understand the terms and concepts that apply to programming with objects.

This lesson describes the terminology used when programming with objects. It also includes an analogy that helps to summarize the key concepts that this lesson introduces.

Objectives

After completing this lesson, you will be able to:

- Use the terminology associated with objects.
- Discuss what methods and properties are and what they do.
- Describe the terminology associated with the creation of objects.

Object Terminology Definitions

- **Objects**
 - Packaged functionality
- **Compiled Objects**

You will encounter some terminology when you start to use objects in your scripts. Each term has a distinct meaning. You will come across these terms frequently. Each term, and how it relates to the others, is described below.

Objects

Objects are packaged pieces of functionality. Objects contain code that you can use in your scripts. The way that you access the functionality that an object contains is called black-box reuse. When you use the functionality that an object provides, you never need to see the code that it contains.

Compiled Objects

The most common type of object that you will use in your VBScript files is compiled objects. With this type of object, you will never need to see the code that it contains. The code that provides the functionality is compiled into a binary file, which is usually a dynamic-link library (DLL) or a Microsoft ActiveX® control file (with the file extension .ocx). All that you must do is to create script in Visual Basic, Scripting Edition that communicates with the object to reuse the prepackaged functionality that it provides.

In addition to containing code that provides reusable functionality, objects also contain data. You can manipulate this data in a manner similar to reusing the packaged functionality. You do not need to know how the object implements its data. You only need to know how to communicate with the object to manipulate the data values. This is the central tenet of the black-box reuse concept.

You communicate with the functionality that an object provides by using its methods. You manipulate the data that an object contains by using its properties. Methods and properties are described in the following section.

Note: Each object that you use has a well-defined list of methods and properties associated with it. For example, one object may contain a **Color** property and a **Paint** method, while another object may contain a **Name** property and a **Play** method. The methods and properties that each individual object contains are designed by the developer of that object.

Methods and Properties

- **Methods**
- **Invoking a method**
- **Using parentheses**
- **Properties**

To manipulate objects, you work with their methods and properties.

Methods

Objects expose their functionality as methods. To use the functionality provided by a particular method, you must know how to call it. You do not need to know how the method works internally. The term “invoke” is often used to describe the process of calling a method.

When you invoke a method in your script, the compiled method code runs in its entirety at the point in your code where you call it. It is important to understand the benefit to you, as a script developer, of invoking a method in this way. The method may contain thousands of lines of code hidden from you by the object. You can simply access this code by calling the method in a single line of script. The approach of hiding complex code is often termed encapsulation.

Invoking a Method

There are three different ways to invoke a method. The approach that you use in your script depends upon how the method is exposed. The three approaches are described below.

`Object.Method`

In this approach, you invoke the method with a simple call. The call uses the dot notation to specify the name of the object and the name of the method that you want to invoke. In this simple call, the method does not require any inputs from your script, nor does it return data to your script.

`Object.Method Param1 [, Param2, ... ,Paramn]`

In this approach, you invoke the method with a call similar to the previous call. Again, the call uses the dot notation to specify the name of the object and the name of the method that you want to invoke. However, this approach passes data into the method from your script by using one or more parameters. The method accepts the data and works with it internally, encapsulated from your script. The parameters are provided as a comma-separated list.

```
MyVariable = Object.Method([Param1, Param2, ... ,Paramn])
```

In this approach, a variable in Visual Basic Scripting Edition is used to hold the return value of the method call. Again, the dot notation is used, but this time the value of the variable is set to the result of the method call.

Note: For more information about variables, see *Creating and Manipulating Objects* later in this module, and see Module 3, “Script Logic,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Using Parentheses

An important addition to this syntax is the inclusion of the parentheses that follow the method call. The parentheses indicate that your script must evaluate the method call. In this case, they hold the evaluated data in the variable.

Methods that return data to the calling script may require parameters. If this is the case, then you must supply those parameters as a comma-separated list inside the parentheses, as shown in the previous syntax. However, some methods may return data and not require parameters. If this is the case, then you must still use the parentheses, even though the code does not contain any parameters. The following syntax illustrates this.

```
MyVariable = Object.Method()
```

Properties

Objects expose their data to your script in their properties. To manipulate this data, you must know how to retrieve and set the property value. You do not need to know how the object stores or handles the data. Again, this reinforces the black-box reuse concept.

The following syntax illustrates some standard approaches to setting and retrieving the values of an object’s properties.

```
Object.Property = Value
```

In this syntax, using the dot notation sets the property’s value. The value may take one of a number of forms, but you must evaluate it as a discrete value. Some of the forms that the value takes are described as follows:

- A literal value.

You can set the property of an object to a literal value such as the number 10 or the string “Cat.”

- A simple expression that can be evaluated as a discrete value.

You can set the property of an object to the result of an expression that your script can evaluate as it runs, such as the result of $18 + 9$ or the result “London” concatenated with “United Kingdom.”

- A property of another object.

You can set the property of one object to the property of another object. You can achieve this in the following single line of code.

```
Object1.Property1 = Object2.Property2
```

- The return value of an object’s method call

You can set the property of one object to the return value of another object’s method. You can achieve this in the following single line of code.

```
Object1.Property1 = Object2.Method([Param1])
```

If you want to retrieve the property value and store it in one of your variables, use the following code. The syntax is very similar to that used to retrieve the return value of a method. However, in this case, you do not use parentheses.

```
MyVariable = Object.Property
```

Instantiation Terminology

- **Classes**
 - Blueprints for your objects
- **An Analogy—Architect Drawings and Buildings**
- **Type Libraries**
 - A collection of classes
- **Instantiation**
 - Multiple instances

To use objects and their corresponding methods and properties, you must write code that creates the object. Object creation involves referring to the object's type library and class. These terms are explained below.

Classes

A class is the blueprint for an object. The developers who allow you to reuse their code as objects do not write the code in the object itself. They write code in a class. When you want to reuse code, your script creates an object that is based on a class. The object code is written in the class in such a way that it gives you the methods and properties for your object.

An Analogy—Architect Drawings and Buildings

One way to think about the purpose of a class is to consider a real-world analogy. Imagine the process involved in planning and constructing a building: Architects have a vision of what they want a building to look like and how they want the building to be structured. To convey their ideas to the builders, they create architect drawings, also called blueprints, in which they specify the materials to be used and the dimensions of the building.

At this point, their job is done. They can hand the blueprint to the builders, who construct the building according to the plans. In fact, the builders can construct many similar buildings, all based on the same drawing. They can reuse the blueprint many times. After the buildings have been constructed, the owners can decide to paint the buildings in any color they choose. They may also decide to install alarms and add furniture.

In this analogy, an architect's blueprint is analogous to a class. Objects are analogous to the buildings that are created from the blueprint.

The blueprint makes allowances that alarms can be installed and furniture can be added. These actions are analogous to methods. The object is manipulated after it has been created by performing these actions.

Furthermore, the blueprint of the building does not specify a fixed color for the paint used to decorate it. The blueprint allows this color to be chosen after the building has been constructed. The paint color is analogous to a property.

Type Libraries

A type library is a collection of classes that can be redistributed to enable you to build objects based on those classes. Type libraries are compiled files that expose their classes to your script. They are usually compiled into DLL or ActiveX control files.

A Collection of Classes

Using the earlier analogy of a blueprint, architects may produce many blueprints for various building types. They may develop one blueprint for a house, one for an apartment building, and one for a skyscraper office complex. They may then store all of these blueprints in a folder that is named Buildings.

When they complete these blueprints, they can give the whole folder to the builders. When the builders want to build a house, they can get the house blueprint from the folder. They could also access the skyscraper or apartment blueprints to construct those types of buildings too.

In this analogy, the folder is the type library. It contains the blueprints for various types of buildings. Each building type has its own set of distinct features and attributes. In object terminology, the methods and properties are these distinct features and attributes.

You must know the name of any type library that you use in your scripts.

Note: For examples of the names of type libraries, see the lab that appears at the end of this module.

Instantiation

The process of creating an object from its class is termed instantiation. When an object is instantiated, it inherits the properties and methods that the class defines. This functionality is held in your computer's memory. As a result, it is efficient for you to manipulate the object.

After the object is instantiated, it no longer must refer back to its class, nor is it desirable that it should do so. After you have created the object, you can manipulate it without affecting the class upon which the object was based.

Using the analogy of the blueprint, imagine that the builders have constructed a house and that the decorator paints it yellow. It is sensible that this action should affect only that house and should not affect the blueprint. If it did affect the blueprint, then all of the new houses would be yellow, which was not the original plan.

Multiple Instances

You will often use multiple instances of the same class in your script. If you create multiple instances of the same class, then you can manipulate each instance, or object, independently of the others.

Using the above analogy, the builders may construct a whole street of houses from the same blueprint. The decorator may then paint one house yellow. This will not affect the other houses, even though they are based on the same blueprint. The decorator may paint the next house green, and so on.

Note: Objects are usually instantiated with a single line of code written in Visual Basic, Scripting Edition. For more information about how to write this instantiation code, see Creating and Manipulating Objects later in this module.

Lesson 2: Creating and Manipulating Objects

- **Creating Objects**
- **Manipulating Objects**

Creating objects and using their methods and properties is the foundation of scripting with objects. This lesson explains how to create and use objects in your scripts.

Objectives

After completing this lesson, you will be able to:

- Create objects using script.
- Manipulate objects using script.

Creating Objects

The diagram shows a slide with a blue header bar. The main content area has a light gray background. A vertical line of bullet points on the left side lists 'Libraries', 'Classes', and 'Example of Object Usage'. To the right of each point is a small blue circular icon. Below this list is a code block containing VBScript code to map a network drive.

```
Dim oNetwork
Set oNetwork = CreateObject("WScript.Network")
oNetwork.MapNetworkDrive "z:", "\\\LON-DC1\LabShare"
```

Administrative scripts commonly use the WSH library.

Libraries

The WSH library is distributed in the Wshom.ocx file, and installs by default on all computers running Windows Vista® and Windows Server® 2003 operating systems.

You can access this type library in a script that uses the following name.

WScript

Classes

The WScript type library contains multiple classes that you use in your scripts. An example is the **Network** object. This enables a script to perform simple network tasks. The object is instantiated using the following syntax.

```
Set <Object Variable> = CreateObject("<Library>.<Class>")
```

Example of Object Usage Example

An example of how to create the **Network** object is as follows.

```
Set oNetwork = CreateObject("WScript.Network")
```

The above syntax uses the **CreateObject** function that is part of the Visual Basic, Scripting Edition language. It accepts a string parameter that specifies the type library and the class to be used to create the object. An object variable is set to the result of this function so that you can then use the variable to refer to the newly created object in subsequent lines of script.

After this object has been instantiated, you can use the object to perform network tasks such as mapping a network drive or printer. You can achieve this by using methods such

as **MapNetworkDrive** and **AddPrinterConnection**. The **oNetwork** object variable also exposes properties such as **ComputerName** and **UserName**, which you can use in the script to identify the user running the script and the computer that it is running on. The following example illustrates how you can use these methods and properties.

```
Set oNetwork = CreateObject("WScript.Network")
oNetwork.MapNetworkDrive "Z:", "\\\\"MyServer\\MyShare"
Wscript.Echo oNetwork.ComputerName
```

The first line of code instantiates the new object and sets the **oNetwork** object variable to it.

The next line of code uses the **MapNetworkDrive** method to create a new drive mapping. The parameters supplied indicate that drive Z is to be mapped to the shared folder called MyShare on the server called MyServer.

The third line of script retrieves the value of the **ComputerName** property from the **oNetwork** object variable. In the same line of code, the result of this operation is displayed to the user with the **Echo** method of the built-in **WScript** object.

Note: Although WScript is the name of a type library, there is also an object called **WScript** that exposes useful methods like **Echo**. This object is related to the host itself, so it is not necessary to instantiate it (as you must do with most objects). It is automatically instantiated by the host in which your scripts run and can therefore be used directly.

Manipulating Objects

• Assigning an Object to a Variable

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

• Manipulating an Object

```
objFSO.CreateFolder("C:\Test")
```

• Deleting an Object

```
Set objFSO = Nothing
```

To use the properties and methods of an object, you must refer consistently to your instance of the object. To do this, you create a named pointer and assign it to the instance of the object. The named pointer is known as an object variable and the object variable points to the object code and data in memory.

Assigning an Object to a Variable

You use the set keyword to assign an object to a variable in your code. The set keyword informs the script that the variable points to an object structure in the computer's memory.

The following example instantiates a **FileSystemObject** object and points the **objFSO** variable to it, so that it can be manipulated in subsequent lines of code.

```
Set objFSO = CreateObject ("Scripting.FileSystemObject")
```

Manipulating an Object

After you have created the object in the memory space of the running script, it is possible to access the methods and properties that the object exposes by using the object variable that points to it.

The following example uses the **CreateFolder** method to create a new folder called Test in the root of drive C.

```
objFSO.CreateFolder ("C:\Test")
```

Deleting an Object

Objects consume computer memory. Therefore, it is good practice to delete objects that are no longer required.

To delete an object, you set its object variable to the keyword **Nothing**, as shown in the following code example.

```
Set objFSO = Nothing
```

Note: Objects that are not explicitly set to **Nothing** are deleted automatically by WSH when the script terminates. However, it is considered best practice to delete objects explicitly when the script no longer requires them, because this makes available the computer memory that is used by the object. Complex objects can use a substantial amount of computer memory, so you must delete complex objects as soon as possible.

Note: The techniques outlined in the previous section can be used in many scripting environments, including Windows PowerShell™ command-line interface.

Lesson 3: Object Models

- Defining Object Models
- COM Objects
- COM Objects in Practice
- Custom COM Components
- Viewing Object Models by Using an Object Browser
- Demonstration: Viewing Object Models by Using an Object Browser
- The Scripting Object Model
- The WSH Object Model

An object model describes a collection of objects that are designed to perform a related set of tasks. This lesson examines how object models function and how you can use them when you write scripts. It also provides an introduction to the COM.

In addition, this lesson tells you more about the Microsoft Scripting Runtime object model and the WSH object model. An understanding of these object models increases the range of administrative tasks that you will be able to perform with scripts.

Objectives

After completing this lesson, you will be able to:

- Describe what object models are.
- Describe the fundamentals of the COM.
- Find an object and determine what functionality it offers by using an object browser.
- Use scripts, including writing, debugging and troubleshooting.

Defining Object Models

- **Describing Object Models**
- **Examples of Object Models**
 - WSH
 - ADSI
 - ADO
 - WMI

Many of the objects that you can use in your scripts are presented to you in an object model.

Describing Object Models

An object model is a collection of objects that perform a related set of tasks such as managing the file system or reading information from a database. The developer groups together the object classes, methods, and properties and presents them as an object model.

In many cases, the objects in the object model are organized into a hierarchy. The higher-level, or root, objects provide access to the objects further down. For example, the **FileSystemObject** object provides access to the drive object. It is not possible to call the drive object without using the **FileSystemObject** object.

While many existing object models implement hierarchies that require you to access one object through another object, a more modern approach is to access all objects directly. For example, you can access directly all of the objects in the ActiveX Data Objects (ADO) model. They are not organized into a hierarchy.

Examples of Object Models

Some of the most commonly used object models in administrative scripts are described below:

- WSH
 - This object model gives you functionality that you can use to perform basic administrative tasks such as creating folders and mapping network drives.
- Microsoft Active Directory® Service Interfaces (ADSI)

MCT USE ONLY. STUDENT USE PROHIBITED

This object model gives you functionality that you can use to access and configure Active Directory directory service features such as users and groups.

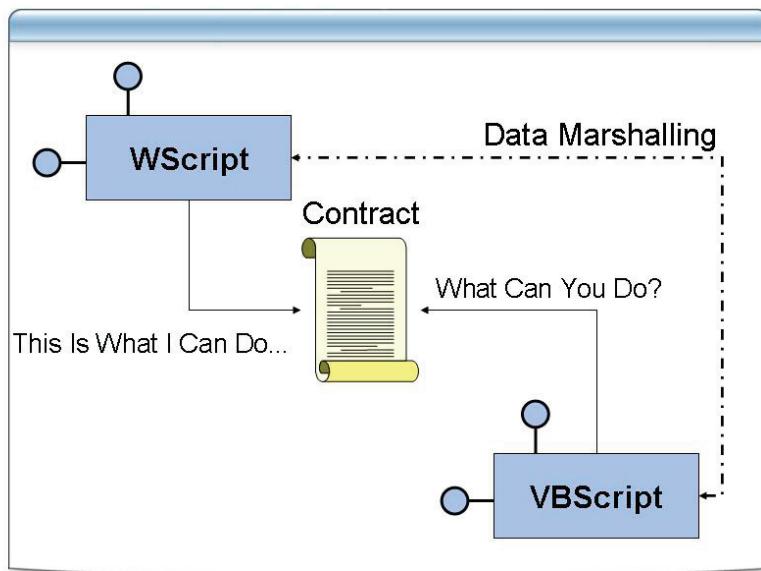
- ADO

This object model gives you functionality that you can use to access data sources such as Microsoft SQL Server™ database software.

- Windows Management Instrumentation (WMI)

This object model gives you functionality that you can use to manage data and operations on Windows operating systems.

COM Objects



The COM provides a specification and a set of rules that define how software systems can be constructed from reusable components. The COM specification also describes the precise way in which client applications, such as WSH scripts, can communicate with objects that are located inside components.

Contracts

A contract consists of a definition of the methods and properties supported by objects. COM enables a contract to be defined and specifies the services that the component provides to its clients. The rules of COM dictate that the contract cannot change after it has been published or made generally available to client software. This alleviates many of the versioning issues in most software systems, particularly those developed before the component era.

COM also governs how data is transferred between the client and object. The process of passing data into and out of an object is called marshalling.

DCOM and Component Services

DCOM extends marshalling support of COM so that objects can be accessed over a network.

Windows Vista and Windows Server 2003 support Component Services provided by COM and COM+ to provide additional functionality to component developers and support for distributed transactions, resource management, and security.

Note: Scripting by using WSH and the other technologies mentioned in this course is separate from the Microsoft .NET Framework and the functionality that it provides. If you want to script against .NET Framework classes, you must install

the .NET Framework and use the languages supported by the common language runtime—C#, VB.NET, JScript.NET, and other languages.

Note: You can script against COM objects from many scripting environments, including Windows PowerShell.

COM Objects in Practice

- **Object Identification**
 - ProgID
- **Automation**
 - Required for script
- **Type Libraries**
 - Defines component's metadata

Classes are identified by using a label called a Programmatic Identifier, or ProgID.

Object Identification

The ProgID is stored in the **HKEY_CLASSES_ROOT** section of the registry. It tells the system which .dll, .exe, or .ocx file contains the associated class definition that is used to construct the object. A ProgID consists of a type library and a class name that are separated by a dot, as in the following example.

Library.Class

The following example uses the **Scripting.FileSystemObject** ProgID to create an instance of the object. The registry contains information about the compiled file that defines this type of object.

```
Set MyObject = CreateObject("Scripting.FileSystemObject")
```

Automation

Scripting environments such as WSH impose certain restrictions on COM objects. One of the key restrictions is the range of data types that can be passed to and from the object. This restriction—and a number of others—means that script code cannot communicate with all COM objects. Those COM objects that comply with the requirements of the script are called Automation objects.

Automation defines a restricted set of data types and imposes one or two other requirements on the developer of COM objects. From the administrator's perspective, this is not a significant issue, because nearly all of the object models that an administrator may want to use from a WSH script are Automation-compliant.

For example, all of the applications in the Microsoft Office System, such as Microsoft Office Word and Microsoft Office PowerPoint®, support Automation. As a result, Visual Basic, Scripting Edition can be used to control their functionality. The following script takes the first slide from an active Office PowerPoint presentation, copies it, and pastes it into an Office Word document called Demo.doc that is located in the C:\My Documents\ folder.

```
Set oPres = CreateObject ("PowerPoint.Application")
oPres.Visible=True
oPres.ActivePresentation.Slides(1).Copy
Set oWord = CreateObject ("Word.Application")
oWord.Visible=True
oWord.Documents.Open "c:\My Documents\Demo.doc"
oWord.Selection.Range.Paste
```

Type Libraries

As mentioned previously, you can use type libraries to define the set of classes, methods, and properties that a component supports. This type of information is the component's metadata, the data that describes data.

Although it is typically saved as part of the component's .dll or .ocx file, it can reside in a stand-alone type library file, usually with a .tlb file extension. By examining the metadata inside a type library, it is possible to discover the objects, methods, and properties that you can use in a script. Object browsers can use this type information to show a graphical view of an object model.

Custom COM Components

- **COM Component Basics**
 - Customized pieces of functionality
 - Use in the same way as standard COM objects
 - Support Automation and handle data types
- **Registering COM Components**
 - Required to access component from scripts

You can script against custom COM components in the same way that you script against other COM objects.

COM Component Basics

Developers can create custom pieces of functionality by developing COM components. You can use the features of any COM component from within a script as long as the script can call the COM component. You create and use COM components in the same way that you script against any other COM object.

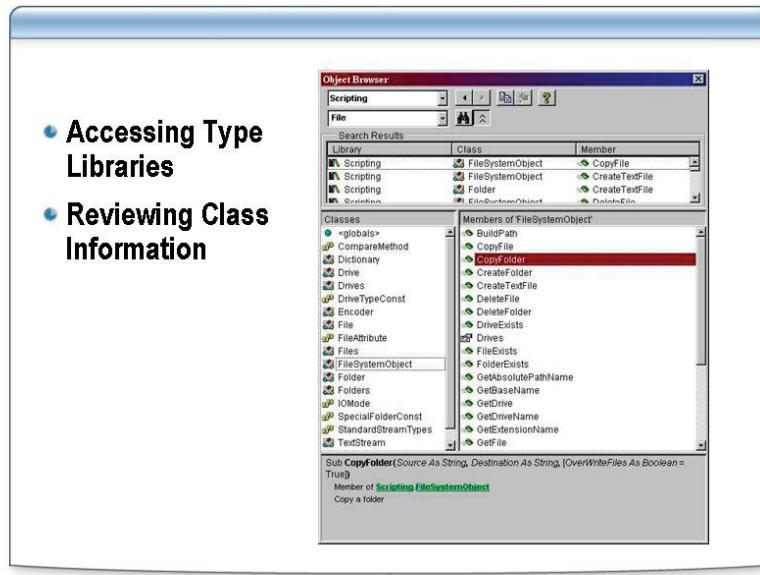
For scripts to use COM components, the COM components must support Automation and correctly handle the script data types. There are many Web sites that offer third-party COM components to help you increase the functionality of scripts.

Registering COM Components

COM components must be registered on the computer on which you will use them. You can do this by using the installation routine of the component itself or by manually using the RegSvr32 utility. For information about using the RegSvr32 utility to register a COM component, see the Microsoft Web site.

As mentioned earlier, the registration process adds the component to the **HKEY_CLASSES_ROOT** key of the registry. This is where the Component Services class registration database is stored. These registry entries are required for the component to be referenced and used.

Viewing Object Models by Using an Object Browser



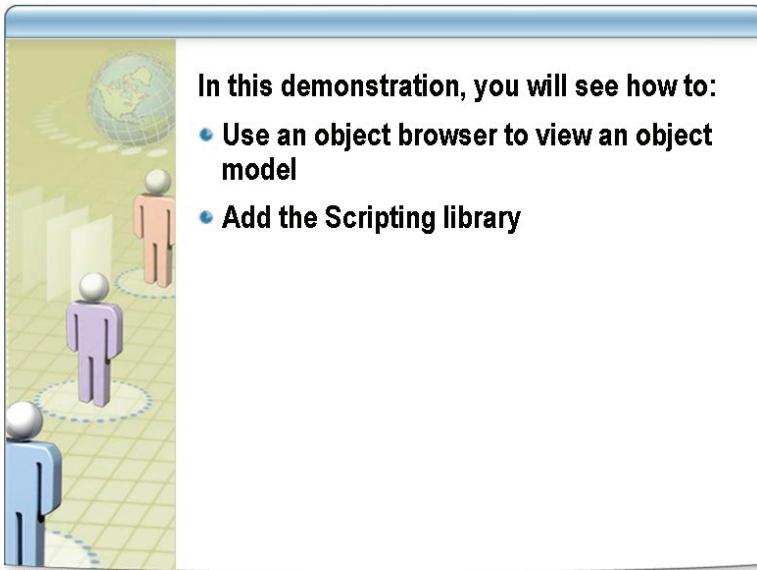
It is helpful to view what objects are available in a type library when you write script. Many tools can give you this functionality.

The Microsoft Object Browser is a tool that you can use to look inside a type library. It displays the information in a readable, graphical form. The object browser ships with Visual Basic for Applications in the Microsoft Office System.

Note: The Object Linking and Embedding (OLE)/COM Object Viewer that ships with Microsoft Visual Studio® development system is another object browser that you can use to determine the ActiveX controls and COM objects available on your systems. It is a developer-oriented tool and will not be used in this course.

You can download it as a stand-alone application from the Microsoft Web site.

Demonstration: Viewing Object Models by Using an Object Browser



In this demonstration, you will see how to:

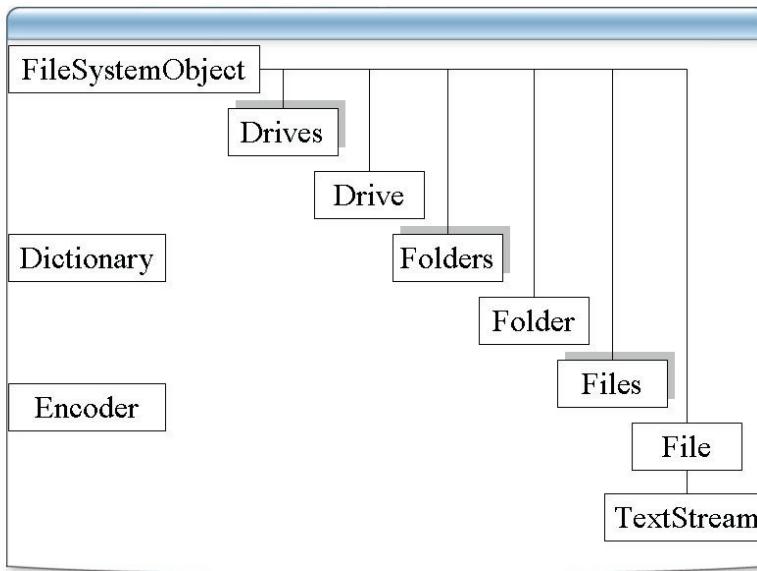
- View an object model by using an object browser.
- Add the Scripting library.

Key Points

The key points of this demonstration are:

- To understand the value of using an object browser.
- To see the number of objects available for you to script against.

The Scripting Object Model



The type information describing the scripting object model, or the Microsoft Scripting Runtime object model, is defined in Scrrun.dll. The primary object that this object model provides is the **FileSystemObject** object. You sometimes use the **Dictionary** object and the **Encoder** object from this object model.

FileSystemObject

You can use the **FileSystemObject** object to manage files and folders on the local computer. In addition to manipulating existing folders and files, the **FileSystemObject** object enables you to create new folders and text files.

You can only create text files by using the **FileSystemObject** object. If you want to create a new Office Word document, you must use the Office Word object model.

The **FileSystemObject** object also does not provide support for managing permissions for NTFS file systems (NTFS). If you must change the security permissions on the file, you must use an extra utility.

Dictionary

The **Dictionary** object is a container object that you can use to store arbitrary items of data. You associate each item with a name or unique key value. You can use the key to retrieve the stored data.

For example, you can use the **Dictionary** object to store the details about all of the administrators in an enterprise. Details may include a name, phone number, and e-mail address. You can associate each dictionary entry with the unique site for which the administrator is responsible.

The site in the following example acts as the key that you can use to retrieve the data item. It uses the **Dictionary** object. The example creates an instance of the **Dictionary** object and then stores the details of three administrators, Alan, Kathie, and Don, in it.

```
Set AdminsList = Wscript.CreateObject("Scripting.Dictionary")
AdminsList.Add "UK", "Alan : +44 (1)234 567-8910"
AdminsList.Add "Europe", "Kathie : +33 (1)234 567-8910"
AdminsList.Add "US", "Don : +1 (0)25 555-0115"
```

To display, for example, the details about the administrator who is responsible for the UK, use the following line of code.

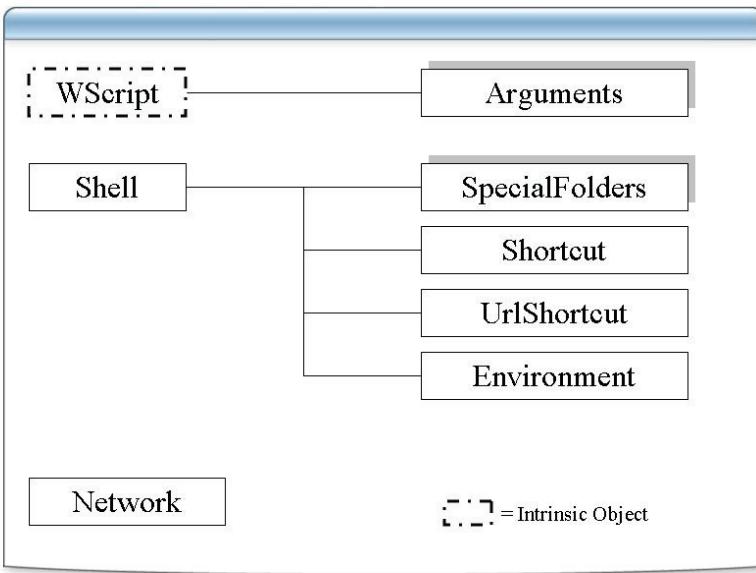
```
WScript.Echo "The UK administrator is: " & AdminsList("UK")
```

This line of script uses the associative feature of the **Dictionary** object to retrieve the detail about the UK administrator. The retrieved data is then concatenated with a string literal and displayed by using the **WScript.Echo** method.

Encoder

Scripts that have been encoded use the **Encoder** object. The encoding process is performed by using the Windows Encoder utility. This enables script writers to encode their script so that the script cannot be viewed, copied, or modified but can still be executed.

The WSH Object Model



The type library Wshom.ocx provides the object model for WSH. This object model provides some of the objects, methods, and properties that are most often used in scripting.

WScript

The WScript object is an intrinsic object, which means that it is automatically created when a script is executed and requires no explicit instantiation. The WScript object supports the following methods:

- **CreateObject**. This method enables the creation of an object for use in the script.

Note: The WScript **CreateObject** method is accessed by using the **WScript.CreateObject("")**. ProgID. If the "WScript" is left out, the VBScript **CreateObject** method is used. For most standard actions, both methods are interchangeable.

- **ConnectObject**. This method connects WSH to an existing object.
- **DisconnectObject**. This method disconnects WSH from an existing object connected by **ConnectObject**.
- **GetObject**. This method enables the retrieval of an existing object from a file, for example, the opening of an existing Office Word document.
- **Echo**. This method displays a message on the screen, either by a command line if the host is CScript or by a message box if the host is WScript.
- **Quit**. This method terminates the execution of the script.

- **Sleep.** This method causes the script execution to pause for the stated number of milliseconds. This can be very useful when sending keystrokes to an external program from a script.

Shell

This object is instantiated with the following statement.

```
Set objShell = WScript.CreateObject("WScript.Shell")
```

You can use this object to start a new process, create shortcuts, and provide the environment collection to handle access to useful environmental variables such as **WINDIR**, **COMPUTERNAME**, and **USERNAME**. This is done by using the following objects:

- **SpecialFolders.** This returns the paths for Windows shell folders such as the Desktop folder, Start menu folder, and personal My Documents folder.
- **Shortcut.** This creates an object reference to a shortcut.
- **UrlShortcut.** This creates an object reference to a URL shortcut.
- **Environment.** This retrieves environment variables from the operating system.

Network

This object is instantiated with the following statement.

```
Set objNetwork = WScript.CreateObject( "WScript.Network" )
```

The **Network** object exposes the Windows network functionality to your scripts, which provides methods that you can use to:

- List all mapped drives.
- Connect and disconnect remote drives.
- List all mapped printers.
- Connect and disconnect remote printers.
- Set the default printer.

Lesson 4: Common Object Models

- CDO
- ADO
- ADSI
- WMI
- Microsoft Office
- Internet Explorer and IIS
- Other Applications

The object models discussed so far provide the basic functionality that you can use to write useful administrative scripts. There are other object models available that you can use to perform specific tasks in your scripts.

This lesson introduces you to some of the other Automation object models that you can use for specific tasks such as sending e-mail messages and accessing databases.

Objectives

After completing this lesson, you will be able to:

- Describe the Collaboration Data Objects (CDO) model.
- Describe the ADO model.
- Describe the ADSI model.
- Describe the WMI model.
- Describe the Microsoft Office model.
- Describe the Microsoft Internet Explorer® internet browser and Internet Information Services (IIS) model.
- Describe object models in other applications.

CDO

- CDO Names
- CDO Library Versions

CDO simplifies the creation of applications or scripts with e-mail messaging functionality. The CDO libraries expose messaging objects such as folders, messages, recipient addresses, and attachments.

CDO Names

CDO has gone through the following name changes in its lifetime:

- Previous versions of CDO were called OLE Messaging. OLE Messaging was first available in Microsoft Exchange Server version 4.0.
- CDO version 1.1 was named Active Messaging. Active Messaging was installed with Exchange Server version 5.0. There were a few feature enhancements in Active Messaging, but the core functionality of the library remained unaltered.
- The current version 1.2.1 has the name CDO. CDO 1.2.1 is installed with Exchange Server 2003. This version has added functionality over CDO 1.1, including support for scheduling meetings and appointments.

Note: CDO version 1.2.1 is not included in Exchange Server 2007. If you must use CDO to support your existing applications, you must download version 1.2.1. To download CDO version 1.21, see the Microsoft Web site and search for CDO.

CDO Library Versions

The following four versions of the CDO library are available:

- CDO version 1.2.1 (Cdo.dll).

This version is installed with Microsoft Office Outlook® 2000, Outlook 2003, and Exchange Server 2003. It can be used on all Windows platforms, and it uses Messaging Application Programming Interface (MAPI) to perform its tasks.

- CDO for Windows 2000 (Cdosys.dll).

CDO 2.0 ships in Windows Server 2003 and is the recommended way to use CDO in applications that are not based on the Microsoft .NET Framework. It supports Simple Mail Transfer Protocol (SMTP) and Network News Transfer Protocol (NNTP) but does not support MAPI. It is designed for administrators because it enables you to do the following:

- Append disclaimers or other notices to e-mail messages sent through a server.
- Create applications that have messaging capabilities using Active Server Pages (ASP pages).
- Detect and discard unsolicited bulk mailings.
- Detect and discard inappropriate newsgroup postings.
- Check incoming messages for viruses.
- Forward and filter messages automatically.
- CDO for Exchange Server 2000 (Cdoex.dll).

The latest version is CDO version 3.0 for Exchange Server 2000 (CDOEX). This is installed with Exchange Server 2000 and Exchange Server 2003. CDOEX upgrades the features of CDO for Windows 2000 to include features such as calendar objects and contact management and support for the Exchange 2000 Web Storage System. You can only use CDOEX on a computer on which Exchange Server is installed.

Note: CDOEX is de-emphasized in Exchange Server 2007 and future versions of Exchange Server may not support CDOEX.

ADO

- **Data Sources**
- **Examples of Database Providers:**
 - ODBC
 - SQL Server
 - Active Directory directory service
 - Exchange 2007

ADO enables the fast and efficient access to, and manipulation of, data in a data source.

Data Sources

Data sources are often relational databases such as SQL Server 2005. However, ADO enables you to access a variety of other non-relational data sources such as Active Directory. You can use ADO to search Active Directory for objects and properties such as users and phone numbers. ADO is of interest to administrators who write scripts because it enables you to search Active Directory.

Note: The ADO provider for ADSI only provides read-only access to objects in Active Directory, any Lightweight Directory Access Protocol (LDAP)-compliant directory service, and Novell Directory Services (NDS).

Examples of Database Providers

To access data that is stored in various formats, ADO requires an OLE DB data provider for each type of data source.

The current ADO OLE DB data providers include:

- Open Database Connectivity (ODBC), for databases without a specific OLE DB data provider
- SQL Server
- Active Directory directory service
- Exchange Server 2007

The ADO model consists of the following objects:

- **Connection.** This represents a connection to an OLE DB data source such as ADSI.
- **Command.** This defines a specific command to execute against the data source.
- **Error.** This contains details about data access errors and is refreshed every time that an error occurs in a single operation.
- **RecordSet.** This represents the results of a command object execution and consists of a sequence of records.
- **Fields.** This represents the collection of fields or columns in a record set.
- **Parameters.** This represents a collection of parameters that can be supplied to the command object to modify its execution.
- **Properties.** This represents values that have been supplied by the provider for a data source.

Note: For more information about how to use ADO to search for information in Active Directory, see Module 5, “ADSI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

ADSI

- **Active Directory Service Interfaces**
- **Gives Access to Multiple Directory Service Providers Through an Open Set of Interfaces**
- **ADSI Providers Include:**
 - LDAP
 - Windows NT
 - NDS
 - NetWare 3 bindery

ADSI gives developers and administrators access to the information stored in multiple-directory service providers through an open set of interfaces. Applications or scripts that are written by using ADSI work with any directory service that has an ADSI provider.

For example, by using ADSI, scripts can access LDAP, NDS, Active Directory, and Windows Server 2003-based directories. They can do this with a single interface, using the ADSI provider on the Microsoft Windows NT® (WinNT) operating system. The only prerequisite for accessing these types of directory structures is that there is an appropriate ADSI provider available. Examples of available ADSI providers include:

- LDAP
The LDAP provider works with any LDAP version 2 or version 3 directory, such as Exchange Server 5.0 and 5.5. This provider is also used to provide the access to Active Directory.
- WinNT
The WinNT provider supports the Windows Server 2003 directory and server-based objects such as the Lanman Server.
- NDS
This provider gives you access to the NDS.
- NetWare 3 bindery (NWCOMPAT)
This provider gives access to Novell's legacy bindery (3.x) servers.

Products compatible with ADSI include Active Directory, Exchange Server 2007, IIS, and Site Server.

Note: For more information about ADSI, see Module 5, "ADSI," in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

MCT USE ONLY. STUDENT USE PROHIBITED

WMI

- **Web-Based Enterprise Management**
 - Standard management API
- **WMI**
 - Can monitor and configure the Windows operating system, system devices, Active Directory, the registry, performance counters, and so on
 - Execute methods on managed objects
 - Receive event notifications

WMI is a single, consistent application programming interface (API) that enables you to manage Windows Vista and Windows Server 2003 operating systems locally and remotely.

Web-Based Enterprise Management

Web-based Enterprise Management (WBEM) is an industry initiative to develop a standard technology for accessing management information in an enterprise. The aim is to lower the total cost of ownership of computers in an enterprise. Many companies are participating in the WBEM initiative.

WMI

WMI is the Microsoft implementation of WBEM. You can use WMI to write management scripts by using a single, object-oriented, remote-enabled, and scriptable interface that provides the following functionalities:

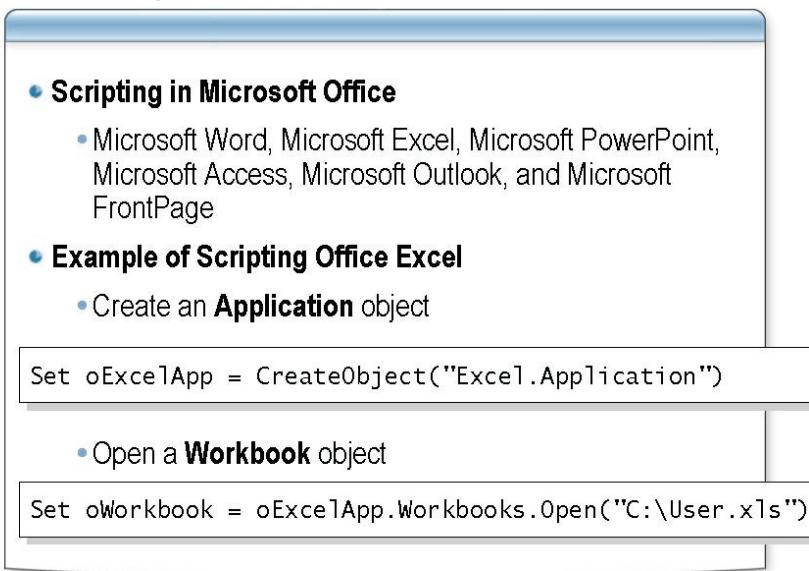
- Monitoring, configuring, and controlling management information about:
 - The Windows operating system
 - System devices
 - Active Directory
 - The registry
 - Performance counters
- Executing methods on managed objects to perform complex operations, for example, to compress a file or restart a remote system.

- Receiving and acting on events based on changes in any management data visible through WMI and events from Simple Network Management Protocol (SNMP) devices and the Windows Event Viewer service.

Note: For more information about WMI scripting, see Module 8, “WMI” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

MCT USE ONLY. STUDENT USE PROHIBITED

The Microsoft Office System



Programs in the Microsoft Office System expose Automation objects, so you can control them with your scripts.

Scripting in Microsoft Office

Microsoft Office programs provide Automation interfaces that enable you to use script to manipulate the program and its data. These programs provide detailed programmable object models that enable you to access through script all of the functions available to an interactive user.

Example of Scripting Office Excel

The following example uses Microsoft Office Excel® spreadsheet software to show what you can achieve by controlling programs in the Microsoft Office System with a script. Although the objects in the example are specific to Office Excel, you can employ a similar approach to control any of the programs in the Microsoft Office System.

Office Excel has an extensive object model that contains various classes. However, you can perform many administrative tasks with just a few objects. The **Excel.Application** object is required to access Office Excel's Automation functionality. To create an instance of the **Application** object, use the **CreateObject** function, as shown in the following code.

```
Set oExcelApp = CreateObject("Excel.Application")
```

This opens a new instance of Office Excel, which is not visible by default. You can make the Office Excel instance visible by setting the **Visible** property of the **Application** object.

After you create the **Application** object, you use the **Workbooks** collection to perform tasks such as opening a file or creating a new file. Collections contain one or more

objects of the same type, in this case, **Workbook** objects. An example of how to open a **Workbook** object and store a reference, or variable, to it is shown below.

```
Set oWorkbook = oExcelApp.Workbooks.Open(C:\Users.xls")
```

Most work that you want to do in Office Excel will involve **Worksheet** objects. This type of object gives you access to other objects such as **Ranges**. There are various ways to navigate a **Worksheet** object. The simplest approach is to specify the ordinal number or the name of the **Worksheet** object that you require in the **Workbook** object's **Worksheets** collection.

It is also useful to automate Office Excel for charting purposes. You can do this by using the **Chart** object. You create a chart by using the **Add** method of the **Workbook** object's **Charts** collection.

You can also automate Office Excel for logging or reporting purposes. You can use the **Cells** property to write and read information.

The following script shows you how to read the value of a cell and write it to the command prompt screen. The script:

- Creates an **Application** object.
- Opens a **Workbook** object.
- Iterates through the **Cells** collection until the script encounters an empty cell.

```
Set objExcel = CreateObject("Excel.Application")
Set objWorkbook = objExcel.Workbooks.Open _
    ("C:\Scripts\New_users.xls")

intRow = 2

Do Until objExcel.Cells(intRow,1).Value = ""
    Wscript.Echo "CN: " & objExcel.Cells(intRow, 1).Value
    Wscript.Echo "sAMAccountName: " & objExcel.Cells(intRow, 2).Value
    Wscript.Echo "GivenName: " & objExcel.Cells(intRow, 3).Value
    Wscript.Echo "LastName: " & objExcel.Cells(intRow, 4).Value
    intRow = intRow + 1
Loop

objExcel.Quit
```

Internet Explorer and IIS

- **Internet Explorer**
 - Script host
 - No access to WSH
- **IIS**
 - Use scripts to administer IIS
- **ASP**
 - Web-based scripting
 - Runs on IIS server

Before WSH, Visual Basic, Scripting Edition and Microsoft JScript® were primarily used for Web scripting.

Internet Explorer

Since Microsoft Internet Explorer® version 3.0, Internet Explorer has had a scripting host embedded in it. Due to this historical link with the Web browser, a lot of Visual Basic, Scripting Edition documentation still assumes that you are using script only in the context of a Web browser.

This may be misleading for systems administrators, because the functionality provided by WSH could be very different from that provided by Internet Explorer. For example, the functionality of the Internet Explorer host has been limited for security reasons, because it would not be desirable for a remote Web script to access the **FileSystemObject** object and delete files or folders.

IIS

You can run scripts to manage IIS using WSH, WMI, and ADSI. You can use these scripting tools to create Web sites and Web applications, and to monitor the health and performance of your IIS Web servers. Any applications that you run on IIS also have access to the scripting objects in WSH, WMI, and ADSI.

ASP

Microsoft introduced ASP in IIS 3.0. ASP pages make it quick and easy to write simple Web applications. ASP pages are also easy to maintain, because they are composed of simple scripting language.

ASP pages combine the power of Visual Basic, Scripting Edition with the display and interactivity of HTML. ASP pages are a simple platform on which you can develop rich and powerful Web applications. For example, you can build intuitive administrative interfaces for Windows Server 2003 by using ASP pages.

When you use script for Web programming, there are two main techniques: client-side scripting and server-side scripting. Both methods are similar, because they mix standard HTML with script.

Server-side scripting has the following advantages over client-side scripting:

- The script runs on the server, so it is independent of the browser.
- Code runs on the server, which reduces data transfer to the client.
- The script has full access to the object models on the server.

ASP is a large subject that is beyond the scope of this course. However, if you start programming with ASP pages, you will find that many of its concepts are easy for you to understand because you use Visual Basic, Scripting Edition in ASP pages to achieve your goals.

Other Applications

- **Utilities**
 - Example: Windows Media Player

Many applications and utilities expose a COM Automation interface. This makes scripted control of the application possible.

Utilities

Many utilities also expose a COM interface, which makes scripting possible. The Windows Media® technologies player is an example of such a utility. To find the possibilities for scripting these utilities, use the Object Browser and examine the object model for the program that you want to script. This shows you the object, methods, and properties that you can use in your scripts.

Lab: Objects in VBScript and WSH



- **Exercise 1
Manipulating the Scripting Object Model**
- **Exercise 2
Manipulating the WSH Object Model**
- **Exercise 3
Automating Microsoft Office Word**

After completing this lab, you will be able to:

- Manipulate the scripting object model by using Visual Basic, Scripting Edition.
- Manipulate the WSH object model by using Visual Basic, Scripting Edition.
- Automate applications in the Microsoft Office System by using Visual Basic, Scripting Edition.

Estimated time to complete this lab: 30 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine.
- Start the 2433B-VISTA-CL1-02 virtual machine.
- Log on to the 2433B-VISTA-CL1-02 virtual machine as **FOURTHCOFFEE\Administrator** using the password **Pa\$\$w0rd**

Lab Scenario

You are an administrator for the FourthCoffee corporate network. You are responsible for gathering information about folders created on computers on a domain, customizing computer settings, and writing the results to a Microsoft Office Word document. You use WSH scripts to perform these tasks.

Exercise 1

Manipulating the Scripting Object Model

In this exercise, you will write code in Visual Basic, Scripting Edition that manipulates the scripting object model. You will instantiate a **FileSystemObject** object and a **Folder** object and manipulate them by using their methods and properties.

The principal tasks for this exercise are as follows:

- To instantiate scripting objects.
- To retrieve scripting object properties.

Task	Supporting information
1. To instantiate scripting objects.	<ul style="list-style-type: none"> • Start PrimalScript. • Using the ScriptFiles template list, create a new VBScript file. • Declare three variables: objFSO, objFolder, and colSubFolders. • Instantiate the FileSystemObject object and assign it to the objFSO variable. • Assign a folder on drive C of the virtual machine to the objFolder variable.
2. To retrieve scripting object properties.	<ul style="list-style-type: none"> • Assign the collection of subfolders associated with objFolder to the colSubfolders variable. • Write the properties of each subfolder in the collection to the screen. • On the PrimalScript Script menu, click Run Script. • Observe the script's results in the PrimalScript output pane.

Questions

Q: What is missing from this line of script?

```
MyFSO = CreateObject("Scripting.FileSystemObject")
```

Q: What ADSI provider is used to access Active Directory?

Exercise 2

Manipulating the WSH Object Model

In this exercise, you will write code in Visual Basic, Scripting Edition that manipulates the WSH object model. You will instantiate a **Shell** object and a **Network** object and manipulate them by using their methods and properties.

The principal tasks for this exercise are:

- To instantiate and manipulate the **Shell** object.
- To instantiate and manipulate the **Network** object.

Task	Supporting information
1. To instantiate and manipulate the Shell object.	<ul style="list-style-type: none">• Using the ScriptFiles template list, create a new VBScript file.• Declare four variables: objShell, objShortcut, strSendToFolder, and strPathToNotepad.• Instantiate the WScript.Shell object and assign it to the objShell variable.• Use the properties and methods of the Shell object to create a Send To shortcut to Notepad when a user right-clicks a file.
2. To instantiate and manipulate the Network object.	<ul style="list-style-type: none">• Using the ScriptFiles template list, create a new VBScript file.• Declare one variable: objNetwork.• Instantiate the WScript.Network object and assign it to the objNetwork variable.• Use the methods of the Network object.• On the PrimalScript Script menu, click Run Script.• Observe the script's results in the PrimalScript output pane.

Questions

Q: What object model provides access to the Windows **Shell** and **Network** objects?

Exercise 3

Automating Microsoft Office Word

In this exercise, you will write code in Visual Basic, Scripting Edition that manipulates the Microsoft Office Word object model. You will instantiate an **Application** object. You will create and modify a **Document** object. You will then save the **Document** object and review its contents.

The principal tasks for this exercise are:

- To automate Microsoft Office Word.

Task	Supporting information
1. To automate Microsoft Office Word.	<ul style="list-style-type: none">• Using the ScriptFiles template list, create a new VBScript file.• Declare three variables: objWord, objDoc, and objSelection.• Instantiate the Word.Application object and assign it to the objWord variable.• Add objDoc to the Documents collection of the Application object.• Assign objSelection to the Selection property of the Application object.• Set the Selection property values to set fonts and text in the Office Word document.• On the PrimalScript Script menu, click Run Script.• A Word document opens with the text and font settings you defined.

Questions

Q: To which protocols does the Cdosys.dll allow scriptable access?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-02 virtual machines and discard any changes.

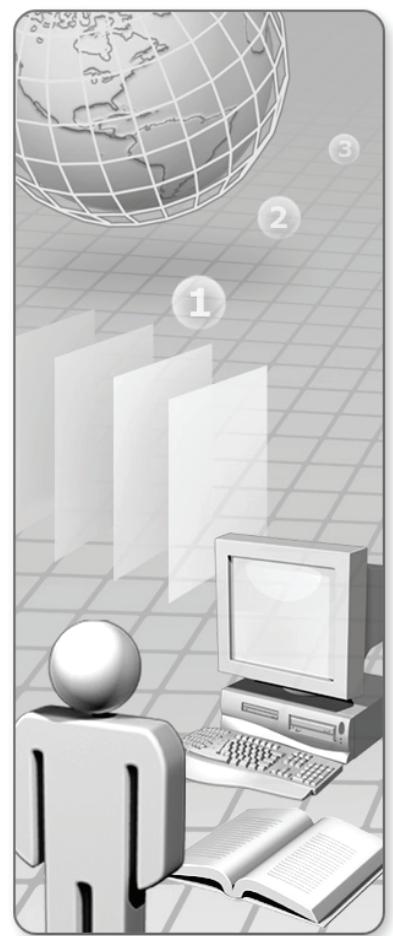
Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

MCT USE ONLY. STUDENT USE PROHIBITED

Module 3: Script Logic

Table of Contents

Module Overview	3-1
Lesson 1: Fundamental VBScript Rules	3-2
Lesson 2: Variables, Constants, and Data Types	3-6
Lesson 3: Operators	3-23
Lesson 4: Conditions and Loops	3-31
Lesson 5: Procedures	3-43
Lesson 6: Script Layout	3-50
Lab: Script Logic	3-57



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Fundamental VBScript Rules**
- **Variables, Constants, and Data Types**
- **Operators**
- **Conditions and Loops**
- **Procedures**
- **Script Layout**

Microsoft® Visual Basic®, Scripting Edition (VBScript) is a powerful scripting language. In this module, you will learn how Visual Basic, Scripting Edition provides an effective, efficient, and powerful framework that you can use to write administrative scripts. You will also learn about script rules and logic, and how to use effective script layouts.

Objectives

After completing this module, you will be able to:

- Describe the rules of the Visual Basic, Scripting Edition language.
- Declare and use variables, constants, and data types in your scripts.
- Use Visual Basic, Scripting Edition language operators.
- Construct conditional code and looping structures.
- Declare and use **Sub** and **Function** procedures.
- Determine an effective script layout.

Lesson 1: Fundamental VBScript Rules

- Case and Space Rules
- Line and Name Rules

Before looking at the mechanics of Visual Basic, Scripting Edition, it is important to understand the basic rules that your scripts should obey.

This lesson will teach you about the Visual Basic, Scripting Edition rules for character case, white space, line length, and naming.

Objectives

After completing this lesson, you will be able to:

- Use Visual Basic, Scripting Edition rules for character case and white space.
- Use Visual Basic, Scripting Edition rules for line length and naming.

Case and Space Rules

- **Case**
 - VBScript is not case-sensitive
- **Space**
 - VBScript ignores extra white space

Visual Basic, Scripting Edition has rules for character case and for white space in code.

Case

Visual Basic, Scripting Edition is not case-sensitive. The following commands are equivalent.

```
WSCRIPT.ECHO  
WScript.echo  
wscript.echo
```

Although Visual Basic, Scripting Edition is not case-sensitive, it is considered bad practice to use all uppercase or all lowercase characters, because this makes the script difficult to read.

White Space

The script engine ignores extra white space in any lines of script when it is parsed. For example, the first line of code below is the same as the second line.

```
WScript.    Echo      "Hello"  
WScript.Echo "Hello"
```

This approach gives flexibility to the script writer when laying out the script in a manner that is easy to read.

Line and Name Rules

- **Lines and Line Length**
 - VBScript does not impose a maximum line length
 - VBScript enables you to split long lines
- **Names**
 - A name written in VBScript cannot exceed 255 characters

Visual Basic, Scripting Edition has rules for line length and naming.

Lines and Line Length

Unlike other languages, such as Visual Basic, Visual Basic, Scripting Edition does not impose a maximum number of characters per line. In addition, you can write what are effectively separate lines of code on a single line by using the colon as a divider. The following combines two logical lines by using the colon.

```
WScript.Echo "Test One" : WScript.Echo "Test Two"
```

Visual Basic, Scripting Edition enables you to type a line of code over several lines in the script. Long statements can be broken into multiple lines by using the line-continuation character (a space followed by an underscore). Using this character can make the script easier to read. The following example shows a long line of code that is split into three lines by line-continuation characters.

```
WScript.Echo "Message to" _  
& vUserName & _  
" Please lock your workstation"
```

If you split a long string across multiple lines, you must manage the string data on each line as a separate string. You can achieve this by ending each string with double quotes, and then concatenating the subsequent string to the previous line by using the string-concatenation character (&). The following example illustrates how to split a string across multiple lines.

```
WScript.Echo "Message to" _  
& vUserName _  
& " Please make sure that you remember" _
```

```
& " to log off or lock your workstation" _  
& " if you leave it unattended, Thanks"
```

Names

Visual Basic, Scripting Edition imposes a limitation of 255 characters on any names that are used to identify items such as variables, constants, and procedures.

Lesson 2: Variables, Constants, and Data Types

- **Variables**
- **Constants**
- **Data Types**
- **Naming Conventions**
- **Arrays**
- **Dynamic Arrays**

Before you can successfully write powerful administrative scripts by using Visual Basic, Scripting Edition, you must be able to use variables, constants, and data types. It is also important to understand what an array is and how to work with both single-dimensional and multi-dimensional arrays.

This lesson will teach you about variables, constants, data types, and arrays. It will also teach you about naming conventions.

Objectives

After completing this lesson, you will be able to:

- Use variables with scripts.
- Define constants to use with scripts.
- Use Visual Basic, Scripting Edition data types.
- Write scripts using consistent naming conventions.
- Use arrays with scripts.
- Use dynamic arrays with scripts.

Variables

- **Containers for Values That May Change**
 - Value is accessible by the variable name
- **String Data**
 - strServer = "MyServer"
- **Dates and Times**
 - dDOB = #05-06-1976#
- **Naming Restrictions**
- **Declaring Variables**
 - Implicit or explicit?
 - Option Explicit

Variables are names that you define, which point to memory locations that your script can access. Visual Basic, Scripting Edition shields the memory locations from you. You can define variables in your script to hold data in memory in a convenient and easy-to-use manner. By holding data in variables, you can manipulate any data required by your script.

Container for Values That May Change

You use variables to store data items. The values stored may change during the lifetime of the script, as the term “variable” implies. You access the value of the data by using the variable name.

```
nMyAge = nMyAge + 10  
WScript.Echo nMyAge
```

In this example, you manipulate the variable nMyAge by retrieving its current value and adding 10 to it. The new value is then echoed to the screen. The script writer does not need to know the value of nMyAge in advance. A user could provide the nMyAge value to the script at run time using a pop-up box. Variables that contain a single value, such as nMyAge, are described as scalar variables.

String Data

A string is a data type that indicates that the variable contains characters, rather than numeric values. To store string data in a variable, you must enclose the data in quotation marks, as the following example shows.

```
strServer = "MyServer"
```

Dates and Times

When assigning dates or times to variables, you must enclose the variables with the number sign (#), as the following example shows.

```
dToday = #05-06-2000#
```

This indicates that the script written in Visual Basic, Scripting Edition interprets the value as a date or time, rather than as a string or a calculation such as 5–6–2000.

Naming Restrictions

When naming variables in Visual Basic, Scripting Edition, the following rules apply. A variable name:

- Must begin with an alphabetic character.
- Cannot contain an embedded period.
- Must not exceed 255 characters.
- Must be unique in the current scope.

You will learn more about scope later in the module.

Note: You cannot assign the name of a VBScript keyword to a variable. For a list of VBScript keywords, see the VBScript documentation.

Declaring Variables

By default, script written in Visual Basic, Scripting Edition implicitly creates a variable the first time that it reads a name that is not a reserved word or a procedure name. This sort of behavior can easily result in script errors due to mistyped variable names.

In the following example, the script writer has mistakenly typed nMiAge instead of nMyAge on the second line. The result of this typographical error is that instead of echoing a value of 45, as expected, a value of 10 is returned. This is because the script written in Visual Basic, Scripting Edition implicitly created a new variable called nMiAge. It is initialized with a value of zero.

```
nMyAge = 35  
nMyAge = nMiAge + 10  
WScript.Echo nMyAge
```

This type of error can be very difficult to trace, so it is considered good practice to force the script written in Visual Basic, Scripting Edition to require explicit variable declarations by using the following line at the top of the script.

```
Option Explicit
```

After you have added the **Option Explicit** command to the script, any undeclared variables will cause the script to generate an error rather than implicitly create a variable.

After you have added **Option Explicit**, you must explicitly declare all new variable names before they can be used. You can declare variables by using any of the following statements:

- **Dim**
- **Private**
- **Public**
- **Static**

You will see the differences between these statements later in this module. You can use all of the statements to declare the name of the variable and reserve some memory to hold the value of the variable, as the following example shows.

```
Dim nMyAge
```

You can include multiple variables in a single declaration statement by using the comma as a separator between each name. The following example illustrates how you can achieve this.

```
Dim nMyAge, nMyName, nMyAddress
```

Constants

- **Values Do Not Change During Script Execution**

- Makes scripts easier to read and debug
 - Assigned at the start of the script

- **The Const Statement**

```
Const COMP_NAME = "Microsoft"
```

- **Type Library Constants**

- **Intrinsic Constants**

Examples: vbCrLf, vbRed

As well as declaring and using variables in your scripts, you can declare and use constants.

Values Do Not Change During Script Execution

A constant is a name that you assign to a memory location in a manner that is similar to variables. Like variables, constants hold your data. However, the value assigned to a constant cannot change during the execution of a script.

Using constants helps to make your scripts manageable. If a script uses a constant value, such as a tax rate, in several places and the value must be modified in the future, the script writer must only change the value at the location where the constant is defined. All of the other instances of the constant then contain the new value.

Constants can also make script easier to read, and therefore easier to debug, as long as you have used meaningful constant names.

The Const Statement

A constant can represent a string, a number, or a combination that includes most arithmetic, comparison, and logical operators.

The following is an example of the syntax used for setting constants.

```
Const COMP_NAME = "Microsoft"
```

If you attempt to change a constant's value in the script, the script written in Visual Basic, Scripting Edition generates the following compilation error.

```
Microsoft VBScript compilation error: Name redefined
```

Type Library Constants

Many type libraries that describe object models expose constant values that can be legitimately passed as method parameters.

For example, in the Microsoft Active Directory® Service Interfaces (ADSI) object model, many of these constants are exposed to help with the creation of new objects such as users and groups in Active Directory. When you create a new group, ADSI determines the type of group (Domain Local, Global, or Universal) by using a creation flag. If the flag has the value 2, ADSI knows to create a new Global Group. Other constant values correspond to the other group types. To help with readability, the ADSI type library exposes these constant values by using meaningful names, as the following example shows.

```
ADS_GROUP_TYPE_GLOBAL_GROUP
```

The type library itself declares the above constant and assigns it a value of 2.

Visual Basic and Microsoft Visual C++® development system provide support for directly referencing the type library. This enables developers to use the predefined constants in their code.

Note: VBScript is unable to read information directly from a type library. You can overcome this limitation by using Windows® Script (.wsf) files. For more information, see Module 5, “ADSI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

If you are using a standard script written in Visual Basic, Scripting Edition, you must discover the values of these constants and explicitly define them as equivalent constants at the top of the script file, as the following example shows.

```
CONST ADS_GROUP_TYPE_GLOBAL_GROUP = &H2
```

This line declares the following constant and assigns the value of 2 to it. The **&H** tells the script that this is a hexadecimal value so that it is stored correctly. In this instance, **&H2** is the same as a decimal value of 2. If the value is **&H32**, the decimal value stored by the script is 50. After this definition is in place, you can use the more meaningful constant name.

Note: These constant values are published in the Windows Server® 2003 R2 Platform software development kit (SDK). A quick way to find the value is to search for the constant name. Alternatively, you can search for ENUM to display a list that contains all of the enumerations available.

Intrinsic Constants

In addition to the constants that the script writer defines, there are a number of useful constants built into Visual Basic, Scripting Edition. These constants, referred to as intrinsic constants, provide a convenient way to use specific values without having to remember the value itself. You are not required to declare these constants in the script

because they are already defined in Visual Basic, Scripting Edition. You can simply use them in place of the values that they represent.

Note: The names of the constants are treated as reserved words because it is not possible to disable intrinsic constants.

For a full list of these intrinsic constants, refer to the VBScript documentation. The following table illustrates the various categories of constants and a brief description of each.

Constant grouping	Definition
Color	Defines eight basic colors that you can use in scripting. For example, vbRed represents the number 255, which is interpreted as red.
Date-and-time	Defines date-and-time constants that are used by various date and time functions. For example, vbMonday represents the value 2, vbSunday equals 1, and vbTuesday equals 3.
Date format	Defines constants that are used to format dates and times.
Miscellaneous	Defines constants that do not conveniently fit into any other category.
MsgBox	Defines constants that are used in the MsgBox function to describe button visibility, labeling, behavior, and return values.
String	Defines a variety of nonprintable characters that are used in string manipulation.
Tristate	Defines constants that are used with functions that format numbers.
VarType	Defines the various Variant subtypes.

The following example illustrates a Visual Basic, Scripting Edition constant.

`vbCrLf`

This constant represents the value of Chr(13) and Chr(10), which are the ASCII characters for the carriage return and line-feed characters. This is useful when formatting the following code.

```
sMessage = "Welcome to the world of VBScript" & vbCrLf &_
    "and all the useful commands that it provides" & _
    vbCrLf & "Please click the OK button to continue"
MsgBox sMessage
```

The first line assigns the message string to a variable called `sMessage`. Then, the **MsgBox** function displays the string as three lines of text rather than one long line.

Data Types

- **Variants**
- **Variant Subtypes**

Empty	Null	Boolean
Byte	Integer	Currency
Long	Single	Double
Date (Time)	String	Object
Data Object	Array	Error

- **Conversion Functions**

Unlike most modern compiled languages such as Visual Basic and Visual C++, Visual Basic, Scripting Edition uses one data type for all data. This type is called the **Variant** data type.

Variants

A **Variant** can contain numeric data, dates and times, objects, Boolean variables, or strings. In most cases, Visual Basic, Scripting Edition automatically detects the type of data stored and deals with it accordingly.

In the following example, vAnswer contains the string "MyServer," because vDemoA and vDemoB contain string values declared in quotes. The addition operator in this example performs a string concatenation.

```
vDemoA = "My"  
vDemoB = "Server"  
vAnswer = vDemoA + vDemoB  
WScript.Echo vAnswer
```

The next example looks very similar, but the vDemoA and vDemoB variables contain numeric data. This time, the result of the addition operation generates another number. As a result, vAnswer holds data that is numeric.

```
vDemoA = 5  
vDemoB = 10  
vAnswer = vDemoA + vDemoB  
WScript.Echo vAnswer
```

Variant Subtypes

It is possible to divide the **Variant** data type into subtypes. These subtypes enable you to exercise a finer level of control on how your script manipulates data.

For example, you can have numeric data that represents currency, a date, or a time. The **Variant** interprets the data accordingly.

A **Variant** contains many subtypes, as the following table illustrates.

Subtype	Description
Empty	Variant is uninitialized.
Null	Variant contains no valid data.
Boolean	Contains either True (-1) or False (0).
Byte	Contains an integer between the range of 0 and 255.
Integer	Contains an integer between the range of 32,768 and 32,767.
Currency	Contains a fixed decimal number between the range of -922,337,203,685,477.5808 and 922,337,203,685,477.5807.
Long	Contains an integer between the range of -2,147,483,648 and 2,147,483,647.
Single	Contains a single-precision, floating-point number between the range of -3.402823E38 and -1.401298E-45 for negative values and 1.401298E-45 and 3.402823E38 for positive values.
Double	Contains a double-precision, floating-point number between the range of -1.79769313486232E308 and -4.94065645841247E-324 for negative values and 4.94065645841247E-324 and 1.79769313486232E308 for positive values.
Date (Time)	Contains a number that represents a date or time between the range of January 1, 100 and December 31, 9999.
String	Contains a variable-length string that can be up to approximately two billion characters in length.
Object	Points to an object.
Data Object	Points to a data access object.
Array	Contains an array.
Error	Contains an error number.

Conversion Functions

Sometimes, a strict level of control over the exact data type is required. In these situations, you can use conversion functions to convert data from one subtype to another.

A typical example of when this might be required is when you extract a particular set of digits from a number. This operation is much easier to achieve by using string-handling functions such as **Left**, **Right**, and **Len**. To use these functions, you must first convert the numeric value to a string by using the **CStr** function. After you have extracted the digits, you can then convert them back into an integer by using the **CInt** function.

The following table lists the functions that can convert data into a **Variant** of a specific subtype.

Function name	Conversion
CInt	Returns a Variant of subtype Integer .
CBool	Returns a Variant of subtype Boolean .
CByte	Returns a Variant of subtype Byte .
CCur	Returns a Variant of subtype Currency .
CDate	Returns a Variant of subtype Date .
CDbl	Returns a Variant of subtype Double .
CLng	Returns a Variant of subtype Long .
CSng	Returns a Variant of subtype Single .
CStr	Returns a Variant of subtype String .

Note: For more information about conversion functions, see the VBScript documentation.

Naming Conventions

- **Hungarian Naming Convention**
 - Prefix names with a type identifier
 - Example: Wscript.Echo objMyServer.Name
 - Constants must be capitalized
- **Be Consistent**

Another method that you can use to help to improve the readability of script is to employ a naming convention for your variables and constants.

Visual Basic, Scripting Edition does not require a naming convention. However, it is generally accepted that implementing some sort of naming convention is good practice.

Hungarian Naming Convention

Some of the rules of this convention are that:

- Variable names should begin with one or more lowercase letters. These letters are used to indicate the type of data stored in the variable, as the following table shows.

Prefix	Data type
str	String
Fn	Function
c (or capitalize each letter in the constant name)	Constant
b	Boolean (True or False)
d	Date
obj	An object reference
n	A numeric value

- Function, method names, and each word in a multiple-word name must begin with a capital letter, as the following examples show.

```
WScript.Echo objMyServer.Name
```

```
Sub CheckDateSub()
```

- Early versions of Visual Basic, Scripting Edition had no specific **Const** command, so constants were simply defined as variables that were named in uppercase, and underscores were used to separate words, as the following examples show.

```
COMPANY_NAME  
DEFAULT_PATH
```

Visual Basic, Scripting Edition now has a **Const** command but the uppercase and underscore format has become the preferred naming convention for constants.

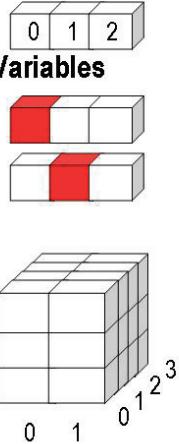
Note: This naming convention is a broad subject, but most of the detail is not relevant to scripting due to the limited data types available to script. For more information, see the MSDN Hungarian Notation Web site.

Be Consistent

You do not have to use the Hungarian naming convention. You can easily develop your own naming convention. For example, you can use dtm as the prefix for dates and times. However, whatever the naming convention that you use, it is important that you apply it consistently in all scripts. This consistency assists in the reading and debugging of the scripts in the future.

Arrays

- **A Variable That Can Contain Multiple Values**
 - Dim MyArray(2)
- **Accessing the Values in Array Variables**
 - MyArray(0) = 100
 - WScript.Echo MyArray(1)
- **Multi-Dimensional Arrays**
 - Dim MultiDArray(1,3,2)



You may want to store multiple values in memory. In some cases, the values are related. Rather than declaring and using multiple, separate variables to achieve your aims, you can instead use arrays.

A Variable That Can Contain Multiple Values

An array is a type of variable that can contain multiple values. You can specify the number of discrete values that an array can store when you declare it. You do this by specifying a number in brackets after the variable name. The number that you supply is one less than the number of values that you want to store. For example, the following syntax specifies that the array can handle three discrete values.

```
Dim MyArray(2)
```

The number is known as the upper bound of the array. The array can store discrete values by using any number between zero and the upper bound, including zero and the upper bound itself. Therefore, it can store a number of values that is one greater than the upper bound.

Accessing the Values in Array Variables

You can access the values in an array by using a notation that uniquely identifies each value inside the array. You use a number between zero and the upper bound of the array to access a discrete value that it contains. The following example sets the first element (element number zero) to a value of 100.

```
MyArray(0) = 100
```

The next example retrieves the value in the second element (element number one) and uses it as a parameter to the **Echo** method of the **WScript** object.

```
WScript.Echo MyArray(1)
```

Multi-Dimensional Arrays

It is possible to have multi-dimensional arrays. You declare a multi-dimensional array by providing a comma-separated list of upper bounds, as the following example shows.

```
Dim MyTable(4, 9)
```

This example shows two numbers being used as the upper bounds, which indicates that this is a two-dimensional array. The following table illustrates how you can visualize a two-dimensional array as a table.

0	1	2	3	4	5	6	7	8	9	10
0										
1										
2										
3										
4										
5										

You can use up to 60 dimensions for an array in Visual Basic, Scripting Edition. You can visualize an array that has three dimensions as a cube. It becomes harder to visualize an array with four or more dimensions. However, most arrays that you will use in Visual Basic, Scripting Edition will have no more than four or five dimensions and will often contain only one or two.

Note: Arrays can contain variables of different data types. This is useful if you are storing values related to the same object. For example, in a two-dimensional array, the first value can be a person's name, and the rest of the row can be the values relating to that person, such as a telephone number. The next row can be the next person.

Dynamic Arrays

- **Creating Dynamic Arrays**
 - Dim MyArray()
- **Resizing an Array**
 - ReDim MyArray(6)
 - ReDim Preserve MyArray(6)
- **Array Statement**
 - Dim MyArray
 - MyArray = Array (Value1, Value2, Value3)
- **Emptying an Array**
- **The UBound Function**

The previous topic described how to declare array variables by providing upper-bound arguments in the declaration statement. However, providing the upper bound in this manner results in a fixed-size array. After you have declared the array, you can no longer change the number of elements that it contains or the number of dimensions that define the array's dimensionality. This approach is ideal if you know the number of dimensions and elements for your array in advance.

Creating Dynamic Arrays

You may want to use arrays in your scripts but may not know the upper bound or the dimensions of the array when you are writing the code. For these situations, you can use a dynamic array.

There are three basic steps to using a dynamic array in Visual Basic, Scripting Edition:

1. Declare the array, but omit the upper-bound argument.

```
Dim MyArray()
```

2. Initialize the array to the required size by using the **ReDim** statement.

```
ReDim MyArray(4)
```

You can use **ReDim** on its own to create and initialize an array in one step.

3. Use the array as described in the previous topic.

Resizing an Array

You can use the **ReDim** statement to resize the array if required. For example, you may start with an array that has five elements, but at some point, you may want to resize it to hold eight values. You can reissue the **ReDim** statement, as the following example shows.

```
Dim MyArray()  
Redim MyArray(4)  
MyArray(0)=100  
MyArray(1)=200  
MyArray(2)=300  
MyArray(3)=400  
MyArray(4)=500  
...  
...  
ReDim MyArray(7)  
...
```

It is important to understand that using the **ReDim** statement removes the existing values in the array. In the previous example, the five-element array was populated with the data as shown, but then this data was lost when the array was resized.

You can use the **Preserve** keyword with the **ReDim** statement to alter this behavior. As the name of the keyword implies, it preserves the contents of the array, if possible, when the resizing takes place.

In the following example, **ReDim** sets the initial number elements in the dynamic array to five. A subsequent **ReDim** statement resizes the array but uses the **Preserve** keyword to protect the data already in the array.

```
Dim MyArray()  
Redim MyArray(4)  
MyArray(0)=100  
MyArray(1)=200  
MyArray(2)=300  
MyArray(3)=400  
MyArray(4)=500  
...  
...  
ReDim Preserve MyArray(7)  
...
```

Caution: If you resize a fully populated array so that it contains fewer elements than before, then you will inevitably remove some data, even if you use the **Preserve** keyword.

Array Statement

You can also achieve dynamic arrays by using the **Array** statement. This has the advantage that it can initialize and populate an array at the same time, as the following example shows.

```
Dim MyArray  
MyArray = Array(100, 200, 300, 400, 500)
```

The **Dim** statement declares the variable name. The variable is then converted into an array, initialized to contain five elements, and populated with the values shown in the parentheses, all in one line. Note that the statement does not require an upper bound to be explicitly declared. The upper bound of the array will be created to fit the number of elements in the parentheses.

Emptying an Array

You can use the **Erase** statement to delete the contents of an array.

```
Erase MyArray
```

This clears the contents of a fixed-size array. The array itself still exists. However, if using the **Erase** statement empties a dynamic array, the data is removed and the array is uninitialized. You must redimension the array by using the **ReDim** statement before you can use the array again.

The **UBound** Function

With dynamic arrays, you may not know how many elements they contain when you write your scripts. However, you must often work this out in code. For example, you must know the upper bound when using a loop to iterate through all of the elements in the array. You can use the **UBound** function in these situations. This function returns the number of the highest element in the array, as the following syntax shows.

```
UBound(arrayname[, dimension])
```

In this syntax, **arrayname** is the name of the array, and **dimension** is the required dimension that you want to check. If only one dimension exists, **dimension** is optional.

Lesson 3: Operators

- Operators
- Operator Precedence
- Class Discussion: Operators

If you want to successfully manipulate your data, you must understand how to use the Visual Basic, Scripting Edition operators on variables, arrays of variables, and constants.

This lesson will teach you about the Visual Basic, Scripting Edition operators, and how to use operator precedence when you are using several operators in a Visual Basic, Scripting Edition statement.

Objectives

After completing this lesson, you will be able to:

- Use arithmetic, concatenation, comparison, and logical operators in your scripts.
- Use operator precedence when there is more than one operator in a statement.

Operators

- **Arithmetic:**
 - ^ - * / \ Mod + -
- **Concatenation:**
 - &
- **Comparison:**
 - = <> < > <= >= IS
- **Logical:**
 - Not And Or XOr Eqv Imp

Operators are the reserved characters that you use with variables and other data in your scripts. The operators provided by Visual Basic, Scripting Edition can be categorized into four types: arithmetic, concatenation, comparison, and logical.

Arithmetic Operators

Arithmetic operators perform basic arithmetic functions such as addition, subtraction, and multiplication. The following table describes the arithmetic operators.

Operator	Description	Symbol
Exponentiation	Raises a number to the power of an exponent.	^
Unary negation	Indicates the negative value of a numeric expression.	-
Multiplication	Multiplies two numbers.	*
Division	Divides two numbers and returns a floating-point result.	/
Integer division	Divides two numbers and returns an integer result.	\
Modulus arithmetic	Divides two numbers and returns only the remainder.	Mod
Addition	Sums two numbers.	+
Subtraction	Finds the difference between two numbers.	-

Concatenation Operator

You may want to create a string by joining together (concatenating) two or more other strings. The string concatenation operator is the ampersand (&). A space must follow the ampersand for the concatenation to work correctly. For example, the following script

sample concatenates two string literals and a string variable into a single string that is then echoed to the screen.

```
WScript.Echo "The path is, " & strPath & " on Server B"
```

You can also concatenate numeric variables with string values.

```
nAge = InputBox("Please enter your Age.")
nTogo = 65 - nAge
MsgBox "You have " & nTogo & " years until you retire!"
```

In this example, the variable `nAge` is assigned the value entered in the input box. A simple calculation is then performed to calculate the years left until the age of 65. The result is stored in the variable `nTogo`. This number is then concatenated with two string literals, and the resulting string is displayed in a message box.

Warning: It is possible to use the plus sign (+) to provide concatenated strings. However, this is not recommended because it will generate errors when mixing string and numeric values.

Comparison Operators

You use comparison operators to compare values. By using these operators, you can build an expression that will test for a condition and return either **True** or **False**. The following table describes the available operators.

Description	Symbol
Equality	=
Inequality	<>
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Object equivalence	IS

The following is an example of the syntax for these operators.

```
result = value1 operator value2
```

Logical Operators

Logical operators are used in a similar way to comparison operators. For example, the **Not** operator performs logical negation on an expression, as the following syntax shows.

```
bNegResult = Not bResult
```

If the value of **bResult** is **True**, the value of **bNegResult** is set to **False**.

The following table describes the logical operators supported by Visual Basic, Scripting Edition.

Operator	Description	Symbol
Negation	Performs logical negation on an expression.	Not
Conjunction	Performs a logical conjunction on two expressions.	And
Disjunction	Performs a logical disjunction on two expressions.	Or
Exclusion	Performs a logical exclusion on two expressions.	XOr
Equivalence	Performs a logical equivalence on two expressions.	Eqv
Implication	Performs a logical implication on two expressions.	Imp

Operator Precedence

- **Precedence List**
 - *, /
 - +, -
 - &
- **Parentheses**
 - Evaluated first

You can build complex tests by using a mixture of operators. It is important to understand how the various operators are resolved and the order in which each part of the expression is evaluated. This order is determined by the order of operator precedence that Visual Basic, Scripting Edition defines.

Precedence List

The following list presents the operators in the order of precedence that Visual Basic, Scripting Edition defines:

- ^
- -
- *, /, and \
- MOD
- +, -
- &
- =, <>, <>, <=, >=, and IS
- NOT
- AND
- OR
- XOR
- EQV
- IMP

The following example shows how operator precedence works with the common arithmetic operators:

$35 + 15 / 5 - 4 * 6$ evaluates to 14

These are the steps in the evaluation:

1. $4 * 6 = 24$
2. $15 / 5 = 3$
3. $35 + 3 = 38$
4. $38 - 24 = 14$

If the arithmetic operators are in a different order, the answer is different. For example:

$35 * 15 / 5 + 4 - 6$ evaluates to 103

These are the steps in the evaluation:

1. $35 * 15 = 525$
2. $525 / 5 = 105$
3. $105 + 4 = 109$
4. $109 - 6 = 103$

Parentheses

You can use parentheses to control the order of precedence. When an operator followed by an opening parenthesis is encountered, the expression in the parentheses is evaluated as an independent expression, and the result is then used with the operator preceding the opening parenthesis. For example:

$(35 + 15) / 5 - 4 * 6$ evaluates to -14

These are the steps in the evaluation:

1. $4 * 6 = 24$
2. $35 + 15 = 50$
3. $50 / 5 = 10$
4. $10 - 24 = -14$

Discussion: Operators

- A. $1+4+10*6-4/2$
- B. $(1+4+10)*6-(4/2)$
- C. $1+4+10*(6-4)/2$
- D. $(1+4+10) \& (6-4)/2$
- E. $1+4+10 \& (6-4)/2$

In this discussion, you will review five calculations with the rest of the class. Each calculation looks similar but, by using the various rules of precedence, produces different results. Record the value of each expression in the space below it.

$$1 + 4 + 10 * 6 - 4 / 2$$

$$(1 + 4 + 10) * 6 - (4 / 2)$$

$$1 + 4 + 10 * (6 - 4) / 2$$

$$(1 + 4 + 10) \& (6 - 4) / 2$$

$1 + 4 + 10 \& 6 - 4 / 2$

MCT USE ONLY. STUDENT USE PROHIBITED

Lesson 4: Conditions and Loops

- **Conditional Statements: If...Then**
- **Conditional Statements: Select Case...End Select**
- **Looping: For...Next**
- **Looping: Do...Loop**
- **Built-in Functions**

Standard programming techniques such as decision-making and looping can add useful functionality to your administrative scripts.

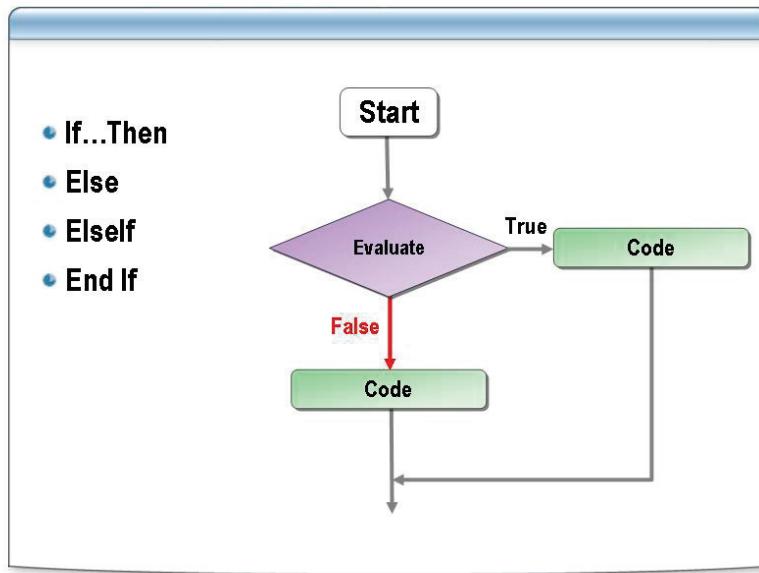
This lesson will teach you how to write conditional code by using two standard programmatic approaches. You will then learn how to construct effective looping structures by applying a variety of techniques. In addition, you will learn about some of the built-in functions that Visual Basic, Scripting Edition provides to enable you to write manageable and effective code.

Objectives

After completing this lesson, you will be able to:

- Use conditional **If...Then** statements in scripts.
- Use conditional **Select Case...End Select** statements in scripts.
- Use **For...Next** loops in scripts.
- Use **Do...Loop** looping in scripts.
- Use Visual Basic, Scripting Edition built-in functions.

Conditional Statements: If...Then



Use conditional structures to control the flow of the code in your scripts. When you want to write code that only runs in certain circumstances, you can achieve this by using conditional structures.

If...Then

You will often use the **If...Then** statement in Visual Basic, Scripting Edition because it provides a simple test to evaluate whether a condition is **True** or **False**. The following is an example of the syntax used for an **If...Then** statement.

```
If <condition> Then <action>
```

In the following example, the message “MyVar = 5” is echoed to the screen only if the MyVar variable contains the value 5. If the variable has any other value, the message does not appear.

```
If MyVar = 5 Then WScript.Echo "MyVar = 5"
```

Else

You can add the **Else** statement to the **If...Then** statement. This enables you to execute statements if the condition within the **If** statement is not met, as the following example shows.

```
If MyVar = 5 Then  
    WScript.Echo "MyVar = 5"  
Else  
    WScript.Echo "MyVar does not equal 5"  
End If
```

In this script, the message “MyVar does not equal 5” is displayed unless the MyVar variable contains the value 5.

Note: The expansion of the structure from the single-line syntax, shown previously, to the multiple-line format, as shown in this example, is known as a **Block If** structure. When you develop **Block If** structures, you must always remember to include the keyword **End If** at the end of the structure.

Elseif

The **ElseIf** construct extends the functionality of the **If** statement by allowing the condition to have more than two different results, as the following example shows.

```
If MyVar = 0 Then
    WScript.Echo "MyVar = 0"
ElseIf MyVar = 1 Then
    WScript.Echo "MyVar = 1"
ElseIf MyVar = 2 then
    WScript.Echo "MyVar = 2"
Else
    WScript.Echo "Sorry value is out of range"
End If
```

End If

You can add the **End If** statement if you want to combine multiple steps in a single test, as the following example shows.

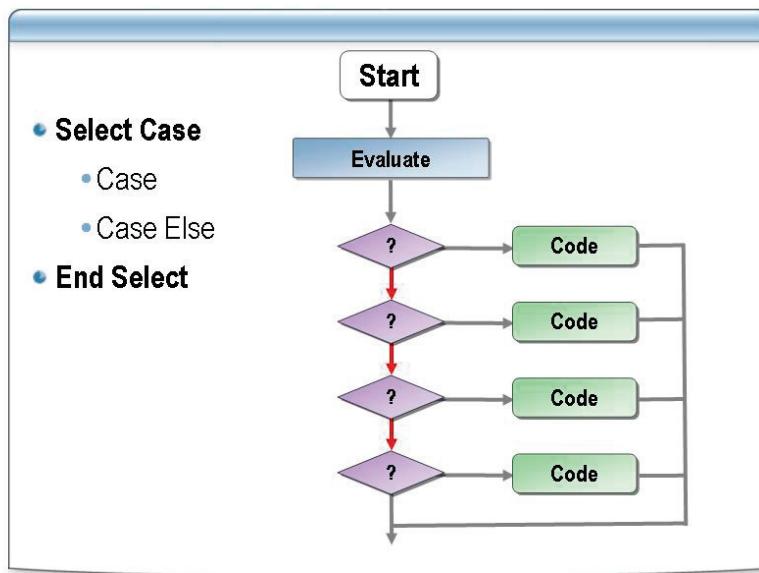
```
If MyVar = 5 Then
    WScript.Echo "MyVar = 5"
    WScript.Echo "Reseting MyVar"
    MyVar = 0
End If
```

In this example, three lines of script are executed if the condition evaluates to **True**. These lines are indented for clarity. Indentation is particularly beneficial when several conditional statements are nested, as the following example shows.

```
If MyVar = 5 Then
    WScript.Echo "MyVar = 5"
        If MyVar2 = 10 Then
            WScript.Echo "MyVar2 = 10"
        End If
    End If
```

If MyVar contains the value 5, the first message is echoed to the screen. Then, a second test is performed on a second variable. If that also returns **True**, a second message is displayed. An **End If** statement is required for each **If** statement.

Conditional Statements: Select Case...End Select



Although you can rely exclusively on the **If** statements described in the previous topic for conditional coding, there is an alternative. You can use the **Select Case** structure. Any **Select Case** structure can be rewritten by using **If** statements, and any **If** statements can be rewritten by using the **Select Case** structure. However, there are times when one structure is more appropriate than the other, because one or the other is easier to code, read, or debug.

Select Case

The **Select Case** statement works with a single condition that is evaluated once in the first line of the statement. The result of the expression is then compared with the values for each **Case** statement listed below the original **Select Case** statement. If there is a match, the block of statements associated with that **Case** statement is executed, as the following example shows.

```
Select Case MyVar
    Case "0"
        WScript.Echo "MyVar = 0"
    Case "1"
        WScript.Echo "MyVar = 1"
    Case "2"
        WScript.Echo "MyVar = 2"
    Case Else
        WScript.Echo "Sorry value is out of range"
End Select
```

This script is clearer than the previous example, which uses the **If...Then...ElseIf** structure. You should note that substituting an **If...Then...ElseIf** structure with a **Select Case** structure only works in this example because the same condition is being tested all

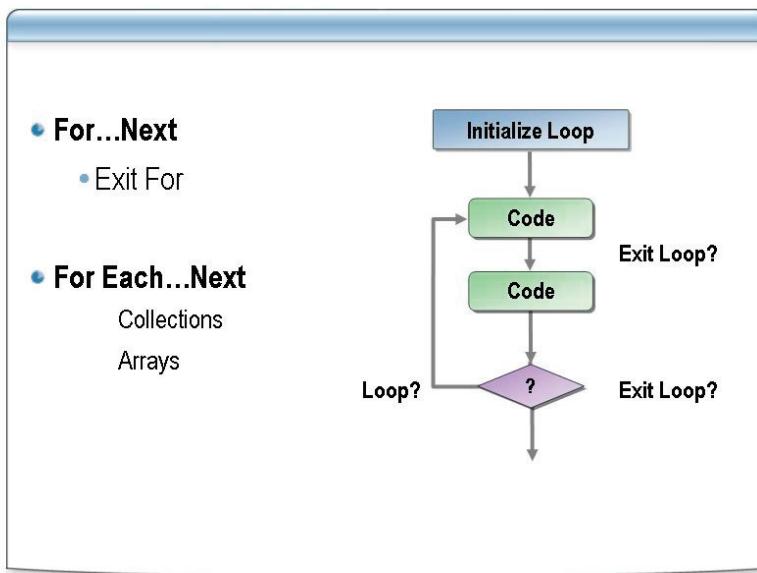
MCT USE ONLY. STUDENT USE PROHIBITED

the way down the **ElseIf** tree. If more than one condition is being tested, the **If...Then** structure is more appropriate.

End Select

You can nest **Select Case** statements. Like the **If** statement, each nested **Select Case** statement must have a matching **End Select** statement.

Looping: For...Next



As well as writing decision-making code, you may want to write code that runs multiple times in each execution of your script.

For...Next

You should use the **For...Next** loop to execute one or more statements a specific number of times. The number of times that the statements are executed is determined by the counter variable of the **For...Next** loop, which is set up when the loop is initialized. This counter is then increased or decreased with each repetition of the loop.

The following example shows the syntax of the **For...Next** construct.

```
For counter = start to finish [Step n]
    action(s)
Next counter
```

For example, the following code illustrates a loop that runs 10 times.

```
For iCounter = 1 To 10
    WScript.Echo "This is loop number " & iCounter
Next
```

The variable **iCounter** is set to 1 when the loop is initialized. The **WScript.Echo** statement is then executed. When the **Next** statement is reached, the variable automatically increases by one and the loop is executed again. This continues until the value of **iCounter** is greater than 10. Then, the loop is exited, and execution continues at the line after the **Next** statement. By default, **iCounter** will increase by one each time around the loop. You can modify this default behavior by using the **Step** statement, as the following example shows.

```
For iCounter = 1 to 10 Step 2
    WScript.Echo "Odd number " & iCounter
Next
```

This **Step** statement results in only five iterations of the loop. As the loop runs, iCounter has the value of one, three, five, seven, and nine.

You can also use the **Step** statement to decrease a counter simply by the addition of a minus sign (-).

```
For iCounter = 10 to 2 Step -2
    action(s)
Next
```

In this example, the loop is iterated five times, starting at the value of 10 and finishing with the value of 2.

You can optionally exit a loop at any time before it finishes by using the **Exit For** statement, as the following example shows.

```
vInput = InputBox ("Enter the Value to stop the loop at")
iStop = CInt(vInput)
For iCounter = 1 to 10
    WScript.Echo "Current iCounter Value = " & iCounter
    If iStop = iCounter Then Exit For
    WScript.Echo "Around we go again"
Next
WScript.Echo "Loop exited at value of " & iCounter
```

This script uses a variable called vInput that takes a value from the user of the script. This value is then converted into an integer and stored in the iStop variable. A **For...Next** loop is then set up and the current value of the counter variable iCounter is echoed to the screen. A test is then performed to see whether iCounter equals iStop. If it is equal, the loop is exited. Otherwise, a message is echoed, and the loop continues. After the loop is exited, a message is echoed that displays the value of iCounter.

For Each...Next

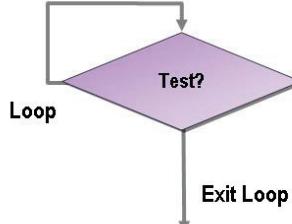
The **For Each...Next** structure is used to iterate through the items in a collection of objects or for each element of an array. This is useful when you do not know how many elements there will be in the collection before running the script, as the following example shows.

```
For Each File in objFiles
    WScript.echo File.Name
Next
```

This code example (also known as a snippet) contains a collection object called **objFiles**. Collection objects usually contain multiple items, so the **For Each...Next** structure is used to loop through each item in the collection. In this example, a collection of files is looped through, and the name of each file is displayed. After all of the items in the collection have been displayed, the loop is exited.

Looping: Do...Loop

- **Do...Loop**
 - Do While...Loop
 - Do Until...Loop
 - Do...Loop While
 - Do...Loop Until
- **Exit Do**



You may want to execute code in a loop without knowing how many times the loop must run when you are writing the code. Additional looping structures are available to you when you encounter this situation.

The **Do...Loop** structure iterates a number of script statements in a similar manner to the **For...Next** structure. The main difference between the **Do...Loop** structure and the **For...Next** structure is that the **Do...Loop** structure does not require you to know the number of times the loop must be run when you write the code. There are two basic forms of this structure: **Do While...Loop** and the **Do Until...Loop**.

Do While...Loop

This loop repeatedly executes a statement while a condition is true. The condition is evaluated each time that the loop begins.

```
Do While vMyVar < 100
    vMyVar = vMyVar + 1
    WScript.Echo "vMyVar = " & vMyVar
Loop
```

Do Until...Loop

This statement repeats the script in the loop until a condition becomes true. The previous examples can be converted into the **Do Until...Loop** structure by making the following changes to the script test conditions.

```
Do Until vMyVar >= 100
    vMyVar = vMyVar + 1
    WScript.Echo "vMyVar = " & vMyVar
Loop
```

Note: It is possible to rewrite a **Do While** structure as a **Do Until** structure by negating the condition, as shown above.

Do...Loop While

You can check the expression that represents the exit condition at the end of the **Do While** loop, rather than at the beginning, as the following example shows.

```
Do
    vMyVar = vMyVar + 1
    WScript.Echo "vMyVar = " & vMyVar
Loop While vMyVar < 100
```

Checking the condition at the end of the loop, rather than at the beginning, forces the loop to execute at least once. Imagine that the condition is already satisfied when the loop begins. If you check the condition at the beginning of the loop, the code inside the loop does not execute at all.

However, if you check the condition at the end of the loop, even if the condition is already met, the code inside the loop runs once. If, by the end of the loop, the condition is still satisfied, the loop is finished. However, note that the code inside the loop may alter the state of the condition. The loop continues until the condition is satisfied once more.

Do...Loop Until

As with the **Do While** loop, you can check the expression that represents the exit condition at the end of the **Do Until** loop, rather than at the beginning, as the following example shows.

```
Do
    vMyVar = vMyVar + 1
    WScript.Echo "vMyVar = " & vMyVar
Loop Until vMyVar >= 100
```

Exit Do

The **Exit Do** statement is useful for exiting a loop if another condition is met while the loop is executed, as the following example shows.

```
vInput = InputBox ("Enter the Value to stop the loop at")
iStop = CInt(vInput)
Do While iCounter < 10
    iCounter = iCounter + 1
    WScript.Echo "Current iCounter Value = " & iCounter
    If iStop = iCounter Then Exit Do
    WScript.Echo "Around we go again"
Loop
WScript.Echo "Loop exited at value of " & iCounter
```

Built-in Functions

- **Date and Time Functions**
 - Now, Time, Date, DateDiff
- **String Manipulation Functions**
 - LCase, UCase
- **Mathematical Functions**
 - Rnd, Round

Visual Basic, Scripting Edition includes various special functions that can be used in script to provide features that are either difficult to write yourself or required by many scripts. These special functions are known as built-in functions. You can use them in expressions with variables and other values to calculate and return values.

Date and Time Functions

There are functions that make handling dates and times in your script easy.

Now

The **Now** function returns the current date and time on the computer system clock, as the following example shows.

```
WScript.Echo Now
```

This function returns a value such as 06/26/2007 15:23:41.

The format of the return value depends on your regional settings.

Time

The **Time** function returns a **Variant** of subtype **Date** that indicates the current time on the computer system clock, as the following example shows.

```
WScript.echo Time
```

This function returns a value such as 15:24:27.

Date

The **Date** function returns the current date on the computer system clock, as the following example shows.

```
WScript.Echo "Today's Date is : " & Date
```

This function returns a value such as 06/26/2007.

DateDiff

The **DateDiff** function returns the number of intervals between two dates.

```
vDate = DateDiff("d", Date, "01/01/3000")
WScript.Echo "Days until the next Millennium: " & vDate
```

The "d" argument indicates that the function must return a number that represents the number of days between the two dates. Other valid argument values include:

- "yyyy" for the number of years between the two dates.
- "m" for the number of months between the two dates.
- "h" for the number of hours between the two dates.

String Manipulation Functions

Visual Basic, Scripting Edition provides several string manipulation functions because you may want to manipulate and compare strings.

These functions return a string value that has been converted to lowercase (**LCase**) or uppercase (**UCase**). This is useful when you compare one string to another string and the script writer does not know which case will be used.

When users type one of the strings, they may mix the case of their input. This can cause any test to fail, because a string comparison of a lowercase and an uppercase letter always returns **False**. By using the **LCase** or **UCase** functions, the script writer is assured that the comparison is performed by using lowercase or uppercase only.

```
vName = InputBox("Please enter your home country.")
vName = LCase(vName)
Select Case vName
    Case "uk"
        WScript.Echo "Welcome to the UK!"
    Case "us"
        WScript.Echo "Welcome to the United States!"
    Case "germany"
        WScript.Echo "Welcome to Germany!"
    Case "france"
        WScript.Echo "Welcome to France!"
    Case Else
        WScript.Echo "Sorry, try again! "
End Select
```

Mathematical Functions

Visual Basic, Scripting Edition provides several mathematical functions because you may want to perform mathematical manipulations.

Rnd

The **Rnd** function returns a random number between zero and one, to seven decimal places. This function only works correctly if you have initialized the random number generator before calling the **Rnd** function. To initialize the random number generator, use the **Randomize** statement.

```
Randomize  
WScript.Echo Rnd
```

By multiplying the value that the **Rnd** function returns, you can generate random numbers within a given range.

```
Randomize  
WScript.Echo Rnd * 100
```

This example generates a random number between zero and 100, including fractions of a number.

Round

If a script requires you to use whole numbers, you can use the **Round** function to convert a number to the nearest integer.

```
Randomize  
vRnd = Rnd * 100  
WScript.Echo Round(vRnd)
```

Note: Many more functions are available in VBScript. For the complete list of functions, see the VBScript documentation.

Lesson 5: Procedures

- Sub and Function Procedures
- Passing Data to Procedures
- Scope and Lifetime

You have learned how to write scripts by using the Visual Basic, Scripting Edition language. When you write your code, you may encounter a situation where you must run similar blocks of code multiple times. When this is the case, you can encapsulate this code into reusable blocks, called procedures.

In this lesson, you will learn about **Sub** and **Function** procedures. You will also learn how procedures and variables interact to create the concepts of scope and lifetime.

Objectives

After completing this lesson, you will be able to:

- Use subroutine and function procedures in scripts.
- Control the scope and lifetime of script-level and procedure-level variables.

Sub and Function Procedures

- **Sub Procedures**
- **Function Procedures**
 - Returns a result
- **Exiting a Procedure**

In Visual Basic, Scripting Edition, there are two kinds of procedures: **Sub** procedures, sometimes called subroutines, and **Function** procedures.

Procedures are used to encapsulate code blocks that you require often.

Sub Procedures

Sub procedures contain code that you can reuse in your scripts many times without having to rewrite the code every time that it is required. Instead, you write the code once inside the procedure, and then call the procedure from other places in your code to indicate that the contents of the procedure must run at that point.

The following example shows how you can use **Sub** procedures in your scripts. Note that the **Sub** procedure named **Status** is called twice. The **Call** statement is optional.

```
Dim sStatusMessage
Call Welcome
sStatusMessage = "Starting..."
Call Status
sStatusMessage = "Ending..."
Call Status
Call Farewell
Sub Welcome()
    WScript.Echo "Hello"
End Sub
Sub Status()
    WScript.Echo sStatusMessage
End Sub
Sub Farewell()
    WScript.Echo "Goodbye"
End Sub
```

Note: The **Call** statement is optional when calling a procedure. However, if you use the **Call** statement to call a procedure that requires arguments, you must enclose the argument list in parentheses. If you omit the **Call** statement, you must also omit the parentheses around the argument list.

Function Procedures

Function procedures are very similar to **Sub** procedures. You write the code once inside the procedure, and then simply call the procedure from other places in your code. The only important difference between a **Sub** and a **Function** procedure is that the latter returns a result to the script that called it, but the former does not.

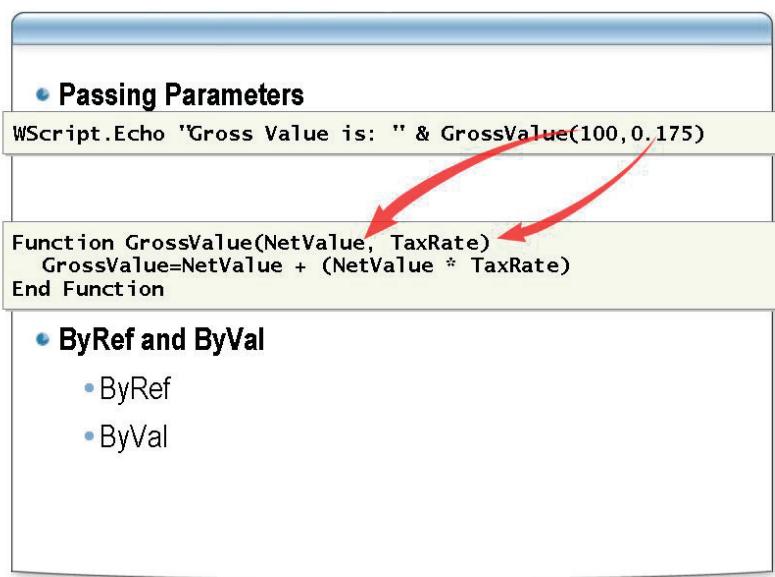
Consequently, in most cases, you use the **Function** procedure as part of a larger expression, because the call to this procedure results in a value being returned in place of the **Function** name. The following example illustrates how the function call returns a value that you can use in any expression.

```
Dim dDate
dDate=Tomorrow
WScript.Echo dDate
Function Tomorrow()
    Tomorrow= DateAdd("d",1,Date())
End Function
```

Exiting a Procedure

If a test such as an **If...Then** structure determines that the procedure must be exited, the procedure is exited by using the exit statement. When you reach the end of the procedure, the **End Sub** statement returns the execution of the script to the point in the script immediately after the procedure was called.

Passing Data to Procedures



The code examples shown so far do not enable you to pass any additional data to the procedures when you call the procedures. However, passing additional data that the procedure works with is a useful and common practice.

Passing Parameters

To pass parameters to your procedure, you must define the parameters that the procedure will receive when you write it. You do this by providing a comma-separated list of parameter names inside the parentheses for the procedure, as the following example shows.

```
Function GrossValue(NetValue, TaxRate)
    GrossValue=NetValue + (NetValue * TaxRate)
End Function
```

You must pass the values of the parameters to the procedure when you call the procedure. The following example calls the **GrossValue** function and passes a value of 100 for the **NetValue** parameter and 0.175 for the **TaxRate** parameter.

```
WScript.Echo "Gross Value is: " & GrossValue(100,0.175)
```

ByRef and ByVal

You pass data into procedures by using variable names that contain the relevant data rather than the literal values themselves. By default, variables are passed to procedures by reference. As a result, if you make any changes to the parameters in the **Sub** procedure or **Function** procedure, you alter the original variable in the calling code as well.

Although this is the default behavior in Visual Basic, Scripting Edition, you can explicitly qualify each argument by using the **ByRef** keyword, as the following example shows.

```
Function GrossValue(ByRef NetValue, ByRef TaxRate)
    GrossValue=NetValue + (NetValue * TaxRate)
End Function
```

Another method for parsing an argument is by value. With this approach, any argument passed to the **Sub** procedure or **Function** procedure is a copy of the original variable. If you make any changes to the parameters in the procedure, the variable in the calling code is not changed.

To specify this option, you use the **ByVal** keyword in front of each argument, as the following example shows.

```
Function GrossValue(ByVal NetValue, ByVal TaxRate)
    GrossValue=NetValue + (NetValue * TaxRate)
End Function
```

You will note that in the examples used so far, **ByRef** and **ByVal** result in identical script behavior. This is because the procedures themselves do not internally modify the values of the parameters that have been passed to them. However, this is not always the case. For example, the following code, which uses **ByRef**, results in a message of “Two times 60 is 60,” which is obviously not intended.

```
Dim iNumber, iResult
iNumber=30
iResult = Doubler(iNumber)
WScript.Echo "Two times " & iNumber & " is " & iResult
Function Doubler(ByRef InputNum)
    InputNum = 2 * InputNum
    Doubler = InputNum
End Function
```

The **InputNum** parameter points to the same memory location as the **iNumber** variable, because **InputNum** was passed by using **ByRef**. Therefore, any changes made to the **InputNum** parameter result in those changes being applied to the **iNumber** variable.

The following example is almost identical. However, the parameter is passed by using **ByVal**. This results in a message of “Two times 30 is 60,” which is intended.

```
Dim iNumber, iResult
iNumber=30
iResult = Doubler(iNumber)
WScript.Echo "Two times " & iNumber & " is " & iResult
Function Doubler(ByVal InputNum)
    InputNum = 2 * InputNum
    Doubler = InputNum
End Function
```

Scope and Lifetime

- **Script-Level Variables**
 - Dim
 - Private
 - Public
- **Procedure-Level Variables**
 - Dim

The scope of a variable defines *where* you can use it in your script. The lifetime of a variable determines *how long* that the variable exists in memory and *how long* it is available to your script.

The scope and lifetime of a variable are determined by where in the script the variable is declared and the keyword used to declare the variable.

Script-Level Variables

When declaring a variable in the main body of the script, it can be referenced from anywhere else in the script. Variables can be used by other statements in the main body of the script and also in any procedures that are called. These variables are known as script-level variables, and they have script-level scope.

You can declare script-level variables by using either the **Dim** keyword or the **Private** keyword. These keywords are synonymous when used to declare script-level variables.

You can also use the **Public** keyword to define variables that are available outside the scope of the script itself. However, you can only do this if you are going to use the script with other scripts that can reference these variables, such as .wsf files. If this is not the case, then script-level variables that are declared with the **Public** keyword behave identically to variables that are declared with the **Private** and **Dim** keywords.

The names of script-level variables must be unique throughout the script. The lifetime of a script-level variable is from the time when the variable is declared to the time when the script terminates.

Procedure-Level Variables

When you declare a variable in a procedure, only script within that procedure can access or change the value of the variable. It has local scope and is called a procedure-level variable. A procedure-level variable exists only when the procedure is running. When the procedure is finished, the variable is destroyed. The variable has procedure-level lifetime.

You can have local variables of the same name in several different procedures, because each variable is recognized only by the procedure in which it is declared. You usually undertake declaration of procedure-level variables at the start of the procedure itself. This is a good practice, because it makes it easier to track the variables.

It is also a good practice to declare variables with the smallest scope possible. This prevents you from having to make a long list of variable names that is visible to the whole script, and helps you to minimize the memory that variable names consume.

Note: You cannot declare procedure-level variables by using the **Public** or **Private** keywords. You must use the **Dim** keyword to declare procedure-level variables.

Lesson 6: Script Layout

- **Script Sections**
- **Using a Template**
- **Best Practices for VBScript**

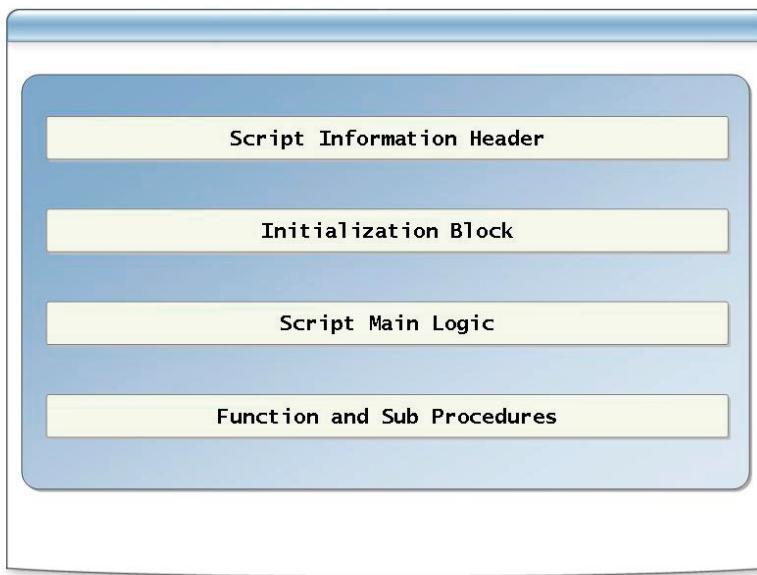
You can organize your scripts in a manner that makes them easy to handle, read, and debug. Saving code examples as templates also makes it easier when writing new scripts. This lesson will teach you about organizing the code in your scripts. You will also learn about best practices for coding.

Objectives

After completing this lesson, you will be able to:

- Use script sections to apply logical layouts to your script code.
- Use script templates.
- Apply best practices when writing code in Visual Basic, Scripting Edition.

Script Sections



As your scripts become larger and more complex, it is increasingly important to have a standard and logical layout for your scripts. There are several recommended standards, but you are free to decide whether you want to use them or not. However, remember that ultimately these standards are there to make working with script easier.

One simple recommended standard is to organize the script into four main blocks:

- Script information header
- Initialization block
- Main logic
- **Function and Sub** procedures

Script Information Header

The script information header contains information about the script itself. It is not required for the execution of the script because it is commented out. The information in the script information header is there to help whoever is reading the script to identify what the script is, who wrote it, and what it should do. An example of a script header is shown below.

```
'=====
'
' LANG      : VBScript
' NAME      : MyScript.wsf
' AUTHOR    : My Name
' DATE      : 06/26/2007
' Description : Demonstration Script Header
' COMMENT   : Add comments here
'
' KEYWORDS   : Indexing Keywords here
'=====
```

The use of the **KEYWORDS** field is to enable an indexing engine to search and retrieve all related scripts.

Initialization Block

The next block of the script is the initialization block. In this section, script-level variables, constants, and object names are declared. This provides a single reference area that contains all variable names, which you can search when working on the script, as the following example shows.

```
'=====
'Variable Declarations

Dim nNetValue
'Used to store the net value of a product

Dim nGrossValue
'Used to store the gross value of a product

Dim nTaxRate
'Used to store the tax rate that must be applied to net value

'=====
```

Script Main Logic

The script main logic section is where you type the main body of the script. Note that this section is not always the biggest part of the script. If the script uses a lot of procedures to complete its task, this section might be no more than a list of procedure calls and a **WScript.Quit** statement. The following is an example of the main body of a script.

```
'=====
' Main Body

' Set the net value
nNetValue = 100

' Set the tax rate
nTaxRate = 0.175

' Call the GrossValue function
nGrossValue = GrossValue(nNetValue, nTaxRate)
```

```
' Display the result to the user
WScript.Echo "The Gross Value is: " & nGrossValue

'=====
```

Function and Sub Procedures

This section contains all of the procedures that are called from within the main script logic. They are not executed in a top-down manner, but are executed when called. You can also store procedures that are not currently used in the script, but can be used at some point in the future. If the main script logic does not call the procedure, it will be ignored.

The following example shows the final script, complete with all of the sections.

```
'=====
'
' LANG      : VBScript
' NAME      : MyScript.wsf
' AUTHOR    : My Name
' DATE      : 06/26/2007
' Description : Demonstration Script Header
' COMMENT   : Add comments here
'
' KEYWORDS   : Indexing Keywords here
'=====
'
'=====

'Variable Declarations

Dim nNetValue
'Used to store the net value of a product

Dim nGrossValue
'Used to store the gross value of a product

Dim nTaxRate
'Used to store the tax rate that must be applied to net value

'
'=====
'
' Main Body

' Set the net value
nNetValue = 100

' Set the tax rate
nTaxRate = 0.175

' Call the GrossValue function
nGrossValue = GrossValue(nNetValue, nTaxRate)

' Display the result to the user
WScript.Echo "The Gross Value is: " & nGrossValue

'
'=====
'
' Procedures

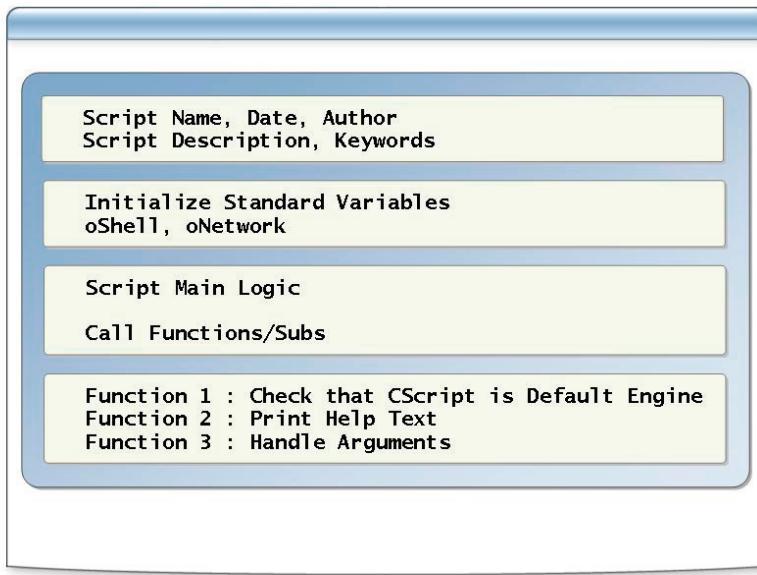
' Returns a gross value based on a net value and a tax rate
```

```
Function GrossValue(nNet, nRate)
    GrossValue=nNet + (nNet * nRate)
End Function

'=====
```

Note: You can use another type of layout with the main script logic at the end of the script. This defines the procedures directly after the initialization of the variables. The layout you use makes little difference to the execution of the script. Ultimately, it is a matter of personal preference.

Using a Template



After you have defined the basic script layout, you can create a template that you can reuse as the basis for future scripts. This template can be as simple or as complex as required. Fundamentally, it should help you to create scripts in a standard way.

The template is not restricted to the script outline. If there are script segments or procedures that are regularly used, such as common variable definitions or usage display procedures, you could also include these as part of the template.

A typical template might have separate sections for:

- Script name, description, date, author, and keywords.
- Script-level variable and constant declarations.
- The main body of the script.
- Procedures.

Some Integrated Development Environment (IDE) tools such as PrimalScript enable the user to define a template as part of creating a new script file. If you are using one of these tools, it is worth spending a few minutes setting up this feature because it can save you a lot of typing.

Note: See the lab later in this module, which uses an example script template that you can customize to meet your requirements.

Best Practices for VBScript

- **Using Option Explicit**
- **Giving Variables Meaningful Names**
- **Commenting Your Script**
- **Using a Consistent Naming Convention**
- **Creating a Useful Script Store**

Coding by using Visual Basic, Scripting Edition is flexible. You can achieve the same results in many different ways. However, there are some best practices to which you should adhere.

Using Option Explicit

Always set **Option Explicit** in your script, because you are required to declare all of your variable names. This reduces bugs caused by mistyped variable names.

Giving Variables Meaningful Names

When possible, give the variables and objects in your script meaningful and user-friendly names. This practice makes it easier to understand a script.

Commenting Your Script

Commenting your script not only makes it easier for other people to understand, but it also helps you if you want to reuse a script after many months.

Using a Consistent Naming Convention

Use a consistent naming convention, even if you choose not to use the Hungarian naming convention. This makes your scripts easy to understand over time.

Creating a Useful Script Store

As you learn script, you will come across many examples of script that solve a specific problem. When you come across these examples, cut and paste them into a script template and store them in a folder that you can then index. By doing this, you can build up a store of examples that you can quickly copy into your script when required.

Lab: Script Logic



- **Exercise 1**
Creating Script Templates
- **Exercise 2**
Inserting and Creating Code Snippets
- **Exercise 3**
Adding Constants, Variables, Loops, and Conditional Structures to Scripts
- **Exercise 4**
Adding Procedures to Scripts

After completing this lab, you will be able to:

- Create script templates.
- Insert and create code snippets.
- Add constants, variables, loops, and conditional structures to scripts.
- Create scripts that use intrinsic Visual Basic, Scripting Edition constants and procedures.

Estimated time to complete this lab: 30 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine.
- Start the 2433B-VISTA-CL1-03 virtual machine.
- Log on to the 2433B-VISTA-CL1-03 virtual machine as **FOURTHCOFFEE\Administrator** using the password **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to use script templates and code snippets, so that other administrators can easily understand and modify your scripts. You want to create a standard script template and use it to create a script that tests a user's name against a known list.

Exercise 1

Creating Script Templates

In this exercise, you will use PrimalScript to create an administrative script template.

The principal tasks for this exercise are as follows:

- To create a PrimalScript template.
- To save the script template.
- To create a new script based on your template.

Tasks	Supporting information
1. To create a PrimalScript template.	<ul style="list-style-type: none">• Start PrimalScript.• Using the Script Files template list, create a new VBScript file.• Edit the template to add a Keywords section in the header, and separate sections for Variable Declarations, Main Body, and Procedures.
2. To save the script template.	<ul style="list-style-type: none">• Save the template as C:\Program Files\SAPIEN\PrimalScript 4.1 Classroom\Templates\File Templates\Script Files\Fourthcoffee Script.vbs.
3. To create a new script based on your template.	<ul style="list-style-type: none">• Create a new script file using the Fourthcoffee Script template.

Questions

Q: Which feature of Visual Basic, Scripting Edition is case-sensitive?

Q: What is wrong with the following code?

```
WScript.Echo "Message to _  
& vUserName _  
& Please make sure that you remember _  
& to log off or lock your workstation _  
& if you leave it unattended, Thanks"
```

Exercise 2

Using and Creating Code Snippets

In this exercise, you will use the code snippets feature of PrimalScript. You will also create a new code snippet, which you will use in the next exercise.

The principal tasks for this exercise are:

- To use a code snippet.
- To create a new code snippet.

Tasks	Supporting information
1. To use a code snippet.	<ul style="list-style-type: none"> • Make sure that the cursor is under Main Body. • Open the Right Nexus window, and click the Snippets Browser tab. • Expand VBScript snippets, and insert the IfElse snippet, to create an empty If...Then...Else statement.
2. To create a new code snippet.	<ul style="list-style-type: none"> • Modify the If...Then...Else statement so that it includes an Elself snippet. • Highlight the edited If...Then...Elself statement and copy it to the clipboard. • In the Snippets Browser, create a new snippet called IfElself. • Save and close the snippet.

Questions

Q: When assigning data to a variable, how do you specify that the data represents a date or time?

Q: The following script causes a compilation error. What is the problem?

```
If MyVar = 5 Then
    WScript.Echo "MyVar = 5"
If MyVar = 7 Then
    WScript.Echo "MyVar = 7"
End If
```

Exercise 3

Adding Constants, Variables, Loops, and Conditional Structures to Scripts

In this exercise, you will use the new template that you created in Exercise 1, and the new code snippet that you created in Exercise 2. You will then use variables and constants to manipulate data that the script user enters.

In addition, you will create decision-making and looping structures that will compare the user's name with a list of pre-approved administrators. You will enable users to enter their name up to three times. If users enter an administrator's name, the script continues to run. If users fail to enter a valid name three times in a row, they are notified that they have not been validated.

The principal tasks for this exercise are:

- To complete the script information header.
- To declare constants and variables.
- To use a decision-making structure.
- To use a **Do...Loop**.
- To test the script.

Tasks	Supporting information
1. To complete the script information header.	<ul style="list-style-type: none"> • In the new file that you created in Exercise 1, modify the header to include the following information: <ul style="list-style-type: none"> • Name: Validate.vbs • Author: Add your name here • Comment: Validates administrative users • Keywords: Loops, variables, constants
2. To declare constants and variables.	<ul style="list-style-type: none"> • Under Variable Declarations, declare the following variables: <ul style="list-style-type: none"> • iLoopCount • sUserName • iAnswer • Declare the following constants: <ul style="list-style-type: none"> • ADMIN_1, with a value of ADRIAN LANNIN • ADMIN_2, with a value of JO BERRY
3. To use a decision-making structure.	<ul style="list-style-type: none"> • Under Main Body, set iLoopCount to be iLoopCount + 1. • On the next line, use the Snippets Browser to insert the IfElseif snippet. • Edit the If...Then...Elseif statement to resemble the following code:

Tasks	Supporting information
	<pre> If iLoopCount > 3 then WScript.Echo "Maximum attempts exceeded!" & vbCrLf ↵ & "This script will now end..." WScript.Quit ElseIf iLoopCount > 1 then iAnswer = MsgBox("You must be an Administrator to run ↵ this script." - & vbCrLf & "Do you want to try again?", vbYesNo) If iAnswer = vbNo then WScript.Quit End If End If </pre> <ul style="list-style-type: none"> Below the final End If statement, use an InputBox statement with the text "<i>Please enter your name (Firstname Lastname)</i>" to gather user input and assign it to the sUserName variable. Save the script as E:\Labfiles\Starter\Validate.vbs.
4. To use a Do...Loop.	<ul style="list-style-type: none"> Create a new line after Main Body. Use the Snippets Browser to insert a LoopWhile snippet. Delete the highlighted <statement>. Move the Loop While <condition> code to immediately follow the InputBox statement. Edit the Loop While statement, so that the code loops while UCase(sUserName) is not equal to ADMIN_1 and ADMIN_2. Immediately below the Loop statement, type the following code: <pre> WScript.Echo "You have been validated!" & vbCrLf _ & "The script will now continue..." </pre> <ul style="list-style-type: none"> Save the script.
5. To test the script.	<ul style="list-style-type: none"> Run the script in PrimalScript, using the name "Yan Li"; try using this name three times to check for the "maximum attempts exceeded" message. Run the script in PrimalScript, using the name "Jo Berry" to check for the "successful validation" message.

Note: The line continuation character (↵) indicates that the text following it should be written on the same line as the code that precedes it.

Questions

Q: What is the primary data type used in Visual Basic, Scripting Edition?

Q: What is the difference between a constant and a variable?

Exercise 4

Adding Procedures to Scripts

In this exercise, you will create a reusable function that carries out the process of validating a user against a known list of administrators. The function will return either **True** or **False**, thereby making it easy to use in the calling code.

The principal tasks for this exercise are:

- To create a **Validate** function.

Tasks	Supporting information
1. To create a Validate function.	<ul style="list-style-type: none"> Create a new line after Procedures. Use the Snippets Browser to insert a Function snippet. In the Function snippet, change <functionname> (<arguments>) to Validate(sUser). In the Function snippet, use the Snippets Browser to replace <statements> with an If...Then...Else statement. Edit the If...Then...Else statement to resemble the following code: <pre>If UCase(sUser) = ADMIN_1 Or UCase(sUser) = ADMIN_2 then Validate = True Else Validate = False End if End Function</pre> <ul style="list-style-type: none"> Under Main Body, modify the Loop statement to read Loop While Not Validate(sUserName). Save the script. Run the script in PrimalScript, using the name “Yan Li.” Verify that you are notified that Yan Li is not an administrator. Run the script in PrimalScript, using the name “Adrian Lannin” to check for the “successful validation” message.

Questions

Q: What would be the result of the following?

$$(10 - 5 + 10) * 2 - (4 * 5 / 2)$$

Q: What are the main benefits of using procedures in scripts?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

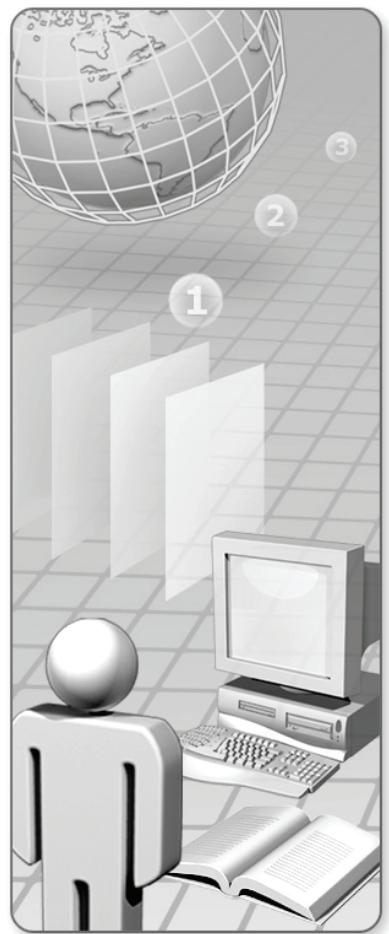
After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-03 virtual machines and discard any changes.

Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

Module 4: Error Handling and Debugging

Table of Contents

Module Overview	4-1
Lesson 1: Error Handling	4-2
Lesson 2: Debugging	4-9
Lab: Error Handling and Debugging	4-15



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Error Handling**
- **Debugging**

It is inevitable that you will sometimes encounter errors in your scripts. When errors occur, you must know how to deal with them.

This module discusses error handling. It also discusses some approaches to debugging, including the use of a debugger application.

Objectives

After completing this module, you will be able to:

- Describe the types of errors that may be present in code written in Microsoft® Visual Basic®, Scripting Edition (VBScript).
- Write code that handles run-time errors.
- Use the Microsoft Script Debugger to locate and fix logic errors.

Lesson 1: Error Handling

- **Types of Errors**
- **Handling Run-Time Errors**
- **The Err Object**
- **Setting an Error Trap**

Errors that prevent your code from performing as intended will be present in some of the code that you develop. Even professional programmers sometimes encounter errors in their code.

This lesson describes the various types of errors that can occur in Visual Basic, Scripting Edition. It also describes some approaches that you can use to deal with run-time errors.

Objectives

After completing this lesson, you will be able to:

- Describe the types of errors that you can encounter when you write scripts by using Visual Basic, Scripting Edition.
- Write code that addresses the possible run-time errors in your script.
- Use the **Err** object in Visual Basic, Scripting Edition to display error messages and clear the most recent error.
- Write code that captures and addresses common errors in your script.

Types of Errors

- **Syntax Errors**
- **Run-Time Errors**
- **Logic Errors**

There are three categories of programming errors: syntax, run-time, and logic errors.

Syntax Errors

Syntax errors result from code that is written incorrectly, such as a mistyped keyword, the omission of required punctuation, or an incorrect construct such as a **Next** statement without a corresponding **For** statement.

Visual Basic, Scripting Edition detects these errors when the scripting engine compiles your script. If syntax errors are found, the script cannot start. Instead, you are notified that a compilation error has occurred. Visual Basic, Scripting Edition reports detailed information about the syntax error, such as its location in your script and a description of the type of error.

Run-Time Errors

Run-time errors occur when your code is syntactically correct, but a statement attempts an operation that is impossible to execute. For example, attempting a calculation that includes a division by zero, or attempting to save a file to the floppy disk drive when no disk is present, results in a run-time error. For this type of error, you must create an error trap. You will learn how to create an error trap in Setting an Error Trap later in this module.

Logic Errors

Logic errors occur when code does not perform in the way that you intended. Your code may be syntactically correct, and it may not generate any run-time errors, but it does not produce the results that you anticipated. You must test and debug your code to determine why the results are different from those that you intended. You will learn how to debug scripts in Lesson 2, Debugging, later in this module.

Handling Run-Time Errors

- **Creating Robust Scripts**
- **Enabling Graceful Exits**
- **Implementing Error Handling**

By detecting run-time errors, you can make your script responsive to common run-time errors.

Creating Robust Scripts

Scripts that trap run-time errors can handle common user errors without stopping program execution. When you anticipate common errors such as trying to open a nonexistent file, and protect your code against these errors, your program is less likely to stop running.

Enabling Graceful Exits

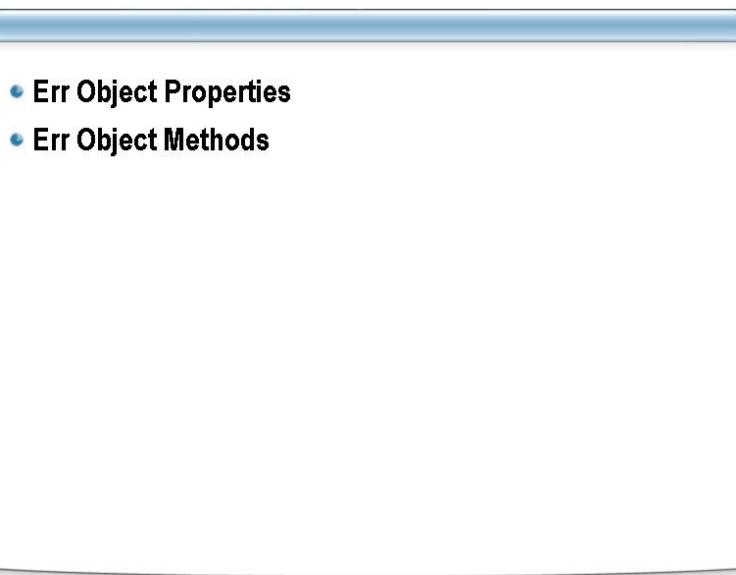
Sometimes the error-handling code in your script cannot resolve a run-time error. When this happens, your code can still perform other actions such as closing any open data files and saving data that would be lost otherwise. In addition, you can inform the user that an error has occurred and give the user more descriptive messages than they may otherwise receive.

Implementing Error Handling

To implement error handling, you must anticipate the lines of code where run-time errors may occur. In addition, you must anticipate the type of error that may occur. The process for implementing error handling is to:

- Set an error trap.
- Determine which error has occurred.
- Take some remedial action or exit the script gracefully.

The Err Object



- **Err Object Properties**
- **Err Object Methods**

Run-time errors can originate from the code that you write in Visual Basic, Scripting Edition. When a run-time error occurs, you can check the properties of the **Err** object to see what error has occurred.

Err Object Properties

The following table lists the most commonly used properties of the **Err** object.

Property	Description
Number	Returns a valid error number. Number is the default property of the Err object.
Description	The error message that corresponds to the Number property.

Err Object Methods

In addition to using the properties of the **Err** object, you can use its methods to control errors.

The following table lists the most commonly used methods of the **Err** object.

Method	Description
Clear	Clears all of the property settings of the Err object. Use this method after processing an error to set the Err object's properties to their original state.
Raise	Generates a run-time error. Use this method when you test your error-handling code.

Setting an Error Trap

- **On Error Resume Next**
- **Fixing the Error**

When Visual Basic, Scripting Edition encounters a run-time error, the default behavior is to halt code execution and display a message to the user of the script. The user often regards the message as cryptic, especially if the user has no experience of using Visual Basic, Scripting Edition or of programming. In addition, the user cannot choose to recover from the error.

On Error Resume Next

To modify the default way in which Visual Basic, Scripting Edition handles run-time errors, you can set a trap. After you have set the trap, you can override the way in which Visual Basic, Scripting Edition handles run-time errors. This enables your error-handling code to control the sequence of actions that is taken either to fix the error or to provide a safer, more graceful, and more meaningful exit for the user.

You set the error trap by using the **On Error Resume Next** statement. This statement enables your code to continue running.

If you use the **On Error Resume Next** statement, it is recommended that you trap potential errors after each line of code that you anticipate may cause possible errors. The following example shows that users can provide numerical input to the script. The script uses the `sError` variable to capture run-time errors that the user introduces.

Example

```
Option Explicit
On Error Resume Next

Dim intNumberOfTests, intTotalPoints, strError, intAnswer, numAverage

Do
```

```
strError = ""

' collect test information
intTotalPoints = InputBox("Enter the Total points scored.")
If intTotalPoints = "" then
    wscript.quit
End If

intNumberOfTests = InputBox("Enter the number of exams.")

If intNumberOfTests <> "" then
    ' calculate average
    numAverage = CDbl(intTotalPoints)/CInt(intNumberOfTests)

    Select Case err.number
        Case 0
            if CInt(intNumberOfTests) <> CDbl(intNumberOfTests) then
                strError = "Number of tests must be a whole number."
            elseif CInt( intNumberOfTests ) < 0 then
                strError = "Number of tests cannot be negative"
            else
                wScript.echo "Average score = " & nAverage
            end if
        Case 11
            strError = "Cannot divide by zero"
        Case 13
            strError = "Only numeric data is allowed"
        Case else
            strError = "An unexpected error occurred"
    End Select

    wscript.echo strError

End If
Loop While sError <> ""
```

Fixing the Error

The example above successfully traps the error. However, it does not attempt to fix the error; it merely enables a graceful exit. The following example illustrates an approach to fixing run-time errors:

- If errors occur, users can try again. The script displays customized error text, then displays a message box that asks users whether they want to attempt the calculation again.
- The script also uses the **Clear** method from the **Err** object. The **Err** object does not delete error information without explicit calls to the **Clear** method. To avoid generating invalid errors on subsequent script processing, you must call the **Clear** method after you have handled an error.

Example

```
Option Explicit
On Error Resume Next
```

```
Dim intNumberOfTests, intTotalPoints, sError, iAnswer, nAverage

Do
    sError = ""

    ' collect test information
    intTotalPoints = InputBox("Enter the Total points scored.")
    If intTotalPoints = "" Then
        wscript.quit
    End If

    intNumberOfTests = InputBox("Enter the number of exams.")

    If intNumberOfTests <> "" Then
        ' calculate average
        nAverage = CDbl(intTotalPoints)/CInt(intNumberOfTests)

        Select Case err.number
            Case 0
                If CInt(intNumberOfTests) <> CDbl(intNumberOfTests) Then
                    sError = "Number of tests must be a whole number."
                ElseIf CInt( intNumberOfTests ) < 0 Then
                    sError = "Number of tests cannot be negative"
                Else
                    wScript.echo "Average score = " & nAverage
                End If
            Case 11
                sError = "Cannot divide by zero"
            Case 13
                sError = "Only numeric data is allowed"
            Case Else
                sError = "An unexpected error occurred"
        End Select

        If Len( sError ) > 0 Then
            wscript.echo sError
            iAnswer = MsgBox("Do you want to try again?", vbYesNo)

            If iAnswer = vbNo Then
                WScript.Quit
            End If

            'without this line, division by zero error persists
            'even after a successful average
            err.Clear
        End If 'len( sError )
    End If
Loop While sError <> ""
```

Lesson 2: Debugging

- The Microsoft Script Debugger
- Debugging Techniques

The most difficult errors to find and fix are usually logic errors. However, it is inevitable that logic errors, sometimes referred to as bugs or glitches, appear in your code, especially as your code becomes more complex.

The Microsoft Script Debugger is an invaluable tool that helps you to debug your scripts. In this section, you will learn how to use the Microsoft Script Debugger.

Objectives

After completing this lesson, you will be able to:

- Use the Microsoft Script Debugger to help you find logic errors in your script.
- Apply debugging techniques to your scripts—using any debugging tool—to help you to find logic errors in your scripts.

The Microsoft Script Debugger

- Processing Errors in WSH Scripts
- Debugging Scripts

The Microsoft Script Debugger can help you to find logic errors in your code. You can use the debugger to:

- View the source code of the script that you are debugging.
- View and control script flow.
- View and change variable and property values.
- View the order of execution of procedures in a script.

Note: The Microsoft Script Debugger is available as a download for the Windows® XP and Windows® Server 2003 operating systems on the Microsoft Download Center. Search for the version supplied for Windows® NT 4.0 and later. Microsoft Script Debugger is deprecated in Windows Vista® operating system and Microsoft offers no support for the tool on that version of the operating system.

Processing Errors in WSH Scripts

The following describes the events that occur when a Windows Script Host (WSH) script is processed:

- The script is loaded into WSH and parsed.
Parsing involves reading code segments, analyzing them, and compiling them into an executable binary format.
- The browser reports any syntax errors that it finds during the parsing stage.
- After successfully parsing a section of script, WSH executes it.

Global or inline scripts that are not part of an event-handling **Sub** procedure or function are executed immediately.

Event-handling **Sub** procedures or functions and procedures that are called by other procedures are parsed immediately. However, they are not executed until they are triggered by an event or called by another procedure.

- If a run-time error occurs when a client script runs, an error message is displayed and the script that contains the error stops.

Debugging Scripts

If an error occurs in a script, WSH displays an error message indicating the error and the number of the line on which it is found. If you have the debugger installed on your computer, you are prompted about whether you want to debug the script. If you choose to debug the script, WSH starts the debugger and displays the current script in a read-only window.

After the script loads into the debugger, you can move through the document, set breakpoints, return to the document to run scripts, and step through the script. By carrying out this process, you can locate run-time and logic errors.

Note: You cannot edit the source code directly in the debugger. You must edit the original .vbs file and then rerun it.

Debugging Techniques

- **Setting Breakpoints and Stepping Through Code**
- **Viewing and Changing Variable Values**
- **Viewing the Call Stack**

The Microsoft Script Debugger provides the following features that you can use to debug your scripts to find logic errors:

- Breakpoints
- Line-stepping functions
- The Command window
- The Call Stack window

Setting Breakpoints and Stepping Through Code

You can specify breakpoints in your script. A breakpoint is a point at which you want your script to temporarily stop running. You can then use stepping options to execute your code on a line-by-line basis. By implementing breakpoints in this way, you can examine the effects of each line of code to locate and resolve logic errors. You can set breakpoints on specific lines to pinpoint problems in your scripts.

To set a breakpoint on a specific line, take the following actions:

- In the Microsoft Script Debugger, place the insertion point in the line where you want the breakpoint.
- On the **Debug** menu, click **Toggle Breakpoint**.

The line where you set the breakpoint is displayed in red to indicate that it is a breakpoint.

You can then repeat the procedure for each breakpoint that you want to set in the document. After you set your breakpoints, you can continue running the script.

The debugger stops at the first breakpoint that it encounters. You can then use the stepping options to execute the lines according to your requirements.

The script debugger toolbar has the following line-stepping options:

- **Step Into**

This option enables you to step through your code one line at a time. When a call to a **Sub** or **Function** procedure is encountered, this option indicates that you want to step through that procedure line by line.

This is useful if you have not already debugged the procedure.

- **Step Over**

This option also enables you to step through your code one line at a time. However, when a call to a **Sub** or **Function** procedure is encountered, this option indicates that you do not want to step through that procedure line by line. Instead, that procedure is run at full speed, and you can continue to step through subsequent lines of code that follow the procedure call.

This is useful if you have already debugged the procedure.

- **Step Out**

If you have stepped into a lengthy procedure, but at some point you want to run the rest of the procedure at full speed, you can use this option. You can continue to step through subsequent lines of code that follow the procedure call.

This is useful if you have found the error in the procedure, or if you stepped into the procedure when you wanted to step over it instead.

Viewing and Changing Variable Values

Another valuable debugging technique is viewing and changing the values of variables or properties as you step through your code. Viewing the values that are stored in variables or properties can help you to determine whether the script is running properly.

You can affect the way in which the script runs by making changes directly to values in the variables that are contained in a running script. After changing these values, you can continue to run the script and see the effects of your changes.

To view and change values, you use the Command window. You can evaluate any expression in the window, enter script commands, and see their effects. Changes that you make in the window directly affect the script that is currently running because the Command window is active.

You can use the Command window to run script commands when you are at a breakpoint or have stepped from a breakpoint to other statements.

Viewing the Call Stack

There are advantages to creating reusable procedures in your script code. However, breaking up scripts into procedures can complicate debugging. When you use the debugger to step through your script, you may encounter code whose execution threads from procedure to procedure. The call stack enables you to track the order of procedures calls.

The Microsoft Script Debugger has a special window for viewing the call stack. You can open the Call Stack window from the **View** menu. You can then use the list of procedure calls to trace the execution of the program and locate any procedures that cause errors.

Lab: Error Handling and Debugging



- Exercise 1
Trapping Run-Time Errors
- Exercise 2
Debugging VBScript

After completing this lab, you will be able to:

- Write run-time error-handling code.
- Use the Microsoft Script Debugger.
- Find and fix logic errors.

Estimated time to complete this lab: 30 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine first and then start the 2433B-VISTA-CL1-04 virtual machine.
- Log on to each virtual machine as **FOURTHCOFFEE\Administrator** with a password of **Pa\$\$w0rd**

Lab Scenario

You are an administrator for the FourthCoffee domain, which wants to create scripting tools to log employee test scores and work hours. You must make sure that the scripts that you want to use work properly before you put them in a production environment. For the first script, you have decided to write some error-handling code, and you have decided to debug the second script on a test server before posting it to a production server.

Exercise 1

Trapping Run-Time Errors

In this exercise, you will create and implement a run-time error trap by using Visual Basic, Scripting Edition.

The principal tasks for this exercise are as follows:

- To execute a script that can cause run-time errors.
- To write error-handling code.
- To test your error-handling code regarding division by zero.
- To test your error-handling code regarding type mismatch.
- To test the recoverability of your code.

Task	Supporting information
1. To execute a script that can cause run-time errors.	<ul style="list-style-type: none">• On the 2433B-VISTA-CL1-04 virtual machine, use Windows Explorer to run the script at E:\Labfiles\Starter\ExamAverage.vbs.• Attempt to divide by zero and observe the results of the script.• Also, enter non-numeric values into the user interface and observe the results of the script.
2. To write error-handling code.	<ul style="list-style-type: none">• Start PrimalScript.• Open the ExamAverage.vbs file using PrimalScript.• Review the TODO Handle Errors section of the script, and uncomment the lines of code.
3. To test your error-handling code regarding division by zero.	<ul style="list-style-type: none">• Run the script from Windows Explorer and attempt to perform division by zero. Observe the change in the script's behavior.
4. To test your error-handling code regarding type mismatch.	<ul style="list-style-type: none">• Run the script from Windows Explorer and attempt to enter a word in the script's user interface. Observe the change in the script's behavior.
5. To test the recoverability of your code.	<ul style="list-style-type: none">• Use the script to perform division of whole numbers and observe the script's behavior.

Questions

Q: What is the name of the object that holds the details about run-time errors?

Q: Name two commonly used properties of the **Err** object.

Exercise 2

Debugging VBScript

In this exercise, you will use the Microsoft Script Debugger to find and fix logic errors.

The scenario is that you are checking a script on a test server environment before you deploy the script in a production server.

The principal tasks for this exercise are:

- To port a script to the server.
- To test the script for run-time errors.
- To debug the script.
- To fix the logic errors in your code.
- To test your fixed code.

Task	Supporting information
1. To port a script to the server.	<ul style="list-style-type: none"> • Use PrimalScript to open the script at E:\Labfiles\Starter\Debug_Exercise.vbs. • Save the script to the domain controller at \\LON-DC1\Labshare\.
2. To test the script for run-time errors.	<ul style="list-style-type: none"> • Switch to the domain controller and log on as a domain administrator. • Use Windows Explorer to browse to the C:\Labshare folder and run the Debug_Exercise.vbs script. • Use five days for the number of days worked for the week. • Note that there are no run-time errors, but notice the number of days for which the script prompts you to enter hours. This is a logic error.
3. To debug the script.	<ul style="list-style-type: none"> • Open a command prompt and change directories to the C:\Labshare folder. • Start the Microsoft Script Debugger from the command prompt with the following command. <pre style="background-color: #e0e0ff; padding: 5px;"><code>cscript debug_exercise.vbs //x</code></pre> <ul style="list-style-type: none"> • Use the Microsoft Script Debugger command window and Step Into command to attempt to find the logic error in the script.
4. To fix the logic errors in your code.	<ul style="list-style-type: none"> • Use the Stop Debugging command and close the Script Debugger. • Open the script file in Notepad to fix the logic errors that you found. • Save the script file.
5. To test your fixed code.	<ul style="list-style-type: none"> • Switch to the command prompt and run the script using the following command. <pre style="background-color: #e0e0ff; padding: 5px;"><code>cscript debug_exercise.vbs</code></pre> <ul style="list-style-type: none"> • Follow the instructions from the user interface and notice the number of days for which the script prompts you to input your hours.

Questions

Q: What are the three stepping options that the Microsoft Script Debugger supports?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

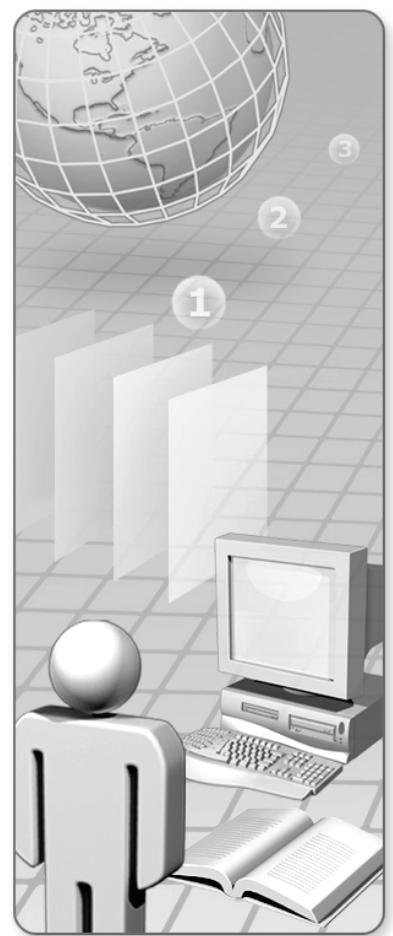
After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-04 virtual machines and discard any changes.

Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

Module 5: ADSI

Table of Contents

Module Overview	5-1
Lesson 1: Overview of ADSI	5-2
Lesson 2: Binding with ADSI	5-10
Lesson 3: ADSI Objects	5-19
Lesson 4: Searching Active Directory	5-27
Lab A: ADO Search	5-33
Lesson 5: Creating New ADSI Objects	5-38
Lesson 6: Managing Security, Shares, and Services by Using ADSI	5-44
Lab B: Scripting Administrative Tasks by Using ADSI	5-53



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Overview of ADSI**
- **Binding with ADSI**
- **ADSI Objects**
- **Searching Active Directory**
- **Creating New ADSI Objects**
- **Managing Security, Shares and Services by Using ADSI**

It is important that you, as a network administrator, can access the functionality of Microsoft® Active Directory® directory service through your administrative scripts. The two most common methods are by using Active Directory Services Interfaces (ADSI), and by using Windows Management Instrumentation (WMI). For more information about WMI, see Module 8, “WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

This module describes ADSI and shows how ADSI enables you to automate many administrative tasks by using scripts.

Objectives

After completing this module, you will be able to:

- Describe how ADSI works.
- Use scripts to bind to ADSI objects.
- Use ADSI object methods and properties in scripts.
- Search Active Directory.
- Create and modify objects in Active Directory.
- Manage security settings, shares, and services using ADSI.

Lesson 1: Overview of ADSI

- **What Is ADSI?**
- **Benefits of ADSI**
- **ADSI Architecture**

Before you write code that interacts with ADSI, you must understand how ADSI works and how it can help you to create administrative scripts.

This lesson will teach you about the uses of ADSI and about the benefits of using ADSI in your administrative scripts. In addition, the lesson will teach you about the ADSI architecture.

Objectives

After completing this lesson, you will be able to:

- Describe ADSI and ADSI versions.
- Describe the benefits of ADSI.
- Describe ADSI architecture.

What Is ADSI?

- **ADSI**
 - Scriptable interface to Active Directory
- **ADSI Versions**
 - Version 6.0 released with Windows Vista
 - Version 5.2 released with Windows Server 2003
- **Provides Access to Directory Service Objects**
- **Support for Multiple Directory Service Providers**
 - LDAP
 - Windows NT
 - NetWare providers

ADSI (previously called OLE Directory Services) is an abstraction layer that exposes the objects of various directory services through a single set of application programming interfaces (APIs).

ADSI Versions

Windows Vista® operating system ships with ADSI version 6.0. Windows Server® 2003 operating system ships with ADSI version 5.2. Microsoft Windows® 2000, Professional and Server, shipped with ADSI version 2.5. You cannot upgrade the version of ADSI without upgrading the operating system.

Provides Access to Directory Service Objects

You can use ADSI to manage the resources in a directory service, regardless of which network environment contains the objects. By using ADSI and scripting, administrators can automate common setup and administrative tasks that occur in a directory service.

Note: You can use ADSI with Windows PowerShell™ command-line interface. For more information about Windows PowerShell, see “Scripting with Windows Powershell” on the Microsoft TechNet Web site.

Support for Multiple Directory Service Providers

Scripts that are written in ADSI work with any directory service that offers an ADSI provider.

ADSI provides access to any Lightweight Directory Access Protocol (LDAP) service such as Microsoft Windows NT® version 4.0, and the NetWare Directory Services (NDS) and NetWare 3 Bindery (NWCOMPAT)-based servers from Novell. These

providers enable communication between the directory service and the client computer as follows:

- LDAP

The LDAP provider works with any LDAP version 2 or version 3 directory such as Microsoft Exchange Server 2007, 2003 and 5.5. This provider also works for Windows Server 2003 and Windows 2000 Active Directory. The LDAP provider uses dynamic-link libraries (DLLs) called adsldp.dll and wldap32.dll.

You can divide an LDAP address into three components:

- The first component is the Common Name (CN). This represents the leaf object, which is an object that contains no other objects. Examples of CNs are users or printers.
- The second component is the Organizational Unit (OU). This is the container object in the directory.
- The final component is the Domain Component (DC).

The following is an example of an LDAP address.

LDAP://CN=Alan,OU=Test,DC=Microsoft,DC=COM

This LDAP address refers to the object called Alan in the OU called Test in the domain called microsoft.com.

Note: Default containers such as Builtin, Computers, and Users are not OUs (even though they contain other objects). They must be referred to using CN naming, for example, CN=Users,DC=fourthcoffee,DC=com. However, the Domain Controllers container is an OU, and must be referred to using OU naming, for example, OU=Domain Controllers,DC=fourthcoffee,DC=com.

- Windows NT

This is the easiest provider to use because of its simple address format. It provides access to computers running Windows Vista, Windows Server 2003, Microsoft Windows 2000, and Windows NT version 4.0. The Windows NT provider provides access to Windows NT domains, and also provides some access to Windows Server 2003 and Windows 2000 Active Directory domains. The local provider DLL is adsnt.dll.

The Windows NT provider is designed to work with Windows NT domains. When used with Active Directory domains, this provider only works for operations across the whole domain, and where Active Directory exposes NT domain compatibility. Therefore, you can use the Windows NT provider to access user accounts in an Active Directory domain, but not to access OUs. The Windows NT provider also requires the use of NetBIOS computer names, and Domain Name System (DNS) naming cannot be used.

- Novell NetWare providers

Windows Server 2003, Windows 2000, and Windows NT 4.0 include two NetWare providers. The Novell NDS provider works with Novell Netware version 4.x or greater to give access to objects in the Novell Directory Service. The provider DLL is adsnds.dll.

The NWCOMPAT provider enables ADSI commands to access objects stored in Bindery-based servers such as Netware versions 3.11 and 3.12. The provider DLL is adsnw.dll.

Note: Windows Vista does not include the Novell NetWare providers.

Benefits of ADSI

- **Ease of Use**
- **Namespace Portability**
- **Powerful Functionality**

ADSI provides several major advantages to script writers.

Ease of Use

ADSI exports a comprehensive but straightforward set of interfaces that you can use to easily access objects in a directory service.

Namespace Portability

ADSI abstracts the complexity of the directory service interfaces, enabling you to use the same set of methods to access multiple directories. This simplifies script writing because you can reuse the basic methods with various directory objects in various namespaces.

Powerful Functionality

ADSI provides a set of powerful features that enable you to develop sophisticated script solutions. These solutions range from simple tasks such as finding objects in an OU through to complex scripts that create entire enterprise directory services and populate them with many objects.

Note: ADSI scripting is useful when you are testing the scalability and performance of Active Directory because it enables you to rapidly create extensive directory structures.

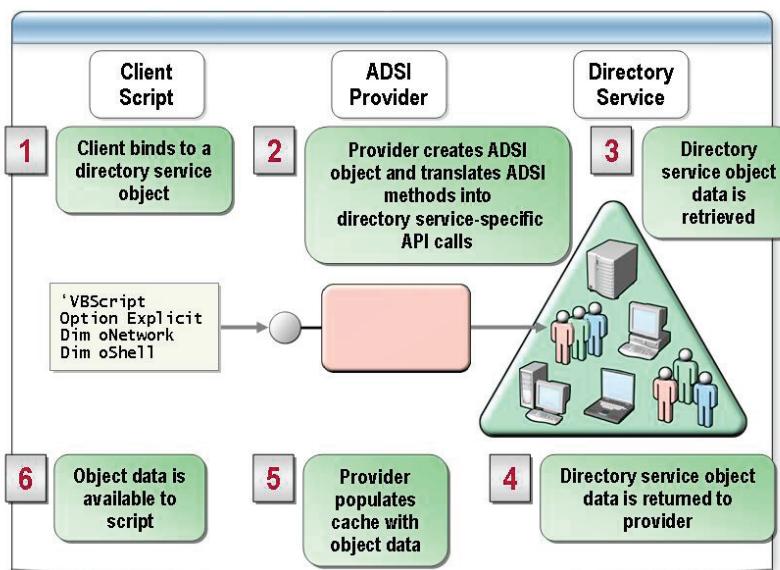
The following are examples of ADSI tasks:

- Read/write properties (attributes) to and from a directory service object:
 - Change the department name for all users in an OU.
 - Update a group's membership.

MCT USE ONLY. STUDENT USE PROHIBITED

- Create new user accounts.
- Manage the schema:
 - Create a report that generates a complete schema list.
 - Add objects to the schema.
- Manage security on directory service objects:
 - Set permissions for all user objects in a domain.
 - Change the permissions on a printer object in an OU.
 - Assign new permissions to a group in a domain.

ADSI Architecture



Before examining how to script ADSI, it is important to understand how the ADSI architecture works.

Binding

The ADSI client program must first find and connect (bind) to an object. The client program can be an application such as a script, a custom application, or one of the support tools supplied on the Windows Server 2003 CD-ROM.

The Provider

After the client program binds to an object, the ADSI provider DLL on the local computer processes the bind request from the client program and translates it into the directory-specific API call to retrieve the required object. The client script or application can then use the methods and properties of the object.

The following steps are involved in this process:

1. The client script or application makes a request to bind to an object in a directory service.
2. The ADSI provider on the client computer receives the request and creates an ADSI object in the client computer's address space. The provider then connects to the required directory service by using the API calls of that particular directory service.
3. The directory service object information, such as the object methods and properties, is retrieved from the directory.
4. The directory service object information is returned to the ADSI provider.

MCT USE ONLY. STUDENT USE PROHIBITED

5. The ADSI provider creates an instance of the ADSI object in the memory space of the client application.
6. The object methods are now available for the client script to use.

Lesson 2: Binding with ADSI

- **Binding**
- **Serverless Binding**
- **Binding with Alternative Credentials**

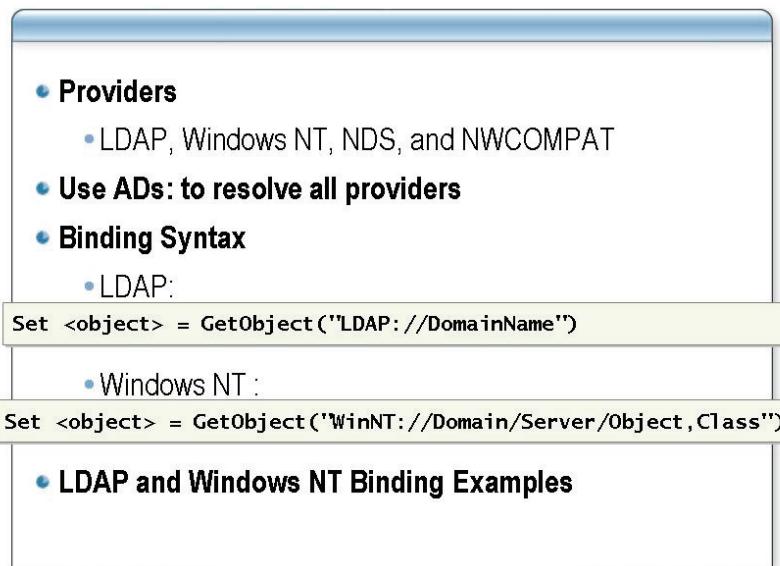
A process called binding must occur before you can use ADSI methods and properties. This lesson will teach you how to bind to directory service objects. The lesson will also describe serverless binding, and how to pass security credentials to ADSI when you bind to ADSI objects.

Objectives

After completing this lesson, you will be able to:

- Use binding with ADSI providers.
- Use serverless binding techniques.
- Use alternative credentials with ADSI binding.

Binding



The binding process creates an object in the client computer's memory address space. After your script has bound to an object, the script can invoke any method relevant to that type of object and can read or modify its properties.

Providers

An important part of the binding request from your script is to specify the provider that you want to bind to the object. As discussed earlier, ADSI includes four different providers:

- LDAP
- Windows NT
- NDS
- NWCOMPAT

Important: These names are all case-sensitive.

ADs:

There is also an **ADs:** name that you can use to resolve all of the available providers on the local computer. For example, if an enterprise uses a new provider called NewProv, you could insert the following code in any scripts that require this non-standard provider.

```
Set objProviders = GetObject("ADs:")
For Each provider In objProviders
    If provider.name = "NewProv:" Then
        WScript.Echo "The required provider is installed"
        WScript.Quit
    End If
```

```
Next  
WScript.Echo "The required provider is missing"  
WScript.Quit
```

The following script lists the providers that are currently installed.

```
Set objProviders = GetObject("ADs:")  
WScript.Echo "The providers installed are: "  
For Each provider In objProviders  
    WScript.Echo provider.name  
Next  
WScript.Quit
```

Binding Syntax

The following example shows the ADSI binding syntax for the Windows NT provider.

```
Set <Object> = _  
    GetObject ("WinNT://Domain/Server/Object,Class")
```

where:

- **<Object>** is the object variable that will contain the reference to the retrieved Active Directory object.
- **GetObject** is the method used to bind to the ADSI namespace and create the local copy of the object that will be used in the script.
- **WinNT** is the ProgID for the required ADSI provider. This name is sometimes referred to as the moniker. (Moniker is a Component Object Model (COM) term that describes the mechanism that is used to bind to COM objects, in this case the ADSI providers.)
- Domain is the domain name.
- Server is the computer name.
- **Object** is the object name.
- **Class** is the class specification. This is optional, but it is useful for preventing ambiguity where various objects have the same names. Using this class specification can also accelerate the binding process.

Together, the provider name and path are called the ADsPath or bind string.

LDAP and Windows NT Binding Examples

The following examples show a number of ways that you can bind to LDAP and Windows NT objects:

- Binding to the root of the LDAP namespace.

```
Set oMyObj = GetObject("LDAP:")
```

- Binding to the root of the Windows NT namespace.

```
Set oMyObj = GetObject("WinNT:")
```

- Binding to the root of a specific Windows Server 2003 domain.

```
Set oMyObj = GetObject("LDAP://MyDomain.com")
```

- Binding to the root of a specific Windows NT domain.

```
Set oMyObj = GetObject("WinNT://MyDomain")
```

- Binding to the root of a specific Windows Server 2003 domain, by providing the class parameter.

```
Set oMyObj = GetObject("LDAP://DC=MyDomain, DC=com")
```

- Binding to the root of a specific Windows Server 2003 domain, by providing a computer name instead of the domain name.

```
Set oMyObj = GetObject("LDAP://MyServer")
```

Note: The preceding example binds to the domain that includes the server. This differs from the Windows NT provider, which binds to the computer object if a computer name is used.

- Binding directly to a specific computer by using LDAP.

```
Set oMyObj = GetObject("LDAP://CN=MyServer,OU=Domain  
Controllers,DC=MyDomain,DC=com")
```

Note: If the computer to which you bind is not a domain controller, the provider binds to the Security Account Manager (SAM) of the local computer.

- Binding directly to a specific computer by using Windows NT.

```
Set oMyObj = GetObject("WinNT://MyServer,computer")
```

- Binding to a user object on a server by using LDAP.

```
Set oMyObj = GetObject  
("LDAP://MyServer.MyDom.com/CN=Alan,OU=CM,DC=MyDom,DC=com")
```

- Binding to a user object on a server by using Windows NT.

```
Set oMyObj = GetObject("WinNT://MyDom/MyServer/Alan,user")
```

Serverless Binding

- **Preferred Binding Method**
 - No hard-coded server name
- **RootDSE Binding**
 - No hard-coded domain name
 - Portable, site-aware, fault-tolerant
- **Global Catalog Server Binding**
 - GC://

The previous topic taught you how to bind to specific computers. This approach may not always be possible or desirable for several reasons.

Preferred Binding Method

The preferred method of binding is called serverless binding. By using this approach, no server name is specified or hard-coded in the script. The following example shows an example of a serverless binding with a specified domain name, but no server name. This method relies on the Windows Server 2003 location service, which uses DNS to bind to the best server for the requested domain.

```
Set oMyObj = GetObject("LDAP://DC=MyDomain, DC=com")
```

Serverless binding is a valuable mechanism to include in your scripts, because it provides the following benefits:

- Site awareness

DNS uses site information on Windows Server 2003 to determine whether a domain controller exists in the same physical site as the client program that runs the script. If no domain controller exists in the site, DNS finds any available domain controller for the script to use, and returns that domain controller for binding to the provider.

- Fault tolerance

Serverless binding binds to a namespace rather than to a specific domain controller. Therefore, if any domain controller is temporarily unavailable, the locator service attempts to find another one in the same domain.

RootDSE Binding

The serverless binding approach works well in a single-domain environment. However, if your script will run from multiple domains, you must use an object called **RootDSE**. This eliminates the requirement for a hard-coded reference to the domain in addition to the server. You can use the **RootDSE** object to resolve the current domain name before binding to it. Each domain controller in Windows Server 2003 has a unique **RootDSE** object that you can use to provide information about the various namespace objects in the domain. This information can then be used in scripts.

Note: DSE stands for DSA-Specific Entry, where DSA is an X.500 term for the directory server.

By retrieving the **DefaultNamingContext** property from the **RootDSE** object, you can bind to the current domain. The following is an example of serverless binding, using the **RootDSE** object.

```
Set oRootDSE = GetObject("LDAP://RootDSE")
Set oMyDomain = GetObject( "LDAP://" & oRootDSE.Get("defaultNamingContext"))
WScript.echo oMyDomain.name
```

This example displays the name of the domain in which the script is running. First, the script binds to the **RootDSE** object. Then, it creates a new object called **oMyDomain** that binds to the current domain by using the **DefaultNamingContext** property of the **RootDSE** object. Finally, **WScript.echo** displays the domain name.

The **RootDSE** object enables the script to run from any domain and to successfully receive a binding for the script to use. The following is a list of some of the most commonly used properties of the **RootDSE** object on a Windows Server 2003 domain controller.

Property	Description
namingContexts	Multi-valued. The distinguished names for all the naming contexts that are stored on this directory server. By default, a Windows 2000 domain controller contains at least three namespaces: Schema , Configuration , and one for the domain that includes the server.
defaultNamingContext	The distinguished name for the domain that includes this directory server.
schemaNamingContext	The distinguished name for the schema container.
configurationNamingContext	The distinguished name for the configuration container.
rootDomainNamingContext	The distinguished name for the first domain in the forest that contains the domain that includes this directory server.
supportedLDAPVersion	Multi-valued. The LDAP versions (specified by major version number) supported by this directory server.

Property	Description
highestCommittedUSN	The highest Update Sequence Number (USN) used on this directory server. Used by directory replication.
dnsHostName	The DNS address for this directory server.
serverName	The distinguished name for the server object for this directory server in the configuration container.
currentTime	The current time set on this directory server.
dsServiceName	The distinguished name of the NTDS settings object for this directory server.

Global Catalog Server Binding

If your scripts must search for objects in multiple domains in a forest, you can bind to a Global Catalog (GC) server in the domain in which the script is running. GC servers contain a list of all the objects in a domain. Each object in a GC includes a complete subset of attributes. Storing the required attribute in this subset makes querying the local GC considerably faster than relying on a complete cross-domain search.

To bind to a GC server, use its name, as the following example shows.

```
Set oGC = GetObject("GC://DC=MyDomain, DC=com")
```

Any searches performed on this GC server return all objects in the forest that match the criteria, including objects from the client computer's local domain.

Binding with Alternative Credentials

- **Binding As a Different User**
 - Use **OpenDSObject** method

```
Set oMyDS = GetObject("LDAP:")  
Set oMyObj = oMyDS.OpenDSObject _  
(AdsPath, UserName, Password, ADS_SECURE_AUTHENTICATION)
```

- **Using ADSI Constants**

In all of the examples shown so far, ADSI binds to the directory by using the logon credentials of the user who is running the script. However, there are situations where your scripts must bind to a particular directory service by using different security credentials from those of the user who is logged on.

Binding As a Different User

ADSI provides an **OpenDSObject** method that allows binding to occur with different security credentials from those of the user who is running the script.

OpenDSObject takes as arguments the **ADsPath** of the object or subtree to bind, the username, the password, and the authentication method. To use this method, you must bind directly to "LDAP;"" as the following example shows.

```
Set oMyDS = GetObject("LDAP:")  
Set oMyObj = oMyDS.OpenDSObject _  
(ADsPath, UserName, Password, ADS_SECURE_AUTHENTICATION)
```

Note: This method requires that the user's name and password be entered in the script as clear text. This presents a security issue that you must consider when planning.

Using ADSI Constants

The last argument in the preceding example specifies the authentication flags to govern how the authentication takes place for the **OpenDSObject** method. You achieve this by using constants that are not intrinsically available to Microsoft Visual Basic®, Scripting

Edition (VBScript). You must define these constants at the top of the script. The following table lists the possible values for the authentication parameter.

Constant	Description	Value
ADS_SECURE_AUTHENTICATION	Requests secure authentication. When set, the Windows NT provider uses NT LanManager (NTLM) to authenticate the client program. Active Directory uses the Kerberos protocol, and possibly NTLM.	1
ADS_USE_ENCRYPTION	Forces ADSI to use encryption for data exchange over the network.	2
ADS_USE_SSL	Encrypts the channel with Secure Sockets Layer (SSL) technology. Certificate Server must be installed to support SSL encryption.	2
ADS_READONLY_SERVER	For a Windows NT provider, ADSI tries to connect to a Primary Domain Controller or Backup Domain Controller. For Active Directory, this flag indicates that a server is not required for a serverless binding.	4
ADS_NO_AUTHENTICATION	Requests no authentication. Setting this flag is the same as requesting an anonymous binding. The Windows NT provider does not support this flag.	16
ADS_FAST_BIND	When you set this flag, ADSI will not attempt to query the ObjectClass property. Therefore, only the common properties and methods supported by all ADSI objects are available, not the object-specific properties. This improves the performance of the binding process if this information is all that is required by the script.	32
ADS_USE_SIGNING	Verifies data integrity to ensure that the data received is the same as the data sent. The ADS_SECURE_AUTHENTICATION flag must also be set to use the signing.	64
ADS_USE_SEALING	Encrypts data by using Kerberos. The ADS_SECURE_AUTHENTICATION flag must also be set in order to use the sealing.	128

Lesson 3: ADSI Objects

- Object Methods
- Object Properties
- Demonstration: Examining Active Directory Objects by Using ADSI Edit

The previous lesson showed you how to manipulate various directory services by binding to a directory and then manipulating the objects contained in the directory.

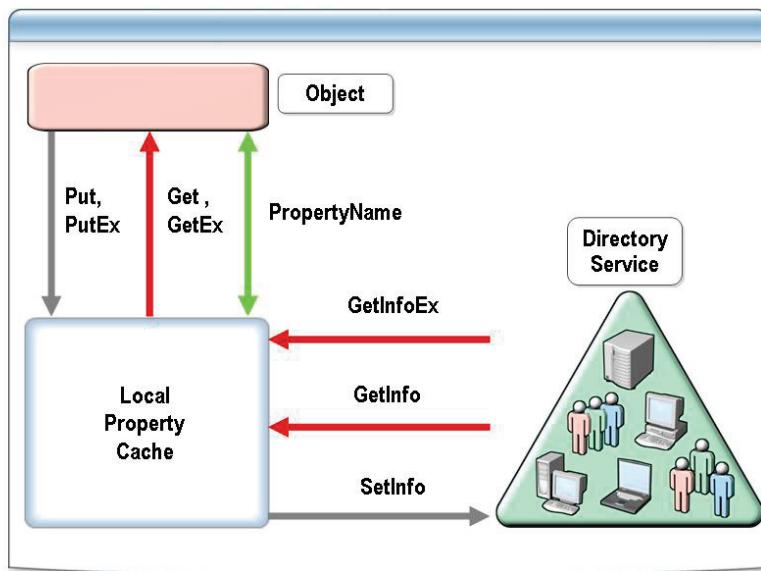
This lesson will teach you more about the objects available to your scripts when you write code that interacts with ADSI. You will see how to invoke object methods and how to set and retrieve the values of object properties.

Objectives

After completing this lesson, you will be able to:

- Use ADSI object methods in scripts.
- Access ADSI object properties in scripts.

Object Methods



After your script has completed the binding process, the instance of the object becomes available for the script to use. The methods that the script can use provide access to both the instance of the object in memory and the object in the directory. They do this by means of a local property cache that is used to store the values being used. Understanding the relationship between these three components is essential to understanding how to develop ADSI scripts.

Using the Local Property Cache

To avoid making a call to a domain controller every time information about an object is read or modified, ADSI caches object properties locally on the client computer. When you first bind to an object with **GetObject**, the properties of that object are emptied into the local cache. When you explicitly instruct ADSI to populate the cache, or when you first try to retrieve an absent property value, ADSI automatically retrieves all property values that have been set for that object, and places them in the cache.

Further handling of the object properties retrieves the values from the local cache, thus requiring no additional calls across the network to the domain controller.

The following table lists the seven ADSI object methods that work with ADSI objects and the local property cache.

Method	Description
Get	Retrieves the value of a property. variable = oMyObj.Get("PropName")
GetEx	Retrieves the values of a property that has multiple values. Array() = oMyObj.GetEx("PropName")
Put	Sets the value of a property.

Method	Description
	oMyObj.Put "PropName", "Value"
PutEx	Sets the values of a property that has multiple values. oMyObj.PutEx ADSProp, "PropName", Array("Value1", "Value2")
GetInfo	Retrieves the values of <i>all</i> of the object's properties from the directory service and puts them in the local property cache. oMyObj.GetInfo
GetInfoEx	Retrieves the values of <i>some</i> of the object's properties from the directory service and puts them in the local property cache. You must specify which properties to retrieve. oMyObj.GetInfoEx Array("PropName"), 0
SetInfo	Saves changes to an object's properties back to the directory service. oMyObj.SetInfo

The following ADSI script example demonstrates the methods that you can use to read the values of properties that belong to an object in Active Directory.

```
'Create an instance of an OU object from existing AD OU
Set oOU = GetObject("LDAP://cn=Admin10,OU=WST,DC=Microsoft,DC=local")
'Get a value from AD into the Local Property cache (LPC)
WScript.Echo oOU.PropertyCount ' 0 Properties cached
oOU.GetInfoEx Array("memberOf"), 0
WScript.Echo oOU.PropertyCount ' 1 Property cached
'Fill the Local Property Cache
oOU.GetInfo
WScript.Echo oOU.PropertyCount ' all Properties cached
'Get a value from the LCP
WScript.Echo oOU.Get("cn")
'Same as above using different method
WScript.Echo oOU.cn
'Get & echo a multivalued property from Local Property Cache
aMemberOf = oOU.GetEx("memberOf")
For Each grp In aMemberOf
    WScript.Echo grp
Next
```

This script starts by creating an instance of an existing user object from an LDAP directory service. The property count of the object's local property cache is then echoed to the screen. At this stage, the property count is zero because no request to populate the local property cache has been made.

Next, the **GetInfoEx** method populates the local property cache with a single property, **memberOf**. This population is checked by echoing the property count again. The **GetInfo** method then pulls from the directory all properties that contain a set value. The property count shows the number of these values now in the local property cache. The next two lines echo the value of the **CN** property stored by using two different techniques, the **Get** method and the **PropertyCount** property.

The last method that is demonstrated is **GetEx**, which is used to access the values of properties that have multiple values. This method returns the values of a property as an array that can then be stepped through in any number of ways. The script demonstrates the use of a **For...Each** loop.

Note: The final part of the script returns an error if the user is not a member of any group or is only a member of one group. You could insert additional code to check for an empty **memberOf** property.

Writing Information into the Directory

The following script demonstrates the methods that you can use to create, update, or append property values for objects in the local property cache. The script then updates the properties back to the directory.

```
'Setup ADSI Constant
Const ADS_PROPERTY_APPEND = 3
'Create an instance of a user object from AD
Set oUser = GetObject("LDAP://cn=Admin10,OU=WST,DC=Microsoft,DC=local")
'Update the local property cache value using the Put method
oUser.Put "Description", "Cache Demo User"
'Same as above using the dot method
oUser.Description = "Cache Demo User again"
'Update LPC with a multivalue property
oUser.PutEx ADS_PROPERTY_APPEND, "otherTelephone", _
Array("555-0100", "555-0199")
'Write the local property cache back to AD
oUser.SetInfo
```

The first task of this script is to define the ADSI constants that the script will use. These constants will be used by the **PutEx** command, which will be discussed later.

After the instance of the Admin10 user has been created, the script writes a new value for the **Description** property, by using the **Put** method and then the **.PropertyName** (or “dot”) syntax. Notice that you can use the **.PropertyName** technique both to read and write values in ADSI scripting. You can then use the **PutEx** method to append an extra value into an existing property that has multiple values. The following example shows the syntax for this method.

```
object.PutEx ADSProperty, "property", Array("Val1", "Val2")
```

In this syntax, **ADSProperty** is a constant value that is used to indicate how the value must be updated. **ADSProperty** can consist of any of the values in the following table.

Constant name	Value
ADS_PROPERTY_CLEAR	1
ADS_PROPERTY_UPDATE	2
ADS_PROPERTY_APPEND	3
ADS_PROPERTY_DELETE	4

In the preceding example, **property** is the schema name of the property to be set. **Val1** and **Val2** are the values that will be used to modify the property. These values are passed to the method as elements of an array.

The last line of the example before that uses the **SetInfo** method to write the updated local property cache back to the directory service.

Note: Failing to call **SetInfo** is a common mistake in ADSI scripting that results in all updates to the cached object being discarded. Note that, if you make a call to **GetInfo** (either explicitly or implicitly through a **Get** method) before **SetInfo** has updated the directory service object, all property values in the local cache will be overwritten by the fresh call to the directory service object.

Object Properties

- **Six Standard Properties**
 - Name
 - Class
 - GUID
 - ADsPath
 - Parent
 - Schema
- **Additional Properties Are Specific to the Class Type**
- **Reading Multiple Properties**
- **Script Uses Schema Names, Not Display Names**

The next major step in using ADSI is to directly access the values stored in the properties for the directory objects. To do this, you must first know the names of the properties to use.

Six Standard Properties

Most ADSI objects expose six standard properties, which the following table describes.

Property	Description
Name	The name of the object.
Class	The schema class name of the object.
GUID	A globally unique identifier structure that uniquely identifies the object.
ADsPath	The string form of the object's path in the directory service. This path uniquely identifies the object.
Parent	The ADsPath name of the object's parent container.
Schema	The ADsPath name of this object's schema class object.

The following example displays these standard properties for an OU.

```
Set oMyObj = GetObject("LDAP://ou=WST,dc=content,dc=com")
WScript.Echo "Name is      " & oMyObj.Name
WScript.Echo "Class is     " & oMyObj.Class
WScript.Echo "GUID is      " & oMyObj.GUID
WScript.Echo "ADsPath is   " & oMyObj.ADPath
WScript.Echo "Parent is    " & oMyObj.Parent
WScript.Echo "Schema is    " & oMyObj.Schema
```

Additional Properties Are Specific to the Class Type

The standard ADSI properties are useful for identifying an object and its location in the directory hierarchy. However, most of the useful information about an object is stored in other properties that are specific to the class of that object.

For example, user objects support a **displayName** property that you can use, as the following example shows.

```
Set oMyUser = GetObject("LDAP://cn=Admin10,ou=WST,dc=fourthcoffee,dc=com")
WScript.Echo "The full name is = " & oMyUser.Get("displayName")
```

Note: There are two ways to return object properties. You can use dot notation with all properties, for example, **oMyUser.displayName**. For some properties, you can also use the following type of notation, for example, **oMyUser.Get("displayName")**. You must use dot notation for all standard ADSI properties except for "name."

Reading Multiple Properties

You may require a list of all properties that have had their values set for a specific object. To achieve this, you can write code that counts all of the properties in the local cache by using the **PropertyCount** and **Item** methods, as the following example shows.

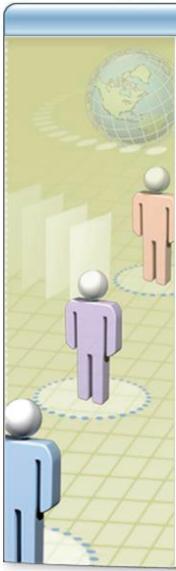
```
Option Explicit
Dim objGroup, i, sPropList, count
On Error Resume Next
Set objGroup = GetObject("LDAP://cn=Admin10,ou=WST,dc=fourthcoffee,dc=com")
objGroup.GetInfo
i = objGroup.PropertyCount
sPropList = "There are " & i _
& " values in the local property cache:" & vbCrLf & vbCrLf
For count = 0 To (i-1)
    sPropList = sPropList & objGroup.Item(CInt(count)).Name & vbCrLf
Next
WScript.Echo sPropList
```

This is a useful script, because it shows the AD names that can be used in other scripts.

Script Uses Schema Names, Not Display Names

If you run the preceding example code, you find that the property names returned are not the same as those shown through the Microsoft Management Console (MMC) snap-in Active Directory Users and Computers. The property names used in the script example are the schema names, not the display names. The user interface tools for Windows Server 2003 use the display names, so it can be useful to browse an object in Active Directory and display the schema names that the script will use. The demonstration at the end of this section uses a tool (ADSI Edit) that displays schema names.

Demonstration: Examining Active Directory Objects by Using ADSI Edit



In this demonstration you will see how:

- To use ADSI Edit to examine Active Directory objects

In this demonstration, you will see how to use ADSI Edit, one of the support tools for Windows Server 2003, to examine Active Directory objects.

Key Points

The key points of this demonstration are:

- To see how you can use ADSI Edit to see the complete Active Directory path for an object.
- To see how you can use ADSI Edit to determine the attributes of an object that are not empty.

Questions

After viewing the demonstration, answer the following questions. Be prepared to discuss your answers with the class.

Q Why is ADSI Edit a useful tool for administrative script writers?

Q Why must you take care when using ADSI Edit?

Lesson 4: Searching Active Directory

- **Searching Active Directory by Using ADO**
- **One Connection For Multiple Searches**
- **Separate Connections For Multiple Searches**

You will often need to search for specific objects located in Active Directory. This lesson describes how you can use Microsoft ActiveX® Data Objects (ADO) to perform efficient and powerful search routines. This lesson also describes how to use ADO connections and other ADO objects to perform Active Directory searches.

Objectives

After completing this lesson, you will be able to:

- Use scripts to search Active Directory.
- Use a single Active Directory connection for multiple searches.
- Use separate Active Directory connections for multiple searches.

Searching Active Directory by Using ADO

- **Using ADO for Active Directory Searches**
 - Connections
 - Commands
 - Recordsets
- **Building an LDAP Query String**

The ADO object model is one the most common approaches used in script to search for objects in Active Directory. ADO can perform queries against the directory based on the Object Linking and Embedding (OLE) database support that is built into ADSI.

Using ADO for Active Directory Searches

ADO is a simple and efficient way to perform Active Directory searches. When performing a search, you will often use three ADO objects: **Connection**, **Command**, and **Recordset**.

- You use the **Connection** object to specify the provider name, alternate credentials (if applicable), and other flags.
- You use the **Command** object to specify search preferences and the query string.

Note: You must associate the **Connection** object with a **Command** object before you execute the query.

- You use the **Recordset** object to represent the result set.

Using these three objects to issue a query against Active Directory involves the following four steps:

1. Establish the connection to the data source by using the **Connection** object.
2. Execute a command by using the **Command** object.
3. Create a recordset by using the **Recordset** object.
4. Read the information in the recordset.

Note: Although it is possible to use Microsoft SQL Server™ syntax for these queries, this technique is outside the scope of this course. For more information about this technique, see the “Getting Data” section of the “ADO Programmer’s Guide” on the MSDN Web site.

Building an LDAP Query String

To successfully search Active Directory by using ADO, you must first understand how the LDAP query strings are constructed. The following example shows the basic syntax for an LDAP query string.

"<FQADsPath>;(Filter);Attributes;Scope"

In this example, FQADsPath is the fully qualified name of the starting point for your search. Filter is the list of elements to be included in or excluded from the search. You can qualify these elements by using the symbols shown in the following table.

Name	Symbol
Logical AND	&
Logical OR	
Logical NOT	!
Equal to	=
Approx equal to	~=
Greater than	>=
Less than	<=

The following example shows how to use a symbol to qualify an element.

```
LDAP://DC=fourthcoffee,DC=com>;(&(objectCategory=Person)
    (objectClass=user)):name;telephoneNumber;subTree
```

In this example, the logical AND symbol is used to ensure that the only value returned belongs to the object category **Person** and the object class **user**.

You can also use wildcard characters such as asterisks, as the following example shows.

(ObjectClass=*)

Note: The search string does not contain any spaces. If you include a space, the returned results will contain incorrectly formatted data.

The following list provides explanations for the different parts of the query string:

- Attributes are the required values that will be returned by the search. A comma separates each attribute. In the above example, these are the user's **name** and **telephoneNumber** properties.

- Scope refers to the boundary of the search. After you have declared the starting point, the scope determines where the search stops, based on the following choices:
 - Base
Search the object itself.
 - OneLevel
Extend the search to the immediate children of the object, but exclude the base object itself.
 - Subtree
Extend the search to multiple sublevels of the object, and include the base object.
If you do not specify the scope, the search will default to a Subtree search.

MCT USE ONLY. STUDENT USE PROHIBITED

One Connection for Multiple Searches

```

Set con = CreateObject("ADODB.Connection")
Set com = CreateObject("ADODB.Command")

con.Provider = "ADsDSOObject"
con.Open

Set Com.ActiveConnection = con

Com.CommandText = "<LDAP://DC=fourthcoffee,DC=com>;" _
& "(objectClass=user);name,telephoneNumber;subTree"

Set rs = Com.Execute

Do Until rs.EOF
    sResultText = sResultText & rs.Fields("Name") & " , " _
    & rs.Fields("telephoneNumber") & vbCrLf
    rs.MoveNext
Loop

WScript.Echo sResultText

```

The following script example echoes to the screen, telephone number, and name for all users in the fourthcoffee.com domain.

The approach shown here is most useful when you must perform several searches consecutively, because only one connection is required. After the connection has been opened, any number of recordsets can be generated. Notice that the third line of the code (excluding the comments) specifies which ADO provider this connection will use for all Active Directory searches. The ADsDSOOObject provider is used.

```

'Create connection and command object
Set con = CreateObject("ADODB.Connection")
Set com = CreateObject("ADODB.Command")
'Open the connection with the ADSI-OLEDB provider name
con.Provider = "ADsDSOObject"
con.Open
'Assign the command object for this connection
Com.ActiveConnection = con
'Create a search string
Com.CommandText = "<LDAP://DC=fourthcoffee,DC=com>;" _
& "(objectClass=user);name,telephoneNumber;subTree"
'Execute the query
Set rs = Com.Execute()
'Navigate the record set
Do Until rs.EOF
    'Build Result Text
    sResultText = sResultText & rs.Fields("Name") & " , " _
    & rs.Fields("telephoneNumber") & vbCrLf
    rs.MoveNext
Loop
WScript.Echo sResultText

```

Separate Connections for Multiple Searches

- Create the ADO Recordset object
- Open the recordset based on the arguments
- Build result text

```
Set rs = CreateObject("ADODB.Recordset")
rs.Open "<LDAP://DC=fourthcoffee,DC=com>;" _
& "(objectClass=user);name,telephoneNumber;subTree" _
, "provider = ADsDSOObject"
Do Until rs.EOF
    sResultText = sResultText & rs.Fields("Name") & " , " _
    & rs.Fields("telephoneNumber") & vbCRLF
    rs.MoveNext
Loop
WScript.Echo sResultText
```

You can script this example of an ADO search of Active Directory much more quickly than the previous example, because it simply creates an instance of an ADO database recordset and then uses it to conduct a search based on a single line of script. The syntax for this search is very similar to the previous example, except that a comma and provider statement are added after the search string, as the following example shows.

```
, "provider = ADsDSOObject"
```

This simple addition enables the query to function without the **Connection** or **Command** objects that were used in the previous example. The following is a complete example of how you can use this approach.

```
' Create the ADO Recordset Object
Set rs = CreateObject("ADODB.Recordset")
' Open the Record Set based on the arguments
rs.Open "<LDAP://DC=fourthcoffee,DC=com>;" _
& "(objectClass=user);name,telephoneNumber;subTree" _
, "provider=ADsDSOObject"
Do Until rs.EOF
' Build Result Text
    sResultText = sResultText & rs.Fields("Name") & " , " & _
    rs.Fields("telephoneNumber") & vbCrLF
    rs.MoveNext
Loop
WScript.Echo sResultText
```

The disadvantage of this example is that a new connection is created each time the query is executed. If you ran multiple queries in the same script, the previous example would be more efficient than this one.

Lab A: ADO Search



- **Exercise 1**
Performing Searches by Using ADSI and ADO
- **Exercise 2**
Searching for User and Computer Information by Using ADSI and ADO

After completing this lab, you will be able to:

- Search through the **2433Users** OU.
- Return a number of properties of the user objects by using the ADSI LDAP query syntax.

Estimated time to complete this lab: 15 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine.
- Log on to the 2433B-LON-DC1 virtual machine as **Administrator** using the password **Pa\$\$w0rd**
- Start the 2433B-VISTA-CL1-05 virtual machine.
- Log on to the 2433B-VISTA-CL1-05 virtual machine as **FOURTHCOFFEE\Administrator** using the password **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to use scripts to help to produce reports about Active Directory. You want to use the search functionality provided by using ADSI and ADO to perform Active Directory searches.

Exercise 1

Performing Searches by Using ADSI and ADO

In this exercise, you will write ADO code that performs searches by using an ADSI provider.

The principal tasks for this exercise are as follows:

- To create a new script by using a template.
- To investigate the ADO Search template.
- To edit the ADO Search template.

Tasks	Supporting information
1. To create a new script by using a template.	<ul style="list-style-type: none">• Start PrimalScript.• Open E:\Labfiles\Starter\ADOTemplate.vbs.• Edit the script information header so that it includes your name and today's date.• Save the file as E:\Labfiles\Starter\ADOSearch.vbs.
2. To investigate the ADO Search template.	<ul style="list-style-type: none">• Locate the variables that are used in the script. The first three variables are used to manipulate ADO objects and are used to instantiate and manipulate a Connection object, a Command object, and a Recordset object, respectively.• Locate the lines of code that are used to instantiate the Connection and Command objects.• Locate the lines of code that are used to manipulate the Connection and Command objects.• Locate the line of code that is used to create the Recordset object.• Locate the lines of code that are used to loop through the Recordset object and build a string from its contents.• Locate the line of code that is used to display the string to the user.
3. To edit the ADO Search template.	<ul style="list-style-type: none">• Edit the active connection to specify that you want to search the 2433Users OU in the fourthcoffee.com domain.• Edit the objectClass statement to specify a class type of user.• Edit the object-attribute statements to specify that you want to return the following attributes:<ul style="list-style-type: none">• sAMAccountname• givenName• sN• telephoneNumber• Edit the aResult.Fields statements to specify that you also want to display the above attributes.• Save the script.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Run the script by double-clicking E:\Labfiles\Starter\ADOSearch.vbs and review the details that are displayed in the message box.

MCT USE ONLY. STUDENT USE PROHIBITED

Questions

Q: What is the main directory service provider used for Windows Server 2003?

Q: What is the main reason for using the ADO object model in ADSI scripts?

Exercise 2

Searching for User and Computer Information by Using ADSI and ADO

In this exercise, you will modify the `ADOSearch.vbs` script, to include additional user attributes. You will also modify the `ADO Template.vbs` script to search for computer information.

The principal tasks for this exercise are:

- To search for additional user information.
- To search for computer information.

Task	Supporting information
1. To search for additional user information.	<ul style="list-style-type: none">• Edit the <code>ADOSearch.vbs</code> script, so that the object-attribute statements include some of the following user object attributes:<ul style="list-style-type: none">• <code>givenName</code>• <code>sn</code>• <code>initials</code>• <code>displayName</code>• <code>physicalDeliveryOfficeName</code>• <code>telephoneNumber</code>• <code>mail</code>• <code>wwwHomePage</code>• <code>userPrincipalName</code>• <code>sAMAccountname</code>• <code>streetAddress</code>• <code>l</code> (lowercase "L" as in Locale)• <code>st</code>• <code>postalCode</code>• <code>c, co, and countryCode</code>• <code>title</code>• <code>department</code>• <code>company</code>• Edit the aResult.Fields statements to specify that you also want to display the same attributes.• Save the file as <code>E:\Labfiles\Starter\ADOSearch-Users.vbs</code>.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Run the script by double-clicking E:\Labfiles\Starter\ADOSearch-Users.vbs and review the details that are displayed in the message box.
2. To search for computer information.	<ul style="list-style-type: none">• In PrimalScript, open <code>E:\Labfiles\Starter\ADOTemplate.vbs</code>.• Edit the script information header so that it includes your name and today's date.

Task	Supporting information
	<ul style="list-style-type: none">• Save the file as E:\Labfiles\Starter\ADOSearch-Computers.vbs.• Edit the active connection to specify that you want to search the whole fourthcoffee.com domain.• Edit the objectClass statement to specify a class type of computer.• Edit the object-attribute statements to specify that you want to return the following attributes:<ul style="list-style-type: none">• sAMAccountname• dNSHostName• operatingSystem• operatingSystemVersion• Edit the aResult.Fields statements to specify that you also want to display the above attributes.• Save the script.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Run the script by double-clicking E:\Labfiles\Starter\ADOSearch-Computers.vbs and review the details that are displayed in the message box.

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

After you complete the lab, do not shut down the virtual machines. You will use them again for the second lab in this module.

Lesson 5: Creating New ADSI Objects

- **Creating New OUs**
- **Creating New Users**
- **Creating New Groups**

In addition to searching for information about objects in Active Directory, you can also write code that creates new objects. This lesson will teach you how to create new OU objects, new users, and new groups.

Objectives

After completing this lesson, you will be able to:

- Use ADSI to create new OUs in Active Directory.
- Use ADSI to create new user accounts in Active Directory.
- Use ADSI to create new groups in Active Directory.

Creating New OUs

Creating an OU

```
Set oRootDSE = GetObject("LDAP://RootDSE")
Set oDom = GetObject _
    ("LDAP://" & oRootDSE.Get ("defaultNamingContext"))
Set oOU = oDom.Create("organizationalUnit", "OU=HQ")
oOU.Description = "Company Headquarters"
oOU.SetInfo
```

By using ADSI, creating a new OU is a simple task.

Creating an OU

The following example script shows the process of creating a new OU object in Active Directory.

```
Set oRootDSE = GetObject("LDAP://RootDSE")
Set oDom = GetObject("LDAP://" & oRootDSE.Get ("defaultNamingContext"))
Set oOU = oDom.Create("organizationalUnit", "OU=HQ")
oOU.Description = "Company Headquarters"
oOU.SetInfo
```

The first line in this example creates an object called **oRootDSE** that returns a connection to the root object of the directory service.

Using the **Get** method of the **oRootDSE** object creates a new object called **oDom**. This object references the default-naming context for the domain.

Using the **oDom** object and the **Create** method creates a new organizational object called **oOU** that has the name HQ. A name is a mandatory property.

It is important to remember that the object is not committed to Active Directory at this point. There is an ADSI object reference on the client computer that you can use to set or modify locally cached attributes such as the **Description**. To save the object back to the directory service, you call the **SetInfo** method.

Creating New Users

• **Creating a New User**

```
Set oOU = GetObject _  
    ("LDAP://OU=HQ,DC=fourthcoffee,DC=com")  
Set oUser = oOU.Create("user", "CN=Kathie Flood")  
  
oUser.Put "sAMAccountName", "kathief"  
oUser.Put "userPrincipalName", "kathie@fourthcoffee.com"  
  
oUser.SetInfo  
  
oUser.SetPassword "TempPa$$w0rd"  
oUser.AccountDisabled = False  
  
oUser.SetInfo
```

After you have created an OU, it is easy to create new users in that OU.

Creating a New User

In the following example, a new user is created in the HQ organizational unit.

```
Set oOU = GetObject ("LDAP://OU=HQ, C=fourthcoffee,DC=com")  
Set oUser = oOU.Create("user", "CN=Kathie Flood")  
  
oUser.Put "sAMAccountName", "kathief"  
oUser.Put "userPrincipalName", "kathie@fourthcoffee.com"  
  
oUser.SetInfo  
  
oUser.SetPassword "TempPa$$w0rd"  
oUser.AccountDisabled = False  
oUser.SetInfo
```

To create a user, you must provide two properties: the common name (CN) and the down-level NetBIOS user name (sAMAccountName).

Note that, in this example, the object is committed back to the directory service twice. This is because you can only use the properties **SetPassword** and **AccountDisabled** against an object that already exists in the directory. The first instance of **SetInfo** commits the new user object to Active Directory, and the second instance sets the password and enables the account.

Note: The password must meet the password policy requirements for minimum password length, complexity, and history. Setting the **userPrincipalName** property is optional when you use a script to create the user.

Creating New Groups

The diagram illustrates the process of creating a new group in Active Directory using ADSI scripts. It shows a main window with a tree view containing three items: "Group Types", "Properties and Methods of Groups", and "Creating Groups". A code snippet for creating a group is shown in the "Creating Groups" section. Below this, another tree view shows "Adding Users to Groups" and "Removing Users from Groups".

```
Set oOU = GetObject("LDAP://OU=HQ,DC=fourthcoffee,DC=com")
Set oGroup = oOU.Create("group", "CN=Script Users")
oGroup.Put "groupType" ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP _
Or ADS_GROUP_TYPE_SECURITY_ENABLED
oGroup.Put "SAMAccountName", "Script Users"
oGroup.Put "name", "Script Users"
oGroup.Put "displayName", "Script Users"
oGroup.Put "description", "Script Users Security Group"
oGroup.SetInfo
```

- Group Types
- Properties and Methods of Groups
- Creating Groups

oGroup.Add("LDAP://CN=MyUser,OU=HQ,DC=fourthcoffee,DC=com")

- Adding Users to Groups
- Removing Users from Groups

In addition to creating users with ADSI scripts, you can create new groups. However, note that Active Directory supports various types of groups.

Group Types

Windows Server 2003 has the following four basic group types:

- Universal groups

Universal groups can contain accounts and groups, except for local and domain local groups, from any domain in the forest. You can also add universal groups to local groups and other universal groups in any domain in the forest. These groups are only available when Active Directory is in native mode.

- Global groups

Global groups can contain only accounts and other groups from their own domain. These groups can be added to global, local, and universal groups in all domains in the forest.

- Domain local groups

Domain local groups can contain accounts and global and universal groups from any domain in the forest, in addition to other domain local groups from the same domain. You can only include them in access-control lists of the same domain in which they were created, and can only be a member of other domain local groups in the same domain.

- Local groups

Local groups exist only in the local computer's SAM. You can only manipulate them on that computer, and they can contain users and groups from any domain in the

forest. On domain controllers, this type of group maps straight to the domain local group type.

Each of these group types can be defined as either security (can be used for assigning permissions) or distribution (equivalent to an e-mail distribution list). If, when you write the script, you do not specify group type or security properties, a global security group will be created. However, if you only define the group type, a distribution group of that type will be created.

Properties and Methods of Groups

In addition to the standard properties and methods shown earlier in this module, groups expose specific properties and methods. The following table describes these.

Property	Description
Description	Provides a place for a string comment about the group.
Members	Returns a collection of the group's members.
IsMember	Returns True if the given ADsPath is a member of the group.
Add	Adds the given ADsPath to the group.
Remove	Removes the given ADsPath from the group.

Creating Groups

The following example shows how you can create groups using ADSI.

```
' Setup Constants
Const ADS_GROUP_TYPE_GLOBAL_GROUP = &H2
Const ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP = &H4
Const ADS_GROUP_TYPE_LOCAL_GROUP = &H4
Const ADS_GROUP_TYPE_UNIVERSAL_GROUP = &H8
Const ADS_GROUP_TYPE_SECURITY_ENABLED = &H80000000

' Bind to OU
Set oOU = GetObject("LDAP://OU=HQ,DC=fourthcoffee,DC=com")
' Create new group
Set oGroup = oOU.Create("group", "CN=Script Users")
' Set the group type
oGroup.Put "groupType", ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP Or _
ADS_GROUP_TYPE_SECURITY_ENABLED
' Set the group properties
oGroup.Put "sAMAccountName", "Script Users"
oGroup.Put "name", "Script Users"
oGroup.Put "displayName", "Script Users"
oGroup.Put "description", "Script Users Security Group"
' Save back to the AD
oGroup.SetInfo
```

In this script, the ADSI constants are declared first. Next, the script binds to the Users container in the fourthcoffee.com domain. Then, a new group called Script Users is

created by using the **Create** method of the **oOU** object. The **Put** method is then used to add the following information to the new group:

- The group type
- The NetBIOS sAMAccountName
- The display name
- The description

Finally, the **SetInfo** method writes this information from the object's local cache back to Active Directory.

The constants (and their values) that are used in the group creation process are listed in the following table.

Constant	Value
ADS_GROUP_TYPE_GLOBAL_GROUP	&H2
ADS_GROUP_TYPE_DOMAIN_LOCAL_GROUP	&H4
ADS_GROUP_TYPE_LOCAL_GROUP	&H4
ADS_GROUP_TYPE_UNIVERSAL_GROUP	&H8
ADS_GROUP_TYPE_SECURITY_ENABLED	&H80000000

Adding Users to Groups

You can use the **Add** method of the **Group** object to add a specific user account to that group. The following example adds a user to the group referenced by the **oGroup** variable.

```
' Bind to Group
Set oGroup = GetObject("LDAP://CN=Script Users,OU=HQ,DC=fourthcoffee,DC=com")
' Add user to group
oGroup.Add ("LDAP://CN=Kathie Flood,OU=HQ,DC=fourthcoffee,DC=com")
' Save back to the AD
oGroup.SetInfo
```

Removing Users from Groups

You can remove users from a group by using the **Remove** method of that **Group** object. The following example removes the user **Kathie Flood** from the Script Users group.

```
' Bind to Group
Set oGroup = GetObject("LDAP://CN=Script Users,OU=HQ,DC=fourthcoffee,DC=com")
' Remove user from group
oGroup.Remove ("LDAP://CN=Kathie Flood,OU=HQ,DC=fourthcoffee,DC=com")
' Save back to the AD
oGroup.SetInfo
```

Lesson 6: Managing Security, Shares, and Services by Using ADSI

- Setting Security in Active Directory
- Managing Shares by Using ADSI
- Controlling Services by Using ADSI

ADSI provides limited functionality for setting Active Directory security, managing network shares, and controlling system services. This lesson describes the functionality that is available and also provides information about alternative tools that offer more advanced options for managing security, shares, and services.

Objectives

After completing this lesson, you will be able to:

- Use scripts to set security for Active Directory objects.
- Manage network shares using ADSI.
- Control system services using ADSI.

Setting Security in Active Directory

- **Security Components**
- **Windows Server 2003 R2 Platform SDK**
- **Script vs. Dsacl.exe**

After you create a new object in Active Directory, you will probably need to modify the security permissions of the objects.

Security Components

Each object in Active Directory has a corresponding security descriptor that enables you to modify and propagate permissions on the object, and perform auditing and other tasks.

Each security descriptor has two access control lists (ACLs). These are the discretionary ACL (DACL) and the system ACL (SACL). The DACL contains all administrator-assigned permissions for the object. The SACL tracks changes to Active Directory in the security event log of the domain controller for audit purposes. Each ACL can contain access control entries (ACEs).

An ACE contains the following components:

- AccessMask
- AceType
- AceFlags
- Trustee
- Flags

Windows Server 2003 R2 Platform SDK

Manipulating security by using ADSI can involve a complex set of tasks. Before you try to write code that manipulates Active Directory security, refer to the Windows Server 2003 R2 Platform Software Development Kit (SDK).

Script vs. Dsacl.exe

Although you can script the manipulation of security in Active Directory, it is much easier to use the command-line tool Dsacl.exe, which is provided as part of the Windows Server 2003 support tools. Using this tool enables you to achieve all of the functionality of ADSI, but requires far less code.

For more information about this tool, see the Help file in the support tools for Windows Server 2003.

Managing Shares by Using ADSI

The screenshot shows a Windows Script Host window. On the left, there's a tree view with two nodes: "Binding to the LanmanServer Service" and "Properties". Under "Properties", several items are listed: Name, CurrentUserCount, Description, HostComputer, MaxUserCount, and Path. On the right, a code editor window displays the following VBS script:

```
Set objShare = GetObject("WinNT://MyComp/LANMANSERVER")
```

Managing shares by using ADSI is a simple process. However, ADSI provides limited functionality for creating, deleting and editing network shares. Windows Vista uses a new network-sharing architecture using Server Message Block (SMB) version 2.0, and ADSI cannot manage all aspects of Windows Vista file sharing. For more information about SMB 2.0, see “New Networking Features in Windows Server 2008 and Windows Vista” on the Microsoft TechNet Web site at www.microsoft.com/technet/network/evaluate/new_network.mspx. The alternative to ADSI is to use WMI. For more information about WMI, see Module 8, “WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Binding to the LanmanServer Service

The first step in using ADSI to create or modify a share in a script is to bind to the LanmanServer file service on the host computer. You cannot use the LDAP provider for this; instead you use the Windows NT provider with the syntax shown in the following example.

```
Set oLMServer = GetObject("WinNT://MyComputer/LanmanServer")
```

Note: You must use NetBIOS names with the LanmanServer service, because the Windows NT provider does not support DNS naming. This limitation also applies when using ADSI to manage network shares.

This code creates an instance of the **LanmanServer** object by binding to the LanmanServer file service on the computer called MyComputer.

Properties

After this binding occurs, and the object is successfully created, the properties shown in the following table are available.

Property	Description
Name	The share name.
CurrentUserCount	The number of users currently connected to the share (read-only).
Description	The friendly description for the file share.
HostComputer	The ADsPath for the computer where this share resides.
MaxUserCount	The maximum number of users who are allowed to connect to this share simultaneously.
Path	The local file system path for the shared folder.

The following example demonstrates how to enumerate all of the shares from a computer called MyComputer.

```
Set oLMServer=GetObject("WinNT://MyComputer/LANMANSERVER")
For Each share in oLMServer
    WScript.Echo share.Name & " = " & share.Path
Next
```

This example works as expected for Windows Server 2003, Microsoft Windows® XP and Windows 2000 computers. However, on Windows Vista this script always shows all shared folders in the Users folder that will be shared as “C:\Users,” unless the share is created using Windows Vista advanced sharing.

The following example shows how to create a share on MyComputer (“MyComp”) with the following properties:

- **Name** = DemoShare
- **Path** = C:\Temp
- **Description** = “This is a demo share”
- **User Limit** = 5

```
Set oLMServer=GetObject("WinNT://MyComp/LANMANSERVER")
Set oNewShare = oLMServer.Create("fileshare", "DemoShare")
oNewShare.Path = "C:\Temp"
oNewShare.Description = "This is a demo share"
oNewShare.MaxUserCount = 5
oNewShare.SetInfo
```

Note: This object has no methods that allow you to set permissions that will apply to the share. To apply share permissions using script you must use WMI. For more information about WMI, see Module 8, "WMI," in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

MCT USE ONLY. STUDENT USE PROHIBITED

Controlling Services by Using ADSI

- **Enumerating Services**
 - Testing service status
- **Starting a Service**
- **Stopping a Service**

ADSI provides limited features for controlling services on a computer running Windows 2000 or later. These features enable you to determine what services are running on a computer and to start and stop these services. More functionality for controlling services is available by using WMI. For more information about WMI, see Module 8, “WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

Enumerating Services

As with network shares, the first step in using ADSI to control services in a script is to bind to the LanmanServer file service on the host computer. You must use the Windows NT provider because you cannot use the LDAP provider for this. The Windows NT provider does not support DNS naming, so you must use NetBIOS names in scripts for ADSI services.

After you have done this, you must enumerate the existing services on a computer, as the following example shows.

```
Option Explicit
Dim objComputer, Service, strSvrList
Set objComputer = GetObject("WinNT://MyServer")
strSvrList = "The Services on " & objComputer.ADsPath _
    & " are " & vbCrLf
For Each Service in objComputer
    If Service.Class = "Service" Then
        strSvrList = strSvrList & Service.Name & vbCrLf
    End If
Next
WScript.Echo strSvrList
```

This example binds to the computer MyServer and then creates a string variable, strSvrList, which is used to hold the list of services. A **For...Next** loop is then created to loop through all of the elements in **objComputer**. For each of these elements, an **If...Then** statement tests whether the **Class** of the element is of the type **Service**. If it is, it is added to the strSvrList variable. After all of the elements have been tested, the **For...Next** loop exits, and the strSvrList variable is echoed.

To return only the services that are currently running, you can add a test of the services' status. You can use the following table of constants when testing for the status of these services:

Constant	Value
ADS_SERVICE_STOPPED	&H1
ADS_SERVICE_START_PENDING	&H2
ADS_SERVICE_STOP_PENDING	&H3
ADS_SERVICE_RUNNING	&H4
ADS_SERVICE_CONTINUE_PENDING	&H5
ADS_SERVICE_PAUSE_PENDING	&H6
ADS_SERVICE_PAUSED	&H7
ADS_SERVICE_ERROR	&H8

The following is an example of a script returning only the services that are running on the computer MyServer.

```
Option Explicit
Const ADS_SERVICE_RUNNING = &H4

Dim objComputer, Service, strSvrList

Set objComputer = GetObject("WinNT://MyServer")
strSvrList = "The Services on " & objComputer.ADsPath _
    & " are: " & vbCrLf
For Each Service in objComputer
    If Service.Class = "Service" Then
        If Service.Status = ADS_SERVICE_RUNNING Then
            strSvrList = strSvrList & Service.Name & vbCrLf
        End If
    End If
Next
WScript.Echo strSvrList
```

This script works in the same way as the previous example script, but an extra **If...Then** statement has been added to test that the status of a service is equal to **ADS_SERVICE_RUNNING** before its name is appended to the strSvrList string.

Starting a Service

If you know the name of the required service, the process of starting the service is very simple, as the following example shows.

```
Set oBrowser = GetObject("WinNT://MyServer/browser")
oBrowser.start
```

In this example, an **oBrowser** object is created that references the browser service on the computer MyServer. A call to the **Start** method starts the service.

Stopping a Service

Similarly, stopping a known service is very simple, as the following example shows.

```
Set oMessenger = GetObject("WinNT://MyServer/messenger")
oMessenger.stop
```

In this example, the object is called **oMessenger**, and the **Stop** method is used to stop the messenger service.

In real-world scripting, it is a good practice to also use the **Status** method to check that the service started or stopped correctly before terminating the script.

Lab B: Scripting Administrative Tasks by Using ADSI



- Exercise 1
Retrieving Properties by Using ADSI
- Exercise 2
Creating OUs, Users, and Groups by
Using ADSI

After completing this lab, you will be able to:

- Create scripts that bind to the domain by using various methods.
- Manipulate various objects by using the ADSI interface.

Estimated time to complete this lab: 45 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Make sure that the 2433B-LON-DC1 virtual machine is still running, and that you are still logged on as **Administrator** after Lab A.
- Make sure that the 2433B-VISTA-CL1-05 virtual machine is still running, and that you are still logged on as **FOURTHCOFFEE\Administrator** after Lab A.

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to use scripts to list common Active Directory objects such as users, computers, groups, and OUs. You want to use the ADSI interface to produce object lists. You also want to create and manipulate common objects by using the ADSI interface, and the helper scripts produced by the ADSI Scriptomatic tool available from the Microsoft Scripting Center.

Exercise 1

Retrieving Properties by Using ADSI

In this exercise, you will write code that interacts with ADSI to produce a list of Active Directory objects of a particular type: users, computers, groups, and OUs.

The principal tasks for this exercise are as follows:

- To create a new script by using a template.
- To investigate the List Objects template.
- To edit the List Objects template.
- To create scripts to list computers, groups, and OUs.

Task	Supporting information
1. To create a new script by using a template.	<ul style="list-style-type: none">• Start PrimalScript.• Open E:\Labfiles\Starter\ListObjectsTemplate.vbs.• Edit the script information header so that it includes your name and today's date.• Save the file as E:\Labfiles\Starter\ListUsers.vbs.
2. To investigate the List Objects template.	<ul style="list-style-type: none">• Locate the first lines of code, which are used to configure the error behavior, and declare the constant used in the script to specify that the search scope is subtree.• Locate the lines of code that are used to instantiate the Connection and Command objects.• Locate the lines of code that are used to manipulate the Connection and Command objects.• Locate the lines of code that are used to specify that the script must return a maximum of 1000 records, and that the script must search the whole Active Directory tree.• Locate the line of code that is used to create the query, and specify the type of object to return.• Locate the line of code that is used to create the Recordset object.• Locate the lines of code that are used to loop through the Recordset object, build a string from its contents, and display the string to the user.
3. To edit the List Objects template.	<ul style="list-style-type: none">• Edit the SELECT statement to specify that you want to search the fourthcoffee.com domain, where the objectCategory is user.• Save the script.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Run the script by double-clicking E:\Labfiles\Starter\ListUsers.vbs and review the details that are displayed in the message boxes.
4. To create scripts to list computers, groups, and OUs.	<ul style="list-style-type: none">• In PrimalScript, open E:\Labfiles\Starter\ListObjectsTemplate.vbs.• Edit the script information header so that it includes your name and today's date.

MOTICER ONLY. STUDENT USE PROHIBITED

Task	Supporting information
	<ul style="list-style-type: none">• Save the file as E:\Labfiles\Starter\ListComputers.vbs.• Edit the SELECT statement to specify that you want to search the fourthcoffee.com domain, where the objectCategory is computer.• Save the script.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Repeat the above steps, changing the objectCategory to group, and save the script as ListGroups.vbs.• Repeat the above steps, changing the objectCategory to organizationalunit, and save the script as ListOUs.vbs.• Close PrimalScript.

Questions

Q: Before you can use ADSI to manipulate the objects in a directory, what action must your script perform with an ADSI provider?

Q: What is the preferred method of binding?

Exercise 2

Creating OUs, Users, and Groups by Using ADSI

In this exercise, you will use the ADSI Scriptomatic tool to create standard ADSI scripts for creating OUs, users, and groups. You will then edit these standard scripts, so that you can use them in your domain environment.

The principal tasks for this exercise are:

- To install ADSI Scriptomatic.
- To use ADSI Scriptomatic to produce a “create user” script.
- To use PrimalScript to investigate the ADSI Scriptomatic code.
- To use PrimalScript to modify the ADSI Scriptomatic code.
- To use ADSI Scriptomatic to create a “create group” script.
- To use ADSI Scriptomatic and PrimalScript to add users to a group.

Task	Supporting information
1. To install ADSI Scriptomatic.	<ul style="list-style-type: none"> • Run the ADSI Scriptomatic installer by double-clicking E:\Labfiles\Starter\EZADScriptomatic.exe. • Unzip the ADSI Scriptomatic files to E:\Labfiles\Starter\ADSI Scriptomatic.
2. To use ADSI Scriptomatic to produce a “create user” script.	<ul style="list-style-type: none"> • Run ADSI Scriptomatic by double-clicking E:\Labfiles\Starter\ADSI Scriptomatic\EZADScriptomatic.hta. • In ADSI Scriptomatic, select the Create an Object task and the user class. • Edit the first two lines of the EzAD Scriptomatic code, so that strContainer is set to ou=2433Users, and strName is set to TestUser. • Run the script by using ADSI Scriptomatic. • Switch to the 2433B-LON-DC1 virtual machine. • Open Active Directory Users and Computers. • Open the 2433Users container, and verify that the user account TestUser has been created. Notice that this account is disabled.
3. To use PrimalScript to investigate the ADSI Scriptomatic code.	<ul style="list-style-type: none"> • Switch to the 2433B-VISTA-CL1-05 virtual machine. • Use ADSI Scriptomatic to save the script as E:\Labfiles\Starter\CreateUser.vbs. • Start PrimalScript. • Open E:\Labfiles\Starter\CreateUser.vbs. • Locate the two variables that are used in the script: strContainer and strName. • Locate the lines of code that are used to instantiate the Connection object. • Locate the lines of code that are used to create the user object, write the sAMAccountname property, and call the SetInfo method.

Task	Supporting information
4. To use PrimalScript to modify the ADSI Scriptomatic code.	<ul style="list-style-type: none"> • Edit the opening lines of the script by entering the following variables and values: <ul style="list-style-type: none"> • strContainer set to ou=2433Users • strName set to Kathie Flood • strFirstName set to Kathie • strLastName set to Flood • strDepartment set to MIS • strPassword set to TempPa\$\$w0rd • Use the .Create and .Put methods to write the account properties, and call the SetInfo method to write the changes to Active Directory. • Use the .SetPassword method to write the password, and use the .AccountDisabled=False method to enable the account. • Call the SetInfo method again to write the changes to Active Directory. • Save the script. • Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window. • Switch to the 2433B-LON-DC1 virtual machine. • In Active Directory Users and Computers, refresh the 2433Users container. • Verify that the user account Kathie Flood has been created, that this account is enabled, and that the object attributes for First name, Last name, and Department have been written.
5. To use ADSI Scriptomatic to create a “create group” script.	<ul style="list-style-type: none"> • Switch to the 2433-VISTA-CL1-05 virtual machine. • Run ADSI Scriptomatic by double-clicking E:\Labfiles\Starter\ADSI Scriptomatic\EZADScriptomatic.hta. • In ADSI Scriptomatic, select the Create an Object task and the group class. • Edit the sixth and seventh lines of the EzAD Scriptomatic code, so that strContainer is set to ou=2433Users, and strName is set to MISUsers. • Run the script by using ADSI Scriptomatic. • Switch to the 2433B-LON-DC1 virtual machine. • In Active Directory Users and Computers, refresh the 2433Users container. • Verify that the group MISUsers has been created.
6. To use ADSI Scriptomatic and PrimalScript to add users to a group.	<ul style="list-style-type: none"> • Switch to the 2433B-VISTA-CL1-05 virtual machine. • In ADSI Scriptomatic, select the Write an Object task and the group class. • Use ADSI Scriptomatic to save the script as E:\Labfiles\Starter\AddUsersToGroup.vbs. • Close ADSI Scriptomatic. • Switch to PrimalScript. • Open E:\Labfiles\Starter\AddUsersToGroup.vbs. • Edit the sixth and seventh lines of the script, so that strContainer is set to ou=2433Users, and strName is set to MISUsers.

Task	Supporting information
	<ul style="list-style-type: none">• Edit the lines that contain samAccountName, description, and mail, so that these three properties are set to MISUsers, MIS Users, and misusers@fourthcoffee.com respectively.• Edit the lines that contain member names to add Kathie Flood, April Reagan, and David Junca.• Edit the lines that contain managedBy names to add Kathie Flood.• Save the script.• Run the script in PrimalScript, and review the details that are displayed in the PrimalScript Output window.• Switch to the 2433B-LON-DC1 virtual machine.• In Active Directory Users and Computers, refresh the 2433Users container.• For the MISUsers object, verify that the object attributes for Description and E-mail have been written, and that the Member list and Managed By information has been updated.

Questions

Q: When you use the properties of an object, where are the updates stored?

Q: How are changes committed back to the directory from the local properties cache?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

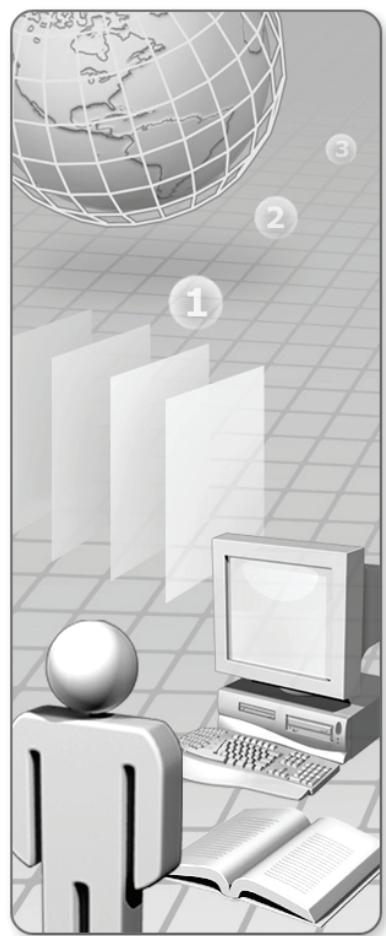
After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-05 virtual machines and discard any changes.

Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

Module 6: Creating Logon Scripts

Table of Contents

Module Overview	6-1
Lesson 1: Verifying the WSH Environment	6-2
Lesson 2: Common Logon Script Tasks	6-10
Lab A: Creating Logon Scripts	6-27
Lesson 3: Managing Logon Scripts	6-31
Lesson 4: Troubleshooting and Best Practices	6-41
Lab B: Assigning Logon Scripts	6-46



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Verifying the WSH Environment**
- **Common Logon Script Tasks**
- **Managing Logon Scripts**
- **Troubleshooting and Best Practices**

To deploy logon scripts successfully throughout the enterprise network, you must understand the processes involved in creating and managing these scripts. This module describes how to create and manage logon scripts.

Objectives

After completing this module, you will be able to:

- Check that the correct version of Windows® Script Host (WSH) is installed.
- Call logon scripts from batch files.
- Accomplish common tasks in logon scripts.
- Assign logon scripts to users.
- Describe common issues with logon scripts.
- Describe best practices for using logon scripts.

Lesson 1: Verifying the WSH Environment

- The Testing Process
- Checking the WSH Installation
- Checking the WSH Version
- Automatically Installing WSH
- Running the Logon Script

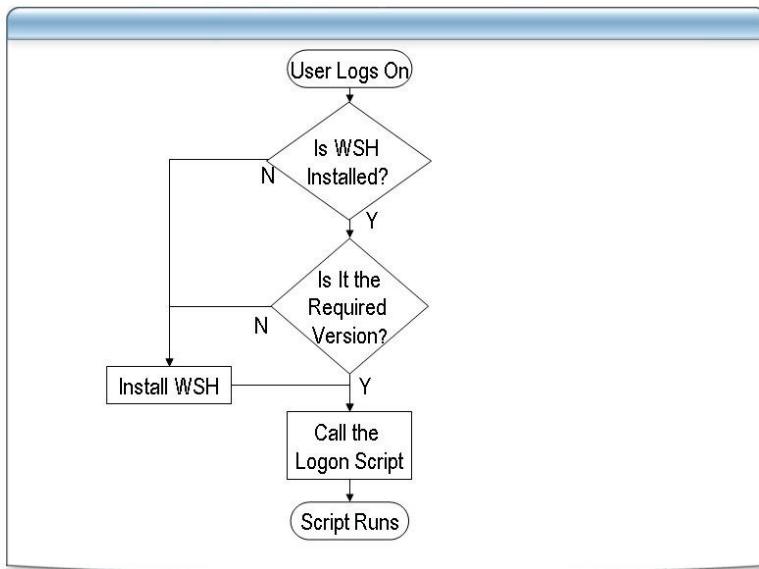
To ensure that your logon scripts run successfully, you must test whether the correct version of WSH is installed on the user's computer. This lesson teaches you how to verify whether the correct version of WSH is available to run your logon scripts. In addition, it describes how to install WSH on a user's system automatically if it is not already installed.

Objectives

After completing this lesson, you will be able to:

- Describe the process to ensure that the computer for which you are writing a script has the correct version of WSH installed.
- Write scripts that test whether WSH is installed on a computer and, if so, what version is installed.
- Write scripts that automatically install or update WSH on a computer.
- Run a logon script on a target computer.

The Testing Process



When a user logs on, you must verify whether your script can run successfully. This involves checking whether WSH is installed on the computer on which the script runs.

If WSH is installed, your script checks whether it is the correct version.

After the computer has passed these tests, your script can continue. If a test fails, your logon script installs the correct version of WSH before it continues.

Checking the WSH Installation

- **Calling Scripts from a Batch File**
 - Test for error level
- **Testing for the Presence of Installed Files**
 - Example: CScript.exe, WScript.exe, VBScript.dll

Before a logon script runs, you can test whether WSH is installed on a user's computer. To test this, you can either run a script from a batch file, or use a batch file to locate the WSH files. If script files are used, and WSH is not installed, then the tests fail.

Calling Scripts from a Batch File

You can create a batch file that attempts to run a script by using CScript. The batch file determines whether an error level is returned. If no error is returned, WSH is installed. You can then test the version of WSH.

The following is an example of a batch file that performs this test. If an error level is returned, the batch file installs WSH.

```
Rem Try and run wshvertest.vbs script
CScript //i //nologo WshVerTest.vbs
Rem If wshver fails install latest WSH in quiet mode
if errorlevel 1 WindowsXP-Windows2000-Script56-KB917344-x86-ene /q
```

Note: For more information about installing WSH, see Automatically Installing WSH later in this module.

Testing for the Presence of Installed Files

Another method that tests for a WSH installation is to search for the files that are present when WSH is installed.

The following files are present:

- Core script files
 - CScript.exe, WScript.exe, WScript.hlp

ScrRun.dll, Wshom.ocx, Wsnext.dll, Msrvct.dll, Scrobyj.dll, Dispex.dll, Advpack.dll

- Script engines
VBScript.dll, JScript.dll

To test for these files, you can use the **If Exist...** statement, as the following example shows.

```
Rem Test places file is usually found
If exist C:\Windows\system32\wshom.ocx goto :FoundFile
If exist C:\Windows\wshom.ocx goto :FoundFile
If exist C:\Windows\system\wshom.ocx goto :FoundFile
Rem No file found install latest WSH in quite mode
WindowsXP-Windows2000-Script56-KB917344-x86-ene /q
:FoundFile
Rem The file was found a version of WSH is installed
Rem Now launch the WSH version testing script
CScript //I //nologo WshVerTest.vbs
```

Tip: To make the process of using batch files easier, you can use the **Start** command to launch the logon batch file in a minimized window, as shown below:

```
START /MIN LOGON.BAT
```

Checking the WSH Version

- **Using Script**
 - **WScript.Version** property
 - Script engine test
 - ScriptEngineMajorVersion
 - ScriptEngineMinorVersion
- **Using Script or Batch Files**
 - File size or file date

Several versions of WSH are available. If you want your logon script to run successfully, your script must first check that the required version of WSH is installed on the client computer.

It is important to understand that there are two major parts to the WSH environment. The first is the host, which calls the relevant script engines and enables them to run. The second is the script engines themselves, which are the most likely to change. The script engine that ships with the Windows Vista® operating system is version 5.7. Version 5.6 ships with the Windows Server® 2003 operating system.

Using Script

You can check the version of WSH by using the properties of the WSH object model and the script engine. The following example tests for the version of WSH.

```
nVer = WScript.Version
If nVer >= 5.6 Then
    WScript.Echo "WSH Version is OK"
    WScript.Quit(0)
Else
    WScript.Echo "WSH needs updating"
    WScript.Quit (1)
End If
```

The following example checks the version of the Microsoft® Visual Basic®, Scripting Edition (VBScript) engine that processes the script.

```
nVerMaj = ScriptEngineMajorVersion
nVerMin = ScriptEngineMinorVersion
If nVerMaj = 5 and nVerMin >= 6 Then
    WScript.Echo "Script Engine version is OK"
    WScript.Quit(0)
Else
```

```
WScript.Echo "Script Engine needs updating"
WScript.Quit (1)
End If
```

Using Script or Batch Files

Another method for checking WSH and script engine versions is to test the size of the required files. This method has one possible advantage over the previous method: it can be conducted from within a batch file and in a script. The following example shows script that tests the size of the required files.

```
Dim oFso, ofile
Set oFso = CreateObject("Scripting.FileSystemObject")
Set ofile = oFso.GetFile("C:\Windows\system32\wshom.ocx")
If ofile.size = 114688 Then
    WScript.Echo "Wshom.ocx File OK"
Else
    WScript.Echo "Wshom.ocx needs updating"
End If
```

By updating the fourth line of this script, you can check the date when the file was originally created.

```
If ofile.DateCreated = "11/2/2006 1:49:18 AM" Then
```

Note: While these examples work on all computers that run Windows Vista, they do not work on Windows Server 2003. For Windows Server 2003 Service Pack 1, you must change the file size to 98304 or the date-created code to 3/25/2005 5:00:00 AM to reflect the Wshom.ocx file installed with that operating system.

Automatically Installing WSH

- **Installing WSH by Using Batch Files**
 - WindowsXP-Windows2000-Script56-Kb917344-x86-enu.exe for Windows XP and earlier versions
 - WindowsServer 2003-Script56-kb917344-x86-enu.exe for Windows Server 2003
- **Performing a Silent Installation**

If your tests fail to find the correct version of WSH installed, you can use the logon batch files to automatically install the correct version of WSH. This approach ensures that all client computers have installed the required version of WSH.

Installing WSH by Using Batch Files

WSH version 5.6 installs with the Windows® XP and Windows Server 2003 operating systems by default. If you include earlier versions of Windows than these in your network, you can use separate programs to update the host on those systems.

For Microsoft Windows® 2000 and earlier versions, use

WindowsXP-Windows2000-Script56-Kb917344-x86-enu.exe. For Windows Server 2003 and earlier versions, use WindowsServer2003-Script56-kb917344-x86-enu.exe. You can download either of these programs from the Microsoft Download Center Web site.

Windows Vista installs with WSH version 5.7, which is not available for download at the time of writing.

Note: Computers on your network that have earlier versions of Windows than Windows XP do not have WSH version 5.6 installed by default. You can upgrade any of them, including Windows Millennium Edition (Me), to WSH 5.6 by running the appropriate installation package.

Performing a Silent Installation

You can use several command-line switches when you install WSH. The /Q switch performs a silent installation.

This switch installs WSH with no user interaction. However, it may require you to restart the system.

Running the Logon Script

- **Calling the Script**
 - Directly: CScript //i //nologo LogonScript.vbs
 - With a .wsh file: CScript LogonScript.wsh
- **Determining the Script Path**
 - %LOGONSERVER%

After your batch files determine the presence of the correct version of WSH, they can call the logon script.

Calling the Script

Usually, your batch file calls the main script file. However, you can call a .wsh file that calls the logon script. Using this method gives you more control over how the script runs. The following examples illustrate both methods. You only use one of these options.

```
CScript //I //nologo LogonScript.vbs
```

```
CScript LogonScript.wsh
```

Determining the Script Path

To run the logon script, the batch file must determine the path to the script and how the script must be run. Windows XP and Windows Vista make this information available in the %LOGONSERVER% system variable. This variable is only available when a user logs on to a domain server. The %LOGONSERVER% variable contains the path to the domain controller that authenticates the client computer and user accounts.

You can use this variable to launch your logon scripts, as the following example shows.

```
CScript \\%LOGONSERVER%\Netlogon\Scripts\Logon.vbs
```

Lesson 2: Common Logon Script Tasks

- **Sending Messages**
- **Getting User Input**
- **Creating Shortcuts**
- **Mapping Drives**
- **Mapping Printers**
- **Launching Utility Programs**

There are several commonly performed tasks that you encounter when you develop logon scripts. This lesson explains how to send messages to the user, retrieve input from the user, create shortcuts, map drives and printers, and launch other programs and utilities.

Objectives

After completing this lesson, you will be able to:

- Write logon scripts that send messages to administrators.
- Write logon scripts that capture user input.
- Create shortcuts on users' desktops when they log on to a network or domain.
- Map network drives and printers when users log on to their computers.
- Launch utility programs when users log on to their computers.

Sending Messages

- **WScript.Echo**
 - Echoes to command line in CScript.exe
- **MsgBox**
 - Opens a message box
- **PopUp**
- **UserName**

You can include messages in logon scripts. The messages display to the user each time that the script runs. You can send messages to users to inform them of network events, or you can send a simple “Message of the Day” as part of a logon script. Messages can also display legal disclaimers and reminders.

There are several methods for displaying messages to users from your logon scripts.

WScript.Echo

If the host running the script is WScript.exe, the **WScript.Echo** method displays output in a dialog box. If the host running the script is CScript.exe, the **Wscript.Echo** method displays output in the command prompt window.

MsgBox

You can use the **MsgBox** function in Visual Basic, Scripting Edition if you want to display a message in a pop-up message box. This opens a message box in all versions of Windows when either script engine is used, even if the batch mode switch (`//B`) has been declared.

The **MsgBox** function also enables you to display various types of message boxes that have various button configurations and icon settings. For example, the **MsgBox** function can display any of the following icons in the message box:

- Critical Message
- Warning Query
- Warning Message
- Information Message

In addition, you can specify the buttons that you want to present in the message box. You can use the following combination of buttons:

- **OK** only
- **OK** and **Cancel**
- **Yes** and **No**
- **Yes**, **No**, and **Cancel**
- **Abort**, **Retry**, and **Ignore**
- **Retry** and **Cancel**

The **MsgBox** function accepts the **buttons** argument, which you can use to specify both the icons and the buttons for the message box.

The following table lists the possible values for the **buttons** argument. Note that you can use the intrinsic Visual Basic, Scripting Edition constants in place of the values to read and write the script more easily.

Constant	Value	Description
<code>vbOKOnly</code>	0	Display the OK button only.
<code>vbOKCancel</code>	1	Display the OK and Cancel buttons.
<code>vbAbortRetryIgnore</code>	2	Display the Abort , Retry , and Ignore buttons.
<code>vbYesNoCancel</code>	3	Display the Yes , No , and Cancel buttons.
<code>vbYesNo</code>	4	Display the Yes and No buttons.
<code>vbRetryCancel</code>	5	Display the Retry and Cancel buttons.
<code>vbCritical</code>	16	Display the Critical Message icon.
<code>vbQuestion</code>	32	Display the Warning Query icon.
<code>vbExclamation</code>	48	Display the Warning Message icon.
<code>vbInformation</code>	64	Display the Information Message icon.
<code>vbDefaultButton1</code>	0	The first button is the default.
<code>vbDefaultButton2</code>	256	The second button is the default.
<code>vbDefaultButton3</code>	512	The third button is the default.
<code>vbDefaultButton4</code>	768	The fourth button is the default.
<code>vbApplicationModal</code>	0	The user must respond to the message box before the script continues.
<code>vbSystemModal</code>	4096	This constant provides an application modal message box that always remains on top of any other programs running.

To specify both an icon and a set of buttons, you can add the numbers or constants, as the following example shows.

```
MsgBox "OK button and Info Icon", vbOKOnly + vbInformation
```

PopUp

The WSH object model provides a method that enables you to display a pop-up window. The **PopUp** method of the **WScript.Shell** object has features that are similar to the **MsgBox** function. However, the **PopUp** method adds a time value that governs how long a message is displayed before it is dismissed automatically. The following example illustrates the syntax.

```
oShell.Popup(Text, [Secs2Wait], [Title], [Type])
```

Text is the body of the message, Secs2Wait is the number of seconds that the message is displayed, Title is the title bar message, and Type is the type of buttons and icons displayed in the pop-up window.

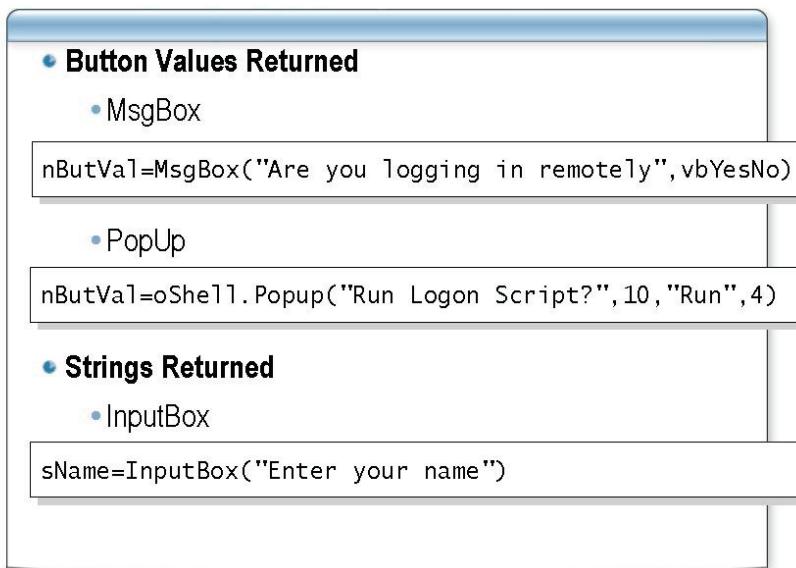
Note: For more information about the **PopUp** method, see the WSH documentation on the Microsoft TechNet Web site.

UserName

If you want the script to display the name of the user who is logging on, you can use the **UserName** property of the **WScript.Network** object to display the user's logon name. This personalizes the messages to the user, as the following example shows.

```
oNetwork = CreateObject("WScript.Network")
WScript.Echo "Welcome " & oNetwork.UserName
```

Getting User Input



Your scripts must often retrieve information from the user at script run time. You can use this data later in the script to adjust the way that the rest of the script runs, based on the answers the user provides. This information can be in the form of simple Yes or No answers or as more complex message strings.

Button Values Returned

Two functions collect information from a user while the logon script runs. Simple Yes or No answers collect integer values that correspond to the button that the user clicks.

MsgBox

In addition to specifying the buttons displayed in a message box, the **MsgBox** function returns a value to your script, indicating which button the user has clicked. Each button returns a different number in the range 1 to 7 because various buttons are available. The following table documents the return values of the **MsgBox** function.

Button	Value	Intrinsic constant
OK	1	vbOK
Cancel	2	vbCancel
Abort	3	vbAbort
Retry	4	vbRetry
Ignore	5	vbIgnore
Yes	6	vbYes
No	7	vbNo

The following example displays a message asking users whether they have logged on by using Remote Access Service (RAS). When the users click a button, the corresponding value is stored in the variable vMyVal. If vMyVal is 6, the user clicked the **Yes** button. As a result, the first message is displayed. If vMyVal is not 6, the second message is displayed.

```
vMyVal = MsgBox ("Logging on using RAS?", vbYesNo)
If vMyVal = 6 Then
    WScript.echo "User Logging in Remotely"
Else
    WScript.Echo "User Logging in on LAN"
End If
```

Note that you can use the intrinsic constant **vbYes** in place of the literal number 6.

PopUp

The **PopUp** method of the **WScript.Shell** object can also collect user feedback in a manner similar to the **MsgBox** function. The message box types, buttons, and return values are identical to the **MsgBox** function. However, as previously mentioned, you can configure this method so that it times out if no input is forthcoming.

If the **PopUp** method times out before the user clicks a button, it returns a value of **-1**. You must always check for this value in addition to the values that are returned by users when they click a button.

Strings Returned

The **InputBox** function can return a text string typed by the user to your script. If you employ this function, you must use error handling to ensure that appropriate input is received from the user.

The **InputBox** function has the following syntax.

```
InputBox(prompt[, title][, default][, xpos][, ypos][, -helpfile, context])
```

The **prompt** and **title** arguments provide the text for the message box and title bar respectively, similar to the **MsgBox** and **PopUp** functions. The **default** argument specifies a default text value for the input box.

The **InputBox** function also gives you a high level of control over the position of the message box on the screen through the **xpos** and **ypos** arguments. The **helpfile** argument enables you to access a help file if you press the F1 key while the message box is displayed. The **context** argument specifies which page of the help file must be displayed.

An example of the **InputBox** function is shown below.

```
Option Explicit
Dim vFirstName, vLastName, vFullName
vFirstName = InputBox ("Enter your first name", "First Name")
vLastName = InputBox ("Enter your last name", "Last Name")
vFullName = vFirstName & " " & vLastName
WScript.Echo vFullName
```

In this example, the first line prompts users for their first name and assigns the answer to the variable vFirstName. The second line prompts users for their last name and assigns it to the variable vLastName. These values are concatenated and stored in the variable vFullName, which the script then echoes to the screen.

Creating Shortcuts

- **Two Shortcut Types**
 - Standard file (.lnk)
 - Universal resource locator file (.url)
- **Copying Existing Files**
 - Using the **SpecialFolders** property
- **Using the WScript.Shell Object**

```
Set oIntraLnk=oShell.CreateShortcut("Path\Intranet.URL")
oIntraLnk.TargetPath = "http://Intra.nwtraders.msft"
oIntraLnk.Save
```

As an administrator, you may want to control user desktop environments.

Windows Server 2003 Group Policy is the preferred way to customize user desktop environments in an organization. However, if you cannot use Group Policy settings for some reason, you can use scripts. Group Policy settings are much easier to manage and update than scripts, so you must only use scripts when you have no other alternative.

Two Shortcut Types

You can use scripts to provide two types of shortcuts: standard file shortcuts (.lnk) and URL file shortcuts (.url). You can create these links in two ways.

Copying Existing Files

You can copy an existing .lnk or .url file from the server to a shortcut folder such as the desktop or programs folder. You can access these folders by using the **SpecialFolders** property of the **WScript.Shell** object.

```
strDesktopPath = oShell.SpecialFolders("Desktop")
strProgramsPath = oShell.SpecialFolders("Programs")
```

Note: The Programs special folder references the Programs folder of the user who is logged on, not the Program Files folder. On Windows XP and Windows Server 2003, the path to this folder is C:\Documents and Settings\Username\Start Menu\Programs. On Windows Vista, the path to this folder is C:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu\Folders.

Using the WScript.Shell Object

Alternatively, you can use the **CreateShortcut** method of the **WScript.Shell** object, supplying the path and shortcut name. If you do not specify a path, the shortcut is created

in the current working folder of the script file. You can then set the **TargetPath** string property with the path to the file. Finally, you must save your changes. The following example shows how to create a .lnk shortcut on the desktop. It uses the **ScriptFullName** property to define the target path.

```
Dim wshShell, strDesktop, oLink  
  
Set wshShell = CreateObject("WScript.Shell")  
Set strDesktop = wshShell.SpecialFolders("Desktop")  
  
Set oLink = oShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")  
oLink.TargetPath = WScript.ScriptFullName  
oLink.Save
```

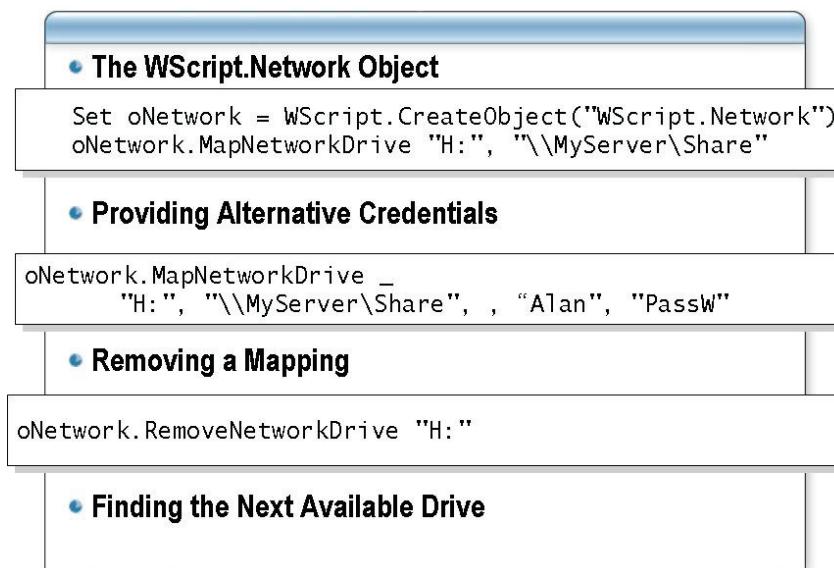
The following example shows how to create a URL shortcut.

```
Set oUrlLink = wshShell.CreateShortcut("MSWeb Site.URL")  
oUrlLink.TargetPath = "http://www.microsoft.com"  
oUrlLink.Save
```

Notice that, for this type of link, the **TargetPath** property is set to the URL for the Web site to which the link opens a browser window.

You can also set other properties of the shortcut, such as **IconLocation**, **Description**, and **WindowStyle**.

Mapping Drives



Mapping network drives is another common task that you may want to perform during the logon process.

The WScript.Network Object

The **WScript.Network** object provides the functionality to map drives through the **MapNetworkDrive** method. The following syntax shows how you can use the method.

```
Object.MapNetworkDrive strLocalName, strRemoteName, [bUpdateProfile], [strUser],  
[strPassword]
```

The previous syntax signifies the following:

- The **strLocalName** argument is the new drive letter that you want to map.
- The **strRemoteName** argument is the share that you want to map to.
- The **bUpdateProfile** argument is an optional Boolean value that indicates whether to save the new mapping in the user profile (the default is **False**).
- The **strUser** argument is an optional alternative user identifier.
- The **strPassword** argument is the optional password for the alternative user.

The following example maps drive H to a shared folder.

```
Dim WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.MapNetworkDrive "H:", "\Server\Public"
```

Note that the security credentials in this example are those of the user running the script.

Providing Alternative Credentials

You can supply alternative security credentials when mapping drives, if necessary. The following example uses alternative credentials to map drive H.

```
WshNetwork.MapNetworkDrive "H:", "\\\Server\Public1",,  
    "Alan", "PassW"
```

Removing a Mapping

To remove a mapping, you can use the **RemoveNetworkDrive** method of the **Network** object. You must pass the drive letter as an argument. The following syntax shows how you can use the method.

```
Object.RemoveNetworkDrive strName, [bForce], [bUpdateProfile]
```

The previous syntax signifies the following:

- The **strName** argument is the drive letter from which you want to remove the mapping.
- The **bForce** argument is an optional Boolean value that indicates whether to force the removal of the mapped drive.
- The **bUpdateProfile** argument is an optional Boolean value that indicates whether to remove the mapping from the user profile (the default is **False**).

The following example removes the mapping from drive H.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")  
WshNetwork.RemoveNetworkDrive "H:"
```

Finding the Next Available Drive

To find the next available drive that you can map, there are two methods that you can use.

The **EnumNetworkDrives** method of the **WScript.Network** object returns a collection that stores the local drives as the first items, followed by any current share mappings. As the method name suggests, it lists only the currently mapped network drives. If no network drive is mapped, no values are returned. The following script is an example of the use of the **EnumNetworkDrives** method.

```
Set oNetwork = CreateObject("WScript.Network")  
Set colDrives = oNetwork.EnumNetworkDrives  
  
For Each objDrive in colDrives  
    WScript.Echo objDrive  
Next
```

This produces an output similar to the following when running CScript.exe from a console window.

```
E:  
\\Myserver\data  
F:  
\\Myserver\copy  
G:
```

```
\Myserver\personal
```

The second method uses the **FileSystemObject** object from the Windows Script Runtime to view all of the system's drives. This method is advantageous, because it returns information about local drives and network drives.

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set colDrives = objFSO.Drives

For Each objDrive in colDrives
    Wscript.Echo "Drive letter: " & objDrive.DriveLetter
Next
```

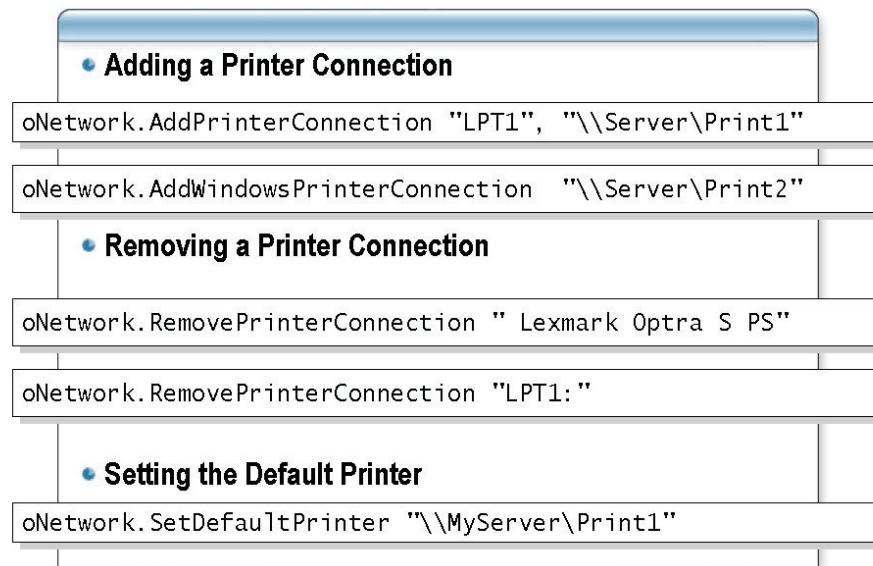
After the last drive letter has been found, you can use the **ASC** and **CHR** functions in Visual Basic, Scripting Edition to calculate the next letter in the alphabet. This option is useful when writing a script that maps a new drive while keeping all existing mappings in place.

The following example shows how to use this approach.

```
Option Explicit
On Error Resume Next
Dim wshNetwork, oFSO, colDrives, Drive, sNextDrive, sFinalDrive
'Create Network, FSO, and colDrives objects
Set wshNetwork = CreateObject("WScript.Network")
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set colDrives = oFSO.Drives

'Loop Through all used drives
For Each Drive in colDrives
    'Take the first letter from the drive
    sFinalDrive = Left(Drive, 1)
Next
'Convert final used drive to ASC, add 1 and return to 'character
sNextDrive = Chr(Asc(sFinalDrive) + 1) & ":"
'Map new drive
oNetwork.MapNetworkDrive sNextDrive, "\MyServer\Share"
```

Mapping Printers



Adding a printer object in a script is similar to adding a drive object.

Adding a Printer Connection

The **AddPrinterConnection** method of the **WScript.Network** object uses syntax similar to the **MapNetworkDrive** method.

```
object.AddPrinterConnection strLocalName,  
→strRemoteName[,bUpdateProfile][,strUser][,strPassword]
```

In the above syntax, the **strLocalName** argument refers to a local port, and the **strRemoteName** argument refers to a shared printer. This method is limited to the number of line printer (LPT) ports on the system.

The following example maps LPT1 to a shared printer.

```
oNetwork.AddPrinterConnection "LPT1", "\\\Server\Print1"
```

You can avoid the limitation of only using the number of LPT ports on the system, by using the **AddWindowsPrinterConnection** method with Windows-based client computers. The following example shows the syntax for this method.

```
object.AddWindowsPrinterConnection(strPrinterPath)
```

In this syntax, the **strPrinterPath** argument is the full Universal Naming Convention (UNC) printer path.

You can enumerate printers like drive mappings. You can use the **EnumPrinterConnections** method in the same way as the **EnumNetworkDrives** method, as the following example shows.

```
Set wshNetwork = CreateObject("WScript.Network")  
Set oPrinters = wshNetwork.EnumPrinterConnections
```

```
WScript.Echo "Network printer mappings: "
For i=0 to oPrinters.Count - 1 Step 2
    WScript.Echo "Port " & oPrinters.Item(i) & " = " & oPrinters.Item(+1)
Next
```

Removing a Printer Connection

Removing a mapping from a printer is a simple task. The **RemovePrinterConnection** method has only one required argument, which is the local name of the printer, such as LPT1, or the remote name if no local name exists. Two optional arguments enable you to remove the printer, even if it is in use, and update the user's profile.

```
object.RemovePrinterConnection strName, [bForce], [bUpdateProfile]
```

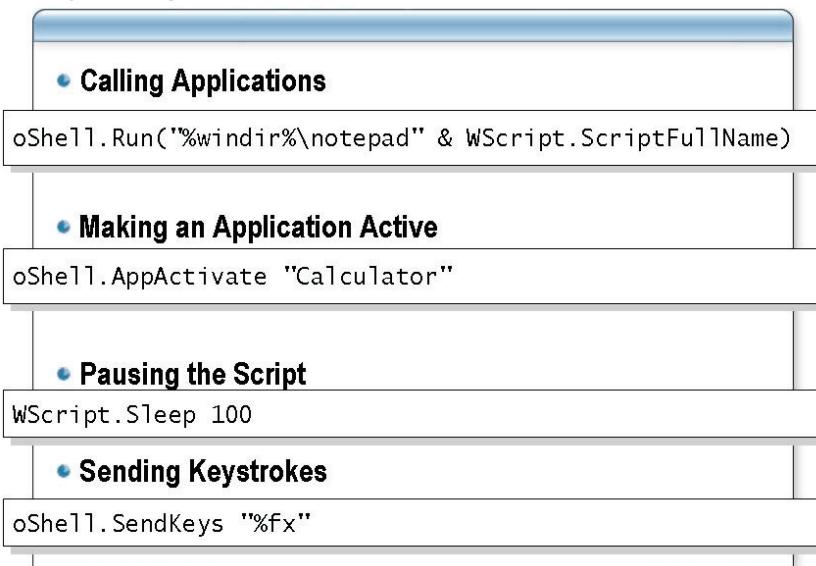
Setting the Default Printer

You can set a printer as the default printer by using the **SetDefaultPrinter** method and passing the remote printer name as an argument, as the following example shows.

```
oNetwork.SetDefaultPrinter "\\\MyServer\Print1"
```

Note: There is a dynamic-link library (DLL) called PrnAdmin.dll that is provided with the Windows Server 2003 Resource Kit. This DLL contains an object that provides many properties and methods that you can use to manage printers. For more information, search the Microsoft Web site for PrnAdmin, or download the Windows Server 2003 Resource Kit Tools from the Microsoft Download Center Web site.

Launching Utility Programs



Your logon scripts may need to launch other programs to perform certain tasks such as virus scanning, disk checking, or other maintenance utilities. The simplest way to launch another program is to use the **Run** method of the **WScript.Shell** object, as the following example shows.

```
object.Run (strCommand, [intWindowStyle], [bWaitOnReturn])
```

The previous syntax signifies the following:

- The **strCommand** argument is the command to run.
- The **intWindowStyle** argument is the window style for the new program, such as minimized or maximized.
- The **bWaitOnReturn** argument enables the script to continue to run without waiting for the new program to end (this is the default setting) or forces the script to wait until the program ends.

Calling Applications

The following example shows how to run Notepad from a script. In addition to starting Notepad, it also passes the **Run** method to the name of the file that the script is to open.

```
oShell.Run "%windir%\notepad.exe" & WScript.ScriptFullName
```

Making an Application Active

You can change the active program by using the **AppActivate** method of the **WScript.Shell** object. In this method, you specify the title of the window that you want to activate.

Pausing the Script

If you want to pause a script, you can call the **Sleep** method of the **WScript** object. You specify the length of time delay in milliseconds.

Sending Keystrokes

You can send keystrokes to another program by first calling the **AppActivate** method and then calling the **SendKeys** method of the **WScript.Shell** object. The effect of this is like a user entering keystrokes manually. By using this method, you can run menu items and shortcut keys such as %fx, which is like a user pressing ALT+F and then X. This key combination closes the program through the **File** menu.

The following table includes a complete listing of various control keys.

Key	Code
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HOME	{HOME}
INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}

Key	Code
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To specify keys that appear in combination with the SHIFT, CTRL, or ALT keys, precede the key code with one or more of the codes listed in the following table.

Key	Code
SHIFT	+
CTRL	^
ALT	%

Caution: Using the **SendKeys** method can be very useful, but it is important to understand its limitations. It is easy for a user to change the focus of a window while the script is running. This change then causes the script to fail. **SendKeys** is not a robust method and must only be used when there is no other way to control an application.

Lab A: Creating Logon Scripts



- Exercise 1
Mapping Drives
- Exercise 2
Creating Shortcuts
- Exercise 3
Assigning Logon Scripts to a User

After completing this lab, you will be able to:

- Create a logon script that maps a network drive
- Create a logon script that creates a desktop shortcut.
- Assign a logon script to a user account on Microsoft Active Directory® directory services.

Estimated time to complete this lab: 30 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine first and then start the 2433B-VISTA-CL1-06 virtual machine.
- Log on to each virtual machine as **FOURTHCOFFEE\Administrator** with a password of **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to create a logon script for users when they log on to the FourthCoffee domain. The script that you create will map a network drive to a user's computer and create a shortcut to a file on the user's desktop. You must then assign the logon script to the user by using Active Directory Users and Computers on the domain controller.

Exercise 1

Mapping Drives

In this exercise, you will create a logon script that maps a drive for a user of the FourthCoffee domain.

The principal tasks for this exercise are as follows:

- To map network drives.

Task	Supporting information
1. To map network drives.	<ul style="list-style-type: none">• On the 2433B-VISTA-CL1-06 virtual machine, start PrimalScript.• Using the Script Files template list, create a new VBScript file.• Declare six variables: objNetwork, objDrive, objFSO, strFinalDrive, intChar, and strNewDrive.• Instantiate the WScript.Network object and assign it to the objNetwork variable.• Instantiate the FileSystemObject object and assign it to the objFSO variable.• Use the objDrive variable to obtain each drive in the FileSystemObject.Drives collection.• Use string conversion and manipulation to obtain the next drive letter in sequence on the virtual machine and assign it to the strNewDrive variable.• Call the MapNetworkDrive method and map the new drive to \\LON-DC1\LabShare. Do this by using the objNetwork variable and use the strNewDrive variable to assign the next drive letter to the new network drive.

Questions

Q: What mechanisms can you use to gather user input in a logon script?

Exercise 2

Creating Shortcuts

In this exercise, you will create a shortcut to a text file by using Visual Basic, Scripting Edition.

The principal tasks for this exercise are:

- To create a shortcut to a text file.
- To test the script.

Task	Supporting information
1. To create a shortcut to a text file.	<ul style="list-style-type: none"> • Declare four more variables to the open VBScript file in PrimalScript: objShell, objShortcut, strDesktop, and strLnkPath. • Instantiate the WScript.Shell object and assign it to the objShell variable. • Assign strDesktop to the Desktop special folder by using the Shell object. • Assign objShortcut to the result of the Shell object's CreateShortcut method by concatenating strDesktop with "readme.lnk:". • Assign E:\LabFiles\Solution\Readme.txt to the shortcut's TargetPath property. • Save the shortcut. • Use the WScript.Echo method to send a message to the user that the logon script has completed.
2. To test the script.	<ul style="list-style-type: none"> • Use the Run Script command in PrimalScript to test the script on the client virtual machine. • Check that the script created a shortcut on the virtual machine's desktop and that it mapped the network drive. • Remove shortcut and disconnect from the new network drive.

Questions

Q: What does the command-line variable %LOGONSERVER% return?

Exercise 3

Assigning Logon Scripts to a User

In this exercise, you will assign the logon script that you have just created to an Active Directory user on the FourthCoffee domain. You will then test that it runs as expected when the user logs on to the domain.

The principal tasks for this exercise are:

- To assign the script to a user.
- To verify that the script runs correctly during the logon process.

Task	Supporting information
1. To assign the script to a user.	<ul style="list-style-type: none">• Use the Save As command in PrimalScript to save the logon script to the \\LON-DC1\sysvol\fourthcoffee.com\scripts folder.• On the domain controller, 2433-LON-DC1, use the following steps to assign the logon script to a user in the Active Directory domain.• Click Start, point to All Programs, point to Administrative Tools, and then click Active Directory Users and Computers.• Click 2433Users.• In the objects pane, right-click April Reagan, and then click Properties.• In the AprilReagan Properties dialog box, on the Profile tab, in the Logon script box, type .\LogonScript1.vbs and then click OK.• Minimize the domain controller virtual machine.
2. To verify that the script runs correctly during the logon process.	<ul style="list-style-type: none">• Switch to the client virtual machine and log off.• Log off the client computer.• Use the following steps to log on to the client virtual machine.• In the User name box, type AprilReagan.• In the Password box, type Pa\$\$w0rd and then press ENTER.• Verify that the shortcut exists on the virtual machine desktop.• Use the following steps to verify that the script mapped the network drive as expected.• Click Start, right-click Computer, and then click F:\LON-DC1\LabShare.

Questions

Q: What object exposes the **MapNetworkDrive** method?

Lesson 3: Managing Logon Scripts

- **Assigning Logon Scripts**
- **Client-Side Extensions**
- **Securing WSH**
- **Protecting Logon Scripts**

In addition to performing the common tasks outlined so far in this module, you must also successfully manage the logon scripts in your enterprise network. This lesson describes assigning logon scripts, client-side extensions, securing the WSH environment, protecting logon scripts, and troubleshooting logon scripts. It also outlines good practices for managing your logon scripts.

Objectives

After completing this lesson, you will be able to:

- Assign logon scripts to individual users.
- Understand Group Policy client-side extensions and when to use each of them.
- Analyze what steps to perform to secure the WSH environment.
- Describe the steps that you can take to protect your logon scripts.

Assigning Logon Scripts

- **Using Active Directory**
 - Assign logon scripts to individual accounts
- **Using Group Policy Client-Side Extensions**
 - User logon
 - User logoff
 - Computer startup
 - Computer shutdown
 - Policy updates and tool updates

Windows Server 2003 provides two mechanisms for assigning a logon script to a user account. You can do this through the user properties in Active Directory or through Group Policy.

Using Active Directory

In the Active Directory Users and Computers snap-in, you can specify a logon script for each user as part of the user properties. This approach enables you to assign logon scripts to individual user accounts.

Using Group Policy Client-Side Extensions

Group Policy provides centralized control over the configuration of computer and user environments across your organization. This includes the assignment of logon scripts to users, groups of users, and computers.

To assign logon scripts to user and computer groups, it is recommended that you use Group Policy Objects (GPOs). Group Policy gives you access to user logon and logoff client-side extensions and to computer startup and shutdown client-side extensions. By using this method, you can also assign multiple logon scripts to a collection of users and computers.

When assigning scripts in this way, you have greater control over how the script runs.

You can configure Group Policy to behave in a particular way by using script options settings for the computer group and the user group. The user group options are:

- Run logon scripts synchronously.

Enable this setting to force the system to run user logon scripts synchronously, one after another. This option also exists for Computer configuration, which can have a

different value. In case of conflict, the Computer configuration setting prevails. On computers running Windows XP and Windows Vista, Windows Explorer starts before the scripts finish running.

- Run legacy logon scripts hidden.

Enable this setting to run legacy, Windows NT version 4–type logon scripts in a hidden window. By default, these scripts run in a visible window and, therefore, users can cancel them.

- Run logon scripts visible.

Enable this setting to run logon scripts in a visible window.

- Run logoff scripts synchronously.

Enable this setting to run logoff scripts before the user logs off.

Important: If you use Group Policy to deploy your logon and logoff scripts, make sure that a large number of scripts do not run synchronously. The overall script execution time can cause an unacceptable delay in the logon time for users. Always test the scripts to make sure that the execution time is acceptable before assigning the logon script in the production environment.

The computer group options are:

- Run logon scripts synchronously.

Enable this setting to force the system to run logon scripts synchronously, one after another. This option also exists for Computer configuration, which can have a different value. In case of conflict, the Computer configuration setting prevails.

Note that this setting takes precedence over the user version of this setting.

- Run startup scripts asynchronously.

Enable this setting to optimize the startup/logon process so that users can log on before startup scripts have finished. The default is for each script to run synchronously (and hidden).

- Run startup scripts visible.

Enable this setting to run startup scripts in a visible window.

- Run shutdown scripts visible.

Enable this setting to run shutdown scripts in a visible window.

- Maximum wait time for Group Policy scripts.

Use this option to set the script time-out interval in seconds. If this time-out is reached, script processing is forcibly stopped, and an error is reported in the system event log.

This setting can be very important, because many scripts require some sort of user interaction. For example, you may write a startup script to connect to a domain

resource. If the account password changes, a dialog box is displayed that prompts the user to type the correct password.

However, in a startup script, this dialog box is not displayed, because the startup script is not interactive. As a result, the system appears to stop responding. In this situation, the setting for maximum wait time for Group Policy scripts governs how long the computer stays in this state before the script is forcibly stopped. By default, the system waits for 600 seconds (ten minutes). Valid values range from 0 to 32,000.

MCT USE ONLY. STUDENT USE PROHIBITED

Client-Side Extensions

- **Order of Execution**
 1. Computer startup scripts
 2. User logon scripts
 3. User logoff scripts
 4. Computer shutdown scripts

Windows Server 2003 Group Policy enables you to define the scripts to run when specific events occur on a user or computer group. These scripts are enabled by the Group Policy client-side extension that is installed on every computer that has a user or computer account managed by Group Policy. The Group Policy client-side extension enables you to run multiple scripts when these events occur. The events for which you most commonly create scripts include user logon, user logoff, computer startup, and computer shutdown. However, you can also use Group Policy to assign scripts to run when the following Group Policy events occur: the User policy refresh event, the User policy force refresh event, the Computer policy refresh event, and the Computer policy force refresh event. You can also assign scripts to run after you use the secedit tool or gpupdate tool.

Order of Execution

The order of execution for the client-side extension events is as follows:

1. Computer startup scripts run when the computer has been turned on and the operating system has started.
2. User logon scripts run after the user has been successfully authenticated.
3. User logoff scripts run when the user has chosen to log off or shut down the computer.
4. Computer shutdown scripts run when the computer shuts down. If a user has a logoff script, it runs before the shutdown script runs.

Securing WSH

- **Change the Default Application Handler for Script Files**
 - Change default action on .vbs .vbe .js .jse .wsf
- **Restrict Access to Host**
 - NTFS file system permissions
- **Sign Scripts with Digital Signatures**
 - Specify trusted script authors
 - Verify that the script has not been changed
- **Restrict the Ability of a User to Run a Script**
 - Edit the registry
- **Disable WSH**
 - Edit the registry

It is important that you understand how to secure the WSH environment to prevent the proliferation of script-based viruses in your enterprise network.

Script-based viruses show that creators of viruses can use script. You can take several steps to minimize the risk of these viruses by using the WSH environment.

Change the Default Application Handler for Script Files

One simple way to prevent script viruses is to change the default application handler for script files.

The following procedure changes the application that opens .vbs, .vbe, .js, .jse, or .wsf files. If those files open, they open by using Notepad.exe and not by using the application handler.

1. Double-click **My Computer**.
2. On the **Tools** menu, click **Folder Options**.
3. On the **File Types** tab, scroll down to the first item in the list above .jse, which is a Microsoft JScript®-encoded file.
4. Click the **.jse** file, and then click **Edit**.
5. In the Action window, click **Open**, and then click **Edit**.
6. You will see the following line of code.

```
C:\Windows\System32\WScript.exe "%1" %*
```

Change it to the following code.

```
C:\Windows\System32\Notepad.exe "%1" %*
```

7. Click **OK**, and then click **Close**.

Repeat these steps for each of the other file types listed previously.

Note: Many administrators and users reconfigure the default method of files with a .vbs or .js extension on their systems so that they open rather than run when invoked. This is an effective countermeasure that you can use to prevent the spread of script viruses.

On systems that have this configuration, startup and shutdown scripts run correctly. Logon and logoff scripts fail, because they use the default method. As a result, instead of the logon script running normally, users see the script in an editor (such as Notepad) when the user logs on and logs off. Users can work around this by calling the login script directly from within a batch file.

Restrict Access to Host

It is possible to further restrict the WSH environment by using NTFS file system (NTFS) permissions to control how users run scripts. You can best achieve this by concentrating on the host executables, because they are the simplest files on which to set permissions. For WSH, the host files are CScript.exe and WScript.exe. By setting appropriate NTFS permissions on these files, you can restrict the ability of a user to run scripts.

Note: For more information about this subject, go to the Security Center on the Microsoft TechNet Web site.

For additional information, see “Information on Preventing Certain Types of Software from Running Automatically” in the Knowledge Base on the Microsoft Help and Support Web site.

Sign Scripts with Digital Signatures

Digital signatures, which were introduced in WSH 5.6, provide a way for you to verify who authored a script and guarantee that a script has not been altered after it was written and signed. This does not necessarily mean that the script is safe—virus writers can obtain digital signatures, too. However, digital signatures do provide two measures of protection:

- You can specify which script writers are to be trusted and which are not. For example, you can specify that only scripts that are signed using a certificate issued by a trusted authority can be run in your organization. If you list Verisign as the only trusted authority, then only scripts signed with a certificate issued by Verisign will be considered safe and be allowed to run. However, if malevolent script writers have obtained a certificate from Verisign, then any scripts that they write will be considered safe, even if they are not.
- You can be sure that the script has not been changed since the time it was written and signed. If you know that the script in the ThirdPartyScript.vbs file is safe to use in your organization, you can distribute the script with the knowledge that no one can

modify the script without violating the digital signature. When a script is signed, a hash is derived and used to verify the signature. If the script has been modified in any way, the hash will be invalid, and the script will be reported as having been altered.

When a script is signed, the digital signature is appended to the end of the file as a set of comments. Adding the signature as commented lines ensures that the script can still run on previous versions of WSH.

Restrict the Ability of a User to Run a Script

As mentioned earlier, by default, double-clicking a .vbs file will immediately run the script. However, if you modify the registry on the host computer, you can prevent the script from running immediately; instead, a warning box that contains a customized message will display.

This does not prevent users from running the script; they can right-click the file, click Open With, locate Wscript.exe or Cscript.exe, and then run the script. Alternatively, users can start the script from the command line by specifying the script host. However, this approach does provide an extra layer of protection by giving users the option to cancel a script before it runs.

Disable WSH

When circumstances call for a more radical approach to securing your organization against malevolent scripts, you can disable WSH. This action prevents users from running any scripts (including those written in the Visual Basic, Scripting Edition and JScript languages) that rely on WSH. You can disable WSH by modifying the registry on the host computer.

Protecting Logon Scripts

- **Script Encoder**
 - Basic script protection
 - Seamless to the user
 - Suffix must be .vbe or .jse
- **Script Encoder Syntax**
- **Examples**

You may want to hide the contents of your scripts from users to protect sensitive data such as passwords, and to prevent users from changing the contents of the file.

Script Encoder

The Script Encoder (Sce10en.exe) is a command-line tool that protects the contents of a script from unauthorized viewing, copying, or modification, while still enabling the script to run. The script file passes to the Script Encoder, which then performs an encoding routine against the file and creates a new file with a .vbe (or .jse for JScript) extension.

This file, if viewed in a text editor, contains what appear to be random command characters. Changing any part of the file renders it inoperable. This ensures the integrity of the encoded script, while enabling it to be decoded and executed by WSH.

Important: This encoding only protects the script from a casual attempt at viewing and modifying script. It does not protect the script from a determined attempt to crack the encoding mechanism.

The Script Encoder is available from the Microsoft Download Center Web site.

It is important to note that the Script Encoder was written primarily for the encoding of Web-based scripts such as Active Server Pages (ASP pages). As a result, many of the switches and examples given as part of the documentation must not be used for encoding WSH files.

Script Encoder Syntax

The following example shows the syntax for the Script Encoder.

```
screnc [/?] [/s] <source> <destination>
```

The following table describes the arguments.

Argument	Description
/?	Help.
/s	Silent. Display no messages.
Source	The file to encode. It can include wildcard characters.
Destination	The destination file.

Note: This table is a simplified list of the syntax for Sce10en.exe and shows the switches that are important for the encoding of standard WSH files. For a more detailed explanation of all of the switches, see the Script Encoder documentation.

Examples

The following are some examples of how you can use the Script Encoder.

This command encodes Demo.vbs into EncodedDemo.vbe.

```
screnc Demo.vbs EncodedDemo.vbe
```

This command encodes Demo.vbs into c:\testscripts\EncodedDemo.vbe, without generating any messages.

```
screnc Demo.vbs /s c:\testscripts\EncodedDemo.vbe
```

Lesson 4: Troubleshooting and Best Practices

- **Troubleshooting Logon Scripts**
- **Best Practices for Creating Logon Scripts**

As in any software development, knowing common troubleshooting techniques and implementing best practices can help you to become a proficient logon script writer. This lesson introduces you to techniques and best practices that can save you time when writing your logon scripts.

Objectives

After completing this lesson, you will be able to:

- Describe how to use the File Replication Service (FRS) to replicate logon scripts across your domain.
- Set time-outs so that logon script performance does not affect the logon experience of your users.
- Describe how network bandwidth issues on the server and the client computer can affect a user's logon experience.
- Perform tests that help you to ensure that files and other required resources for your scripts exist on the computer on which you run the logon script.
- Describe the best practices to use when creating logon scripts.

Troubleshooting Logon Scripts

- **Replicating Logon Scripts**
- **Setting Time-Outs**
- **Network Bandwidth**
 - At server and client
- **Testing for Required Files**

Troubleshooting your logon scripts on the networks and client computers on which they will run is a key step in creating logon scripts. There are several techniques that you can use to ensure that your scripts do not degrade the logon experience for your users.

Replicating Logon Scripts

On Windows Server 2003, you can use the FRS to replicate logon scripts between domain controllers in your enterprise domains. You must simply make sure that you save the GPOs and define the logon scripts that you want to use across the domain. These GPOs and scripts are stored in the Windows Server 2003 System Volume (SYSVOL). The SYSVOL provides a standard location to store important elements of GPOs and scripts so that the FRS can distribute them to other domain controllers in a particular domain.

Setting Time-Outs

Setting time-outs can help to prevent a script that has stopped responding from degrading the logon experience for users.

You can set the script time-out by changing either the settings for the script host or the setting in a .wsh file. However, you can also set a time-out by using the Script Parameters option when you assign the startup/shutdown or logon/logoff scripts in Group Policy. This setting accepts any of the standard command-line switches for the script engines. For example, the following switch sets the maximum run time for the script to 30 seconds.

//T:30

Network Bandwidth

There are two areas to consider when using scripts over a network:

- Server bottlenecks

Server bottlenecks may be the result of multiple simultaneous logon attempts at peak times. Domain controllers are responsible for authenticating users and accepting incoming connections from client computers to retrieve computer startup scripts and user logon scripts. Servicing all of these requests may have a negative impact upon the performance of domain controllers.

- Connectivity bottlenecks

The speed of physical links such as Wide Area Network (WAN) links for remote sites, or RAS links for mobile users, is a factor when assigning scripts to users and computers. Before any of the scripts run, they must be copied over the link to the local computer. This can significantly increase the time it takes for a script to run.

To avoid these bottlenecks, consider the following:

- Ensure that there are enough domain controllers to authenticate users. By using Microsoft DNS records, script requests from client computers running Windows XP and Windows Vista can be balanced across multiple domain controllers. The authenticating domain controller for each user has a copy of the scripts that are assigned to that user in the Netlogon share.
- Distribute the processing across multiple member servers. The user's logon script in the Netlogon share of the domain controller can be a pointer to the main script. The client computer can then connect to a share on a member server to retrieve this script.

Testing for Required Files

It is always a good practice to ensure that scripts run without generating unnecessary errors. As a result, a script must test for the possibility of an error. If possible, a script must correct the error, or at least exit gracefully with a helpful message. A logon script must check that the files required for the script to run successfully are present and that the files are the required version.

Best Practices for Creating Logon Scripts

- **Test All Changes to Production Logon Scripts**
- **Implement Error Handling**
- **Document Scripts**
- **Be Aware of Network Bandwidth Issues**
- **Protect Scripts with Encoding**
- **Create a “Script Fix” and “Test User” Logon**

Best practices outline particular strategies that you can use when you create logon scripts. This topic outlines several techniques that you can use to make your logon scripts work as you intend them before you deploy them in a production environment.

Test All Changes to Production Logon Scripts

Do not make any changes to a production logon script without first testing the script thoroughly in all environments in which the script is likely to run. Consider using a version control system like Microsoft Visual SourceSafe® to monitor versions of your scripts. By doing so, you can retrieve older versions of scripts if the changes implemented are problematic.

Implement Error Handling

Error handling is as essential for scripts as it is for compiled programs. If you fail to test for errors, your script may not complete all of the functions that you intended.

Document Scripts

Comments help script writers and administrators to understand a script written by someone else, or even a script that they wrote themselves a few months previously. Comments make maintaining scripts easier.

Be Aware of Network Bandwidth Issues

Do not forget about network bandwidth issues. Assess whether scripts have unnecessary functionalities, for example, they copy items over the network that can be stored on the local computer.

Protect Scripts with Encoding

If you do not want users to view the contents of your logon scripts, consider encoding them with the Windows Script Encoder.

Create a “Script Fix” and “Test User” Logon

Having a user account specifically for fixing problems with logon scripts can be a benefit if something goes wrong with a script. Make sure that no scripts are assigned to this user. Then, you can log on anywhere without any scripts running.

It may also be useful to create a test user for the logon script. This test user has the first account to have a new or modified logon script assigned to it. This account must have the same level of permissions as the accounts for the users to whom the script will be assigned.

Lab B: Assigning Logon Scripts



• **Exercise 1**
Configuring Scripts by Using Group Policy

After completing this lab, you will be able to:

- Explain the client-side extensions available in Windows Vista and how to use them to run a script.
- Assign a script to a client-side extension.

Estimated time to complete this lab: 45 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine first and then start the 2433B-VISTA-CL1-06 virtual machine.
- Log on to each virtual machine as **FOURTHCOFFEE\Administrator** with a password of **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You want to ensure that all of the computers and users in the Fourth Coffee domain have the appropriate settings. Therefore, you must write startup, logon, logoff, and shutdown scripts. You want to use Group Policy settings to achieve this.

Exercise 1

Configuring Scripts by Using Group Policy

In this exercise, you will write logon scripts and manage them by using Group Policy settings. You will create scripts for each of the client-side extensions: startup, shutdown, logon, and logoff. You will assign these scripts by using Group Policy.

The principal tasks for this exercise are as follows:

- To create the script folder.
- To create the scripts.
- To open the Group Policy Microsoft Management Console (MMC).
- To assign the startup script.
- To assign the shutdown script.
- To assign the logon script.
- To assign the logoff script.
- To save the Microsoft Management Console.
- To test the scripts for the client-side extensions.

Task	Supporting information
1. To create the script folder.	<ul style="list-style-type: none"> • Using Windows Explorer, create a LogonScripts folder on drive C of the client virtual machine.
2. To create the scripts.	<ul style="list-style-type: none"> • Open PrimalScript. • Using the ScriptFiles template list, create a new VBScript file. • Create a simple script using a message box or other easy to code interface, and save the file to the LogonScripts folder. Name the file Startup.vbs. • Perform the above step three more times, naming the files Logon.vbs, Logoff.vbs, and Shutdown.vbs. <div style="border: 1px solid black; padding: 5px; background-color: #e0f2ff; margin-top: 10px;"> <p>Note: You may want to perform a more complex task for startup and shutdown scripts because the computer monitor is not available during these stages to display a user interface from a script.</p> </div>
3. To open the Group Policy Microsoft Management Console.	<ul style="list-style-type: none"> • Open the Group Policy MMC console. On Windows Vista, it is easiest to open the MMC by using the command prompt. • After you have opened the MMC, you must use the Add/Remove Snap-in command and add the Group Policy Object Editor.
4. To assign the startup script.	<ul style="list-style-type: none"> • Use the tree pane to expand the Local Computer Policy node, the Computer Configuration node, the Windows Settings node, and the Scripts node. • Assign Startup.vbs to the Startup client-side extension.

Task	Supporting information
5. To assign the shutdown script.	<ul style="list-style-type: none">Assign Shutdown.vbs to the Shutdown client-side extension.
6. To assign the logon script.	<ul style="list-style-type: none">Use the tree pane to expand the User Configuration node, the Windows Settings node, and the Scripts node.Assign Logon.vbs to the Logon client-side extension.
7. To assign the logoff script.	<ul style="list-style-type: none">Assign Logoff.vbs to the Logoff client-side extension.
8. To save the Microsoft Management Console.	<ul style="list-style-type: none">Save the Group Policy MMC.
9. To test the scripts for the client-side extensions.	<ul style="list-style-type: none">Restart the virtual machine to test the scripts for the client-side extensions.

Questions

Q: What are the four client-side extensions that are provided by Windows Server 2003?

Q: What can you assign a logon script to?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

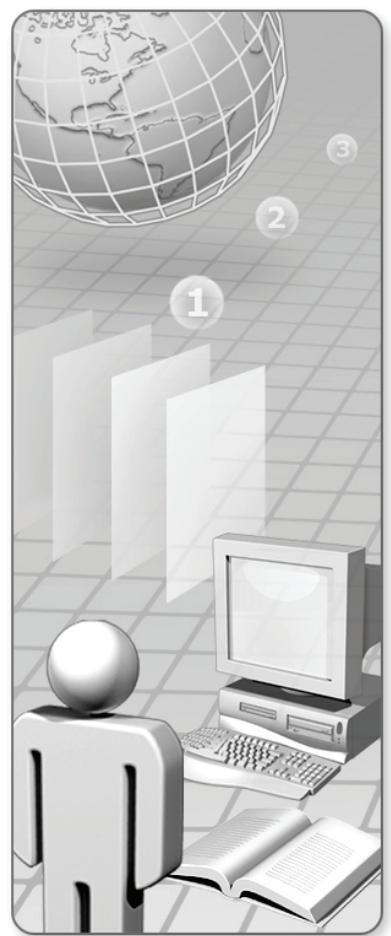
After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-06 virtual machine and discard any changes.

Important: If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

Module 7: Administrative Scripts

Table of Contents

Module Overview	7-1
Lesson 1: Administrative Scripts	7-2
Lesson 2: Producing Event Logs and E-Mail	
Messages	7-10
Lab A: Administrative Scripts	7-16
Lesson 3: Managing the Registry	7-21
Lesson 4: Controlling Drives, Folders, and Files	7-26
Lab B: File and E-mail Scripts	7-39



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Module Overview

- **Administrative Scripts**
- **Producing Event Logs and E-Mail Messages**
- **Managing the Registry**
- **Controlling Drives, Folders, and Files**

Using script to perform administrative functions offers many advantages. This module describes how to use script to perform typical administrative tasks by using the various scripting models that are available through Windows® Script Host (WSH) and Microsoft® Visual Basic®, Scripting Edition (VBScript).

Objectives

After completing this module, you will be able to:

- Manage arguments and use scheduling in scripts.
- Add entries to Windows Event Logs and use Collaboration Data Objects (CDO) to generate e-mail messages.
- Use script to manage the registry.
- Control drives, folders, and files by using script.

Lesson 1: Administrative Scripts

- **Script Arguments**
- **Scheduling Scripts**
- **Best Practices**

This lesson introduces several important principles for use with administrative scripts. Arguments supply parameters to scripts at run time, so that the same script can be used in various ways. The Task Scheduler service enables you to set a script to run at pre-determined times. This lesson also describes best practices to use with administrative scripts.

Objectives

After completing this lesson, you will be able to:

- Use arguments to pass run-time parameters to scripts.
- Use the Task Scheduler with scripts.
- Apply best practices when writing administrative scripts.

Script Arguments

The diagram illustrates a window-like interface containing script code and descriptive text. At the top left, there's a bulleted list: '• WScript Object' and '• Arguments collection'. Below this is a code block with a light green background, enclosed in a thin black border. The code is:

```
For i = 0 to WScript.Arguments.Count - 1
    MsgBox "Argument " & i+1 & " is "
        & WScript.Arguments.Item(i)
```

Below the code, the word 'Next' is aligned to the right. To the right of the code block, another bulleted list is shown under the heading '• Drag-and-Drop Operation':

- File name passed as an argument
- If script does not check for arguments, they are ignored

As with most command-line tools, you can send arguments to a script. This feature is especially useful for administrative scripts. To avoid changing your script files every time you require them to perform a series of slightly different tasks, you can develop the scripts so that they accept arguments. You can then supply these arguments at run time to modify the tasks that are performed.

WScript Object

The **WScript** object has an **Arguments** collection that gives your code access to all of the values that are passed to the script as arguments. The following script example shows how you can loop through the **Arguments** collection and process each argument individually.

```
For i = 0 to WScript.Arguments.Count - 1
    MsgBox "Argument " & i+1 & " is "
        & WScript.Arguments.Item(i)
Next
```

When you invoke a script, you can provide multiple arguments by separating them with spaces. If you want to pass values that contain spaces to the script, you can enclose the values in double quotes (""). The following is an example of a command line that provides two arguments to a script. Note that one of the arguments contains spaces.

```
Cscript.exe Args.vbs kathief@fourthcoffee.com "Kathie Flood"
```

Drag-and-Drop Operation

If you drag a file onto a script, the full path and file name are passed to that script as an argument. If the script has no argument-handling routine, the arguments are ignored, and the script runs as if no arguments were passed to it.

Scheduling Scripts

- **Task Scheduler Scripts**
 - Windows Vista only
- **SchTasks.exe**
 - Windows Vista, Windows Server 2003, and Windows XP
- **Scheduling Service**
 - Scheduled Tasks Wizard
- **Error Log If Task Fails**
 - Windows Vista – Task Scheduler Operational log
 - SchedLgU.txt

Scripting certain tasks eases the administrative load on an organization. You can run some of these tasks, such as running and controlling maintenance or tool programs, after normal office hours. If you script a process, you do not have to be in the office to run the script. Instead, you can schedule scripts to be executed at appropriate times.

One advantage of using a scheduled script to control an application, rather than scheduling the application executable itself to run, is that you can capture errors and design your script to respond or report those errors. Another advantage of using scheduled scripts is that you can specify a second application to run immediately after the first application has closed.

Task Scheduler Scripts

In the Windows Vista® operating system, you can access the Task Scheduler application programming interface (API) through scripts. This method only works for Task Scheduler 2.0, which is currently only available on Windows Vista.

The following example uses the **Schedule.Service** object; the script examines the root Task Scheduler folder and lists all of the tasks in that folder.

```
Set service = CreateObject("Schedule.Service")
service.Connect

Set rootFolder = service.GetFolder("\")
Set colTasks = rootFolder.GetTasks(0)

For Each task In colTasks
    Wscript.Echo task.Name
Next
```

Note: For more information about scripting the Task Scheduler in Windows Vista, see “Scripting in Windows Vista, Task Scheduler: Part 1” on the Microsoft TechNet Web site.

SchTasks.exe

Windows Vista, Windows Server® 2003, and Windows® XP operating systems include SchTasks.exe, a command-line tool for adding and removing tasks from the schedule; starting and stopping tasks on demand; and displaying and changing scheduled tasks.

Note: In Microsoft Windows® 2000, you must use the AT command-line tool (AT Scheduler) to schedule the running of scripts from the command line. This tool is still available in Windows Vista, Windows Server 2003, and Windows XP but provides less functionality than SchTasks.exe.

Scheduling Service

In Windows Vista, you use the Task Scheduler tool, in the Accessories\System Tools folder, to create and modify tasks. In Windows Server 2003, Windows XP, and Windows 2000, you use the Scheduled Tasks folder in Control Panel, which includes a wizard that helps you to add any application or script file to a schedule. In all cases, you can set the schedule for daily, weekly, monthly, and other frequencies. The wizard also requires a user name and a password that have sufficient security permissions to run the program or execute the script. In addition to these basic features, you can set advanced properties to control the schedule, including the ability to run the application or script only when the computer is in an idle state.

Tip: In Windows Server 2003, Windows XP, and Windows 2000, scheduled tasks are stored as job files in the Windows\Tasks folder. You can copy these tasks in the same way as any file, so you can create your job on one computer, and then copy the file as part of a logon script to make distribution easier.

Error Log If Task Fails

If an error occurs during a scheduled task, the scheduling service will record the error. The location of the error log varies, depending on the Windows version.

Windows version	Error log
Windows Vista	Task Scheduler errors are recorded in the Task Scheduler Operational log, and can be viewed using Event Viewer.
Windows Server 2003	Task Scheduler errors are appended to %systemroot%\tasks\SchedLgU.txt
Windows XP	Task Scheduler errors are appended to %systemroot%\SchedLgU.txt
Windows 2000	Task Scheduler errors are appended to %systemroot%\SchedLgU.txt

Note: In Windows Vista, the %systemroot%\tasks\SchedLgU.txt file still exists, but it only records events that are related to the Task Scheduler service itself. This log does not record errors associated with particular scheduled jobs.

MCT USE ONLY. STUDENT USE PROHIBITED

Best Practices

- **Build Error Handling into the Scripts**
- **Test All Administrative Scripts Extensively**
- **Use the Microsoft Scripting Center**

Consider the following practices when using administrative scripts.

Build Error Handling into the Scripts

Administrative scripts must include built-in error handling. It is very important that you are aware of any script failures.

If a running script encounters an error, the default behavior is to halt script execution at the line of code that contains the problem. Depending on the script, an error message may also appear on screen. During testing it is important to see all errors, so the default behavior is usually appropriate. However, there are occasions when more sophisticated error handling is more useful. The following script uses the **On Error Resume Next** mechanism and script functions, so that script execution continues after any error, and a specific error-handling routine reports the cause of the error.

```
Call Main()

Sub Main()
    On Error Resume Next
    MsgBox 1/0
    If Err.Number <> 0 Then
        Call ErrReport(Err)
        Exit Sub
    End If
    MsgBox "No error"
End Sub

Sub ErrReport(oError)
    MsgBox "Error!" & VbCrLf & "Error number: " & oError.Number _
        & VbCrLf & "Error description: " & oError.Description
End Sub
```

The first **MsgBox** statement attempts to report the result of 1 divided by 0. This generates an error (**Err.Number**) value that is not zero, which the **ErrReport** function reports to the screen, providing the following error message: “Error! Error number: 11. Error description: Division by zero.”

Caution: You must be careful when using **On Error Resume Next** in production scripts. For example, you must not use **On Error Resume Next** in logon scripts because your users will not necessarily be aware of any problem with the script unless errors are reported to screen.

Test All Administrative Scripts Extensively

Always test administrative scripts in a test environment before releasing them in a production system. Try testing scripts in various scenarios to ensure that they will function correctly.

Use the Microsoft Scripting Center

The Microsoft Scripting Center provides a wide range of sample administrative scripts that you can use as the basis of your own scripts. For more information, see the Microsoft Scripting Center Web site at www.microsoft.com/technet/scriptcenter/default.mspx.

Lesson 2: Producing Event Logs and E-Mail Messages

- Event Logs
- E-Mail Messages

For many scripts, it is important that you can supply feedback or reports that can be recorded as Windows events or as e-mail messages. This lesson describes how to use scripting to produce event logs and e-mail messages.

Objectives

After completing this lesson, you will be able to:

- Use scripting to write to Windows Event Logs, and read information from existing logs.
- Use scripting to generate e-mail messages.

Event Logs

• Writing to the Windows Application Event Log

```
Set oShell = CreateObject("WScript.Shell")
oShell.LogEvent 4, "Login Script Succeeded"
```

- SUCCESS 0
- ERROR 1
- WARNING 2
- INFORMATION 4
- AUDIT_SUCCESS 8
- AUDIT_FAILURE 16

• Reading from Event Logs

When a script is being executed, it can log events to the Windows Vista, Windows Server 2003, Windows XP, and Windows 2000 Application Event Logs.

Writing to the Windows Application Event Log

You can use the **LogEvent** method of the **Shell** object to log events. The source of the log message is shown in Event Viewer as “WSH.” The following example shows the syntax for this command.

```
object.LogEvent (intType, strMessage [,strTarget])
```

The **intType** argument is an integer value that represents the type of event to log.

You can choose from the integer values in the following table.

IntType argument	Meaning
0	SUCCESS
1	ERROR
2	WARNING
4	INFORMATION
8	AUDIT_SUCCESS
16	AUDIT_FAILURE

Note that **strMessage** is the text entry of the event, and **strTarget** is the target system where the event must be logged. The default target is the local system.

The following example shows the correct code to log an event.

```
Dim oShell  
  
Set oShell = CreateObject("WScript.Shell")  
  
oShell.LogEvent 4, "Logon Script Succeeded"
```

Reading from Event Logs

You cannot use the WSH objects to read information from system event logs. However, you can read this information by using the following alternative solutions:

- Use the WevtUtil.exe tool for Windows Vista, or the EventQuery.vbs script for Windows Server 2003 and Windows XP. Both of these tools are part of the operating system and are located in the %SystemRoot%\System32 directory. For Windows 2000, you can use the Dumpel.exe tool from the Windows 2000 Resource Kit. You can use all of these tools to create a tab-separated text file that you can manipulate by using the **FileSystemObject** object.
- Use the Windows Management Instrumentation (WMI) object model. WMI has an event-log provider that provides access to data and event notifications from all of the Windows event logs. For more information about WMI, see Module 8, “WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.
- Use a third-party Component Object Model (COM) component that provides read/write permission for the system event logs.

E-Mail Messages

- Generating E-Mail Messages

- CDO Object Model

- Sending an E-Mail Message

```
Set oMessage = CreateObject("CDO.Message")
oMessage.To      = "helpdesk@fourthcoffee.com"
oMessage.Subject = "Error Log from test"
oMessage.TextBody = "The error log is attached."
oMessage.AddAttachment "C:\ErrorLog.txt"
oMessage.Send
```

- Configuration Changes

- Inherits either IIS or default identity of Windows Mail, Office Outlook, or Outlook Express

You can use the CDO libraries to programmatically access messaging functionality. These libraries are especially useful for tasks such as sending administrative alerts by e-mail.

Generating E-Mail Messages

The CDO libraries for Windows 2000 and later versions are implemented using Cdosys.dll. In order to provide e-mail functionality, the library must reside on a computer that has either local or network access to a Simple Mail Transfer Protocol (SMTP) or Network News Transfer Protocol (NNTP) service.

Note: Version 6.x of Cdosys.dll is included with Windows Vista, Windows Server 2003, Windows XP, and Windows 2000. Different CDO libraries are included with Microsoft Exchange Server 5.5, Exchange Server 2000, and Exchange Server 2003. Exchange Server 2007 does not include a CDO library.

Sending an E-Mail Message

The CDO library **Message** object provides the functionality that enables you to send e-mail messages as part of your scripts.

The following example shows how easy it is to add messaging features to your scripts.

```
Set oMessage = CreateObject("CDO.Message")
oMessage.To      = "helpdesk@fourthcoffee.com"
oMessage.Subject = "Error Log from test"
oMessage.TextBody = "The error log is attached."
oMessage.AddAttachment "C:\ErrorLog.txt"
oMessage.MIMEFormatted = False
oMessage.Send
```

```
WScript.Echo "Message Sent From : " & oMessage.From  
Wscript.Echo "To    : " & oMessage.To
```

The properties and methods of the **Message** object are self-explanatory, such as **To** (comma separators for multiple addresses), **Subject**, and **From**. The **AddAttachment** method can be called multiple times to add several attachments to a message. If **From** is not assigned, this method will attempt to load the default value from any current mail profile.

Configuration Changes

You can also configure many settings, such as language, time zone, proxy, and values for timing out when a connection is not made, by using the **CDO.Configuration** object.

If you do not change these settings, default values are loaded from one of two places. One of these is Internet Information Services (IIS). The other is the default Messaging Application Programming Interface (MAPI) profile that is used by your mail client application (such as Microsoft Office Outlook® messaging and collaboration client). Therefore, it is advisable to use configuration objects if it is possible that the default configuration settings may be inappropriate, or if there is no local profile. If a local SMTP or NNTP service is found on the computer, the default submittal method for messages is through the associated pickup directory.

The following example shows how to send a message using custom configuration settings.

```
Set oMessage = CreateObject("CDO.Message")  
oMessage.Subject      = "Test message from CDO"  
oMessage.Sender       = "administrator@fourthcoffee.com"  
oMessage.To           = "adrianlannin@fourthcoffee.com"  
oMessage.TextBody     = "Hi Adrian - checking server settings."  
  
'==This section provides the configuration information for the remote SMTP server.  
oMessage.Configuration.Fields.Item _  
("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2  
  
'Name or IP of Remote SMTP Server  
oMessage.Configuration.Fields.Item _  
("http://schemas.microsoft.com/cdo/configuration/smtpserver") =  
"london.fourthcoffee.com"  
  
'Server port (typically 25)  
oMessage.Configuration.Fields.Item _  
("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 25  
  
oMessage.Configuration.Fields.Update  
  
oMessage.Send
```

In this example, the send method (SMTP), the SMTP server name, and the server port are all configured in the message itself. Any e-mail configuration settings associated with the user's mail profile are ignored.

For more information about configuration, see “Configuring the Message Object” on the MSDN Web site.

MCT USE ONLY. STUDENT USE PROHIBITED

Lab A: Administrative Scripts



- **Exercise 1
Passing Arguments to Scripts**
- **Exercise 2
Writing an Event to the Application Event Log**

After completing this lab, you will be able to:

- Use the drag-and-drop operation to pass arguments to a script.
- Write an entry to the Windows Server 2003 Application Event Log.

Estimated time to complete this lab: 30 minutes

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine.
- Log on to the 2433B-LON-DC1 virtual machine as **Administrator** using the password **Pa\$\$w0rd**
- Start the 2433B-VISTA-CL1-07 virtual machine.
- Log on to the 2433B-VISTA-CL1-07 virtual machine as **FOURTHCOFFEE\Administrator** using the password **Pa\$\$w0rd**

Lab Scenario

You are the administrator of the Fourth Coffee corporate network. You are responsible for writing scripts, and you want to make your scripts as reusable as possible. One way to achieve this is to enable arguments to be passed to the scripts when they are run. You also want to use event logs, so that you can easily find out whether scheduled scripts ran successfully or generated errors.

Exercise 1

Passing Arguments to Scripts

In this exercise, you will create a script that accepts a file name as an argument and displays a message box that contains the script's full path and file name.

The principal tasks for this exercise are as follows:

- To create a script that accepts file names as arguments.
- To run the script and use file names as arguments.

Task	Supporting information
1. To create a script that accepts file names as arguments.	<ul style="list-style-type: none"> • On the 2433B-VISTA-CL1-07 virtual machine, start PrimalScript. • Using the Script Files template list, create a new VBScript file. • Edit the template to declare the Option Explicit command and the following variables: <ul style="list-style-type: none"> • vArg • aArgs() • iCount • Use the Snippets Browser to insert an If...Else block and a For...Next loop (labeled “ForTo” in the Snippets Browser). The code should resemble the following example. <pre>If WScript.Arguments.Count = 0 Then MsgBox "No Arguments Supplied" WScript.Quit Else WScript.Echo "Argument count is: " & → WScript.Arguments.Count ReDim aArgs(WScript.Arguments.Count - 1) For iCount = 0 to WScript.Arguments.Count - 1 aArgs(iCount) = WScript.Arguments(iCount) MsgBox aArgs(iCount) Next End If</pre> • Save the script as E:\Labfiles\Starter\VBArgs.vbs.
2. To run the script and use file names as arguments.	<ul style="list-style-type: none"> • Use Windows Explorer to open the E:\Labfiles\Starter folder. • Drag Text1.txt onto VBArgs.vbs. • Note the messages. • Select Text1.txt, Text2.txt, and Text3.txt and drag these files onto VBArgs.vbs. • Note the messages. • Double-click the VBArgs.vbs script to run it directly without any arguments. • Note the messages.

Note: The line continuation character (→) indicates that the text following it should be written on the same line as the code that precedes it.

Questions

Q: Why would you use the **On Error Resume Next** mechanism in a script?

Q: If an error occurs during a scheduled task, where does the scheduling service record the error?

Exercise 2

Writing an Event to the Application Event Log

In this exercise, you will add a new section to the previous script. The new section will write a new entry to the Windows Server 2003 Application Event Log to indicate which files were passed as arguments.

The principal tasks for this exercise are:

- To write events to the Application Event Log.
- To review the events in the Application Event Log.

Task	Supporting information
1. To write events to the Application Event Log.	<ul style="list-style-type: none"> • Switch to PrimalScript. • Use the Snippets Browser to add the following lines of code to the end of the VBArgs.vbs script. <pre>Dim oShell, sLogEntry Set oShell = CreateObject("WScript.Shell") sLogEntry = "Script was run with: " For iCount = 0 to UBound(aArgs) sLogEntry = sLogEntry & vbCrLf & aArgs(iCount) Next oShell.LogEvent 4, sLogEntry</pre> <ul style="list-style-type: none"> • Save the script as E:\Labfiles\Starter\VBArgsEvents.vbs. • Switch to Windows Explorer. • Select Text1.txt, Text2.txt, and Text3.txt and drag these files onto VBArgsEvents.vbs. • Note the messages.
2. To review the events in the Application Event Log.	<ul style="list-style-type: none"> • Run Event Viewer. • Open the Windows Application Log. • Review the Application Log entry at the top of the list. • Close Event Viewer. • Close PrimalScript.

Questions

Q: To which event log can you write by using the **LogEvent** method of the **Shell** object?

Q: What is WevtUtil?

Note: The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

Lab Shutdown

After you complete the lab, do not shut down the virtual machines. You will use them again for the second lab in this module.

Lesson 3: Managing the Registry

- **Accessing the Registry**
- **Modifying the Registry**

The Windows registry holds information about computer configurations and user settings, and it is often useful to access and modify this information using scripts. This lesson describes how to use scripts to list information in the registry, and how to modify and delete existing values and keys.

Objectives

After completing this lesson, you will be able to:

- Use scripting to list information from the registry.
- Use scripting to write to the registry and delete registry keys and values.

Accessing the Registry

• Reading from the Registry

```
sValue = oShell.RegRead ("HKLM\System\Current...")
```

• Example

```
Set oShell = CreateObject("WScript.Shell")
MsgBox oShell.RegRead("HKLM\Software\Microsoft\" & _
"Windows NT\CurrentVersion\CurrentVersion")
```

The registry is one of the most important storage locations in the Windows environment. Administrators often require access to information stored in the registry and must manipulate key configuration settings to reconfigure systems. You can accomplish this by using script. The **WScript.Shell** object enables access to the registry to read, write, and delete keys and values. The following table lists the abbreviations you use to refer to the registry subtrees when you are writing scripts to access the registry.

Abbreviation	Subtree
HKLM	HKEY_LOCAL_MACHINE
HKCU	HKEY_CURRENT_USER
HKU	HKEY_USERS
HKCC	HKEY_CURRENT_CONFIGURATION
HKCR	HKEY_CLASSES_ROOT

Reading from the Registry

To read data in the registry, you can create a **WScript.Shell** object and use the **RegRead** method, which returns the Windows version number. The following example shows this.

```
Set oShell = CreateObject("WScript.Shell")
MsgBox oShell.RegRead("HKLM\Software\Microsoft\" & _
"Windows NT\CurrentVersion\CurrentVersion")
```

This method returns various data types, depending on the key value that is set in the registry. The **RegRead** method works with the majority of data types that are strings, integers, or binaries. However, this method may cause problems where other types of value are returned.

Tip: Use the VBScript **Join** function to read from a **REG_MULTI_SZ** type of registry entry.

For more information about the **Join** function, see the documentation about the Visual Basic, Scripting Edition language.

If your script attempts to read a key that does not exist, an error occurs. You can trap the error by using the **On Error Resume Next** statement and checking for an **Err.Number** value that is not zero. In this way, you can verify the existence of a key by attempting to read from it.

The following script demonstrates this approach. The script is testing for the existence of a registry key for the Microsoft SQL Server™ database software.

```
Dim oShell, sSQLServ
On Error Resume Next
Set oShell = WScript.CreateObject("WScript.Shell")
sSQLServ = oShell.RegRead(_
    "HKLM\SOFTWARE\Microsoft\MSSQLServer\MSSQLServer\" _ 
    & "CurrentVersion\CurrentVersion")
If Err.Number = 0 Then
    WScript.Echo "SQL Server Version is: " & sSQLServ
Else
    WScript.Echo "SQL Server Version not found"
End If
```

Modifying the Registry

• Writing to the Registry

```
oShell.RegWrite "HKLM\.. \Current...", "New value"
```

• Example

```
Dim oShell  
Set oShell = WScript.CreateObject("WScript.Shell")  
oShell.RegWrite "HKLM\Software\NewKey\", "My new key"
```

• Deleting from the Registry

```
oShell.RegDelete "HKLM\System\Current..."
```

• Backing Up the Registry Before Testing Scripts

Writing to the registry is more restrictive than reading registry values. You can only write strings, integers, and binary values to the registry by using script. You cannot write **REG_MULTI_SZ** (array) values by using script.

You can use the **RegWrite** method to write to the registry. In the following example, a new string value will be created in a registry key.

```
Dim oShell  
Set oShell = WScript.CreateObject("WScript.Shell")  
oShell.RegWrite "HKLM\Software\NewKey\", "My new key"  
oShell.RegWrite "HKLM\Software\NewKey\NewValue", "My new value"
```

If the last character of the key name is a backslash (\), a new key is created instead of a value. A third optional string argument tells the function what type of value to write: **REG_SZ** (string), **REG_DWORD** (integer), or **REG_BINARY** (binary). If no value type is specified, the decision is made by analyzing the second argument to see whether it is a string or an integer.

Deleting from the Registry

You can easily delete a key or a value from the registry by using the **RegDelete** function, as the following example shows.

```
oShell.RegDelete "HKLM\Software\NewKey\NewValue"      'Delete value "NewValue"  
oShell.RegDelete "HKLM\Software\NewKey\"                'Delete key "NewKey"
```

MCT USE ONLY. STUDENT USE PROHIBITED

Backing Up the Registry Before Testing Scripts

If you are writing scripts to modify the registry of any systems, ensure that you have backups that you can restore if the script damages the integrity of the system.

Caution: Use extreme care when deleting or modifying the registry by using a script. Modifying or deleting certain values or keys may make the operating system unstable or even unusable.

Lesson 4: Controlling Drives, Folders, and Files

- Accessing Drives, Folders, and Files
- Creating and Deleting Folders
- Text Files
- Setting Folder-Level and File-Level Security

Scripting gives you a high level of control over drives, files, and folders. WSH exposes a variety of methods that you can use to manage file system resources. You can use these methods to create, delete, write, and extract information from local and remote drives, files, and folders. You can also record this information for later reference.

This lesson will describe how to manage file system resources by using script.

Objectives

After completing this lesson, you will be able to:

- Use the **FileSystemObject** object to access drives, folders, and files.
- Create and delete folders.
- Use files to read information from text files, and write new information.
- Apply file and folder security.

Accessing Drives, Folders, and Files

- **FileSystemObject**
- ```
Set oFS = CreateObject("Scripting.FileSystemObject")
```
- **Drive Access**
    - **Drives** collection or **GetDrive** method of FileSystemObject
  - **Folder Access**
    - **GetFolder** method of FSO
    - **RootFolder** property of a drive
    - **SubFolders** collection of folder
  - **File Access**
    - **GetFile** method of FSO
    - **Files** collection of folder

The Scripting Object Model (Scrrun.dll) provides useful objects that you can use to manage drives, folders, and files. The main file object is the **FileSystemObject**. All other objects are created or retrieved from **FileSystemObject**.

### FileSystemObject

By using **FileSystemObject**, you can create, copy, delete, and retrieve information from folders, files, and drives. You must create this object before any other objects in the Scripting Object Model can be instantiated or used. You use the following line of script to create an instance of the **FileSystemObject** object.

```
Set oFSO = CreateObject("Scripting.FileSystemObject")
```

### Drive Access

You can access disk drives through the **Drives** collection property of the **FileSystemObject** object or by calling the **GetDrive** method. The following example shows how to loop through available drives and display some of the drive properties. Note the use of the **IsReady** property to check the availability of a drive. This is useful when you access removable media such as floppy disks. Also note the use of the **With** statement, which enables you to perform a series of actions on a specified object without having to retype the name of the object for each action.

```
Dim oFS, oDrive, sOutput
Set oFS = CreateObject("Scripting.FileSystemObject")
For Each oDrive In oFS.Drives
 With oDrive
 If .IsReady Then
 sOutput = "Drive:" & .DriveLetter & vbCrLf
 sOutput = sOutput & " Type: " & .FileSystem _
```

```
& vbCrLf
sOutput = sOutput & " Serial: " & .SerialNumber _
& vbCrLf
sOutput = sOutput & " Label: " & .VolumeName _
& vbCrLf
sOutput = sOutput & " Size: " & _
Round((.TotalSize / 1073741824), 2) & "GB" _
& vbCrLf
sOutput = SOutput & " Avail: " & _
Round((.AvailableSpace / 1073741824), 2) & "GB" _
& vbCrLf
WScript.Echo sOutput
else
WScript.Echo "Drive: " & .DriveLetter & " not ready"
End If
End With
```

Next

## Folder Access

You can access folders by using the **GetFolder** method of the **FileSystemObject** object or by using the **RootFolder** property of the **Drive** object to gain access to the **SubFolders** collection.

The following example loops through the **SubFolders** collection of drive C.

```
Dim oFS, oDrive, oFolder
Set oFS = CreateObject("Scripting.FileSystemObject")
Set oDrive = oFS.GetDrive("C")
For Each oFolder In oDrive.RootFolder.SubFolders
 WScript.Echo oFolder.name
Next
```

It is possible to loop through all of the child folders of a particular item, because each folder also has the **SubFolders** collection property.

## File Access

You can use the **Files** collection of the **Folder** object to access file objects on the hard disk drive. File properties, including **Name**, **Path**, **Size**, and **Type**, are available to your script. The following example loops through the **Files** collection of a folder and displays each file name and type.

```
Dim oFS, oDrive, oFolder
Set oFS = CreateObject("Scripting.FileSystemObject")
Set oDrive = oFS.GetDrive("C")
For Each oFile In oFolder.Files
 Wscript.Echo " " & oFile.Name & " " & oFile.Type
Next
```

**Caution:** Be careful when using script to manipulate the file system because it is easy to delete every file and folder from a hard disk drive when using these powerful objects and methods. Be sure to test your scripts thoroughly before releasing them in a production environment.

MCT USE ONLY. STUDENT USE PROHIBITED

## Creating and Deleting Folders

- **Creating Folders**  
`oFS.CreateFolder "C:\Temp\New Folder"`
  - Parent folder must exist (C:\Temp)
- **Deleting Folders**  
`oFS.DeleteFolder "C:\Temp\New Folder", True`
- **Sharing Folders**
  - Using ADSI
  - LanManServer Object

---

The **FileSystemObject** object provides methods that you can use to create or delete folders.

### Creating Folders

To create a folder, call the **CreateFolder** method of the **FileSystemObject** object, as the following example shows.

```
Dim oFS
Set oFS = CreateObject("Scripting.FileSystemObject")
oFS.CreateFolder FolderName
```

When you create a folder by using this object, the parent of the folder must exist before the new child folder is created, otherwise an error will occur.

### Deleting Folders

You can use two methods to delete a folder. One is the **DeleteFolder** method of the **FileSystemObject** object. The syntax for this method is shown in the following example.

```
Dim oFS
Set oFS = CreateObject("Scripting.FileSystemObject")
oFS.DeleteFolder folderSpec [, force]
```

The first argument, *folderSpec*, is the folder name and can include wildcard characters. The optional second argument, **force**, is a Boolean value that forces folder deletion, even on read-only folders. The default value of this argument is **False**. The **DeleteFolder** method also deletes any files or subfolders in the chosen folder.

Alternatively, you can delete a folder by calling the **Delete** method of the **Folder** object itself.

**Caution:** These methods delete the folders and the files that they contain completely. The folders and files are not sent to the Recycle Bin.

Be careful when using **FileSystemObject** for deletions. **FileSystemObject** can remove every file and folder from a hard disk drive without a single prompt, if instructed to do so. Make sure that your script gives adequate warning if it is about to delete files or folders from a user's hard disk drive, and incorporate a routine to enable the user to cancel the script if it was run in error.

### Sharing Folders

You cannot share a folder by using the Microsoft Scripting Runtime library. To do this, you must use the **LanmanServer** object in Microsoft Active Directory® Service Interfaces (ADSI). For more information about ADSI and the **LanmanServer** object, see Module 5, "ADSI," in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

## Text Files

- **Creating a Text File**  
`Set oFile = oFS.OpenTextFile("c:\Users.txt", 1)`
- **Reading from a Text File**
  - OpenTextStream
  - Read, ReadLine, ReadAll
- **Writing to a Text File**
  - CreateTextFile
  - OpenTextFile
  - Write, WriteLine, WriteBlankLines

---

The **FileSystemObject** library enables you to create, read, and write to text files. If you want to create, read, or write to other file types such as Microsoft Office documents, you must use the appropriate library to provide this functionality. The Microsoft Office System exposes objects and methods to enable you to do this.

### Creating a Text File

The main way to create a file is by using the **CreateTextFile** method of the **FileSystemObject** or **Folder** objects. This method creates a new file and returns the **TextStream** object from the file, enabling you to write to the file. The syntax for creating a file is shown in the following example.

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

The optional second argument is a Boolean value that indicates whether you want to overwrite an existing file. The default value for this argument is **False**. The optional third argument specifies the type of file to create, which is either ASCII (the default) or Unicode.

The following example demonstrates how to create the file C:\Users.txt, and write the string “Jo Berry” to the file.

```
Dim oFS, oFile, aFileName, aString

aFileName = "C:\Users.txt"
aString = "Jo Berry"

Set oFS = CreateObject("Scripting.FileSystemObject")
Set oFile = oFS.CreateTextFile(aFileName)
oFile.WriteLine aString
oFile.Close
```

## Reading from a Text File

You can use the following methods of the **TextStream** object to read data from a text file.

| Method   | Task                                               |
|----------|----------------------------------------------------|
| Read     | Read a specified number of characters from a file. |
| ReadLine | Read an entire line.                               |
| ReadAll  | Read the entire contents of a text file.           |

By using the **Skip** and **SkipLine** methods, you can select the data that the script will read. The text string returned by these methods can be displayed, parsed by string functions (such as **Left**, **Right**, and **Mid**), concatenated with other data, and so on. The following example, which is written in the Visual Basic, Scripting Edition language, demonstrates how to open a file and then read the first line from it.

```
Dim oFS, oFile, strLine

Set oFS = CreateObject("Scripting.FileSystemObject")
Set oFile = oFS.OpenTextFile("c:\Users.txt", 1)
strLine = oFile.ReadLine
MsgBox strLine
```

## Writing to a Text File

The **TextStream** object provides write access to a text file. You can open a **TextStream** object by using the **CreateTextFile** method of the **FileSystemObject** or **Folder** object, the **OpenTextStream** method of the **FileSystemObject** object, or the **OpenAsTextStream** method of a **File** object.

The following example shows the **OpenTextFile** syntax.

```
object.OpenTextFile(filename[, iomode[, create[, format]]])
```

The **filename** argument is the path and file name of the file to be opened.

By using the **iomode** switch, you can specify whether the file is to be opened as read-only, write-only, or appended to. You cannot open a file with the intention of reading some lines and then writing others; you must open the file as read-only, then close the file, and then reopen it as append or write. The following table shows the constant values that are used to specify the mode in which the method opens the file.

### OpenTextFile Constants

| Constant     | Value | Description                                                   |
|--------------|-------|---------------------------------------------------------------|
| ForReading   | 1     | Open a file for reading only. You cannot write to this file.  |
| ForWriting   | 2     | Open a file for writing only. You cannot read from this file. |
| ForAppending | 8     | Open a file and write to the end of the file.                 |

By using the **Create** argument, you can ensure that the method creates a new file if the requested file does not exist. You must set the value to **True** if you are creating a new file or to **False** if this is not required. The default for this value is **False**.

The **Format** argument enables you to specify that the file must be opened as ASCII, Unicode, or as the system default. You do this by using the values shown in the following table.

### Format Constants

| Constant           | Value | Description                                 |
|--------------------|-------|---------------------------------------------|
| TristateUseDefault | -2    | Opens the file by using the system default. |
| TristateTrue       | -1    | Opens the file as Unicode.                  |
| TristateFalse      | 0     | Opens the file as ASCII.                    |

The following example demonstrates how to append the string “Success” to the file C:\Logs\ErrorLog.txt.

```
Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim oFS, oFile, aFileName, aString

aFileName = "C:\Logs\ErrorLog.txt"
aString = "Success"

Set oFS = CreateObject("Scripting.FileSystemObject")
Set oFile = oFS.OpenTextFile(aFileName, ForAppending, True)
oFile.WriteLine aString
oFile.Close
```

## Setting Folder-Level and File-Level Security

- **ADSI**
  - Windows Vista, Windows Server 2003, Windows XP  
ADsSecurityUtility object—built-in
  - Windows 2000  
ADsSecurity.dll—part of ADSI SDK
- **WMI**
- **Xcacls**
  - Xcacls.vbs—Windows 2000 and later
  - Xcacls.exe—Windows Server 2003, Windows XP, Windows 2000

---

**FileSystemObject** does not have any methods for setting folder-level or file-level security permissions. If you want to set or change permissions by using script, you can use the ADSI interface, WMI, or the Extended Change access control list (Xcacls) tool.

### ADSI

Before you can use this interface to set security permissions, you must have COM extensions to enable ADSI to access NTFS file system (NTFS) permissions. These extensions provide a consistent security interface for file permissions. Windows Vista, Windows Server 2003, and Windows XP include the **ADsSecurityUtility** object for this purpose.

**Note:** Windows 2000 requires a resource kit dynamic-link library (DLL). ADsSecurity.dll is part of the ADSI 2.5 Software Development Kit (SDK). You can download it from the Microsoft TechNet Web site.

Each computer that attempts to set permissions by using these extensions must have them installed, or the script will fail. The process is very similar to the process of setting permissions on objects in Active Directory. For more information about setting permissions on objects in Active Directory, see Module 5, “ADSI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

### WMI

You can change security permissions through WMI by using the **ChangeSecurityPermissions** method of the **Win32\_Directory** class. For more information about WMI, see Module 8, “WMI,” in Course 2433, *Microsoft Visual Basic, Scripting Edition and Microsoft Windows Script Host Essentials*.

## Xcacls

An alternative method of setting security permissions is to use the Xcacls tool. Xcacls is available in an executable version (Xcacls.exe) and a version written in the Visual Basic, Scripting Edition language (Xcacls.vbs). The following table lists the versions available on each Windows version.

| Windows version     | Xcacls.exe                        | Xcacls.vbs               |
|---------------------|-----------------------------------|--------------------------|
| Windows Vista       | (Not available)                   | <a href="#">Download</a> |
| Windows Server 2003 | Windows Server 2003 support tools | <a href="#">Download</a> |
| Windows XP          | Windows XP support tools          | <a href="#">Download</a> |
| Windows 2000        | Windows 2000 Resource Kit         | <a href="#">Download</a> |

You can download the latest version of Xcacls.vbs from the Microsoft Downloads Web site.

**Note:** You must modify the current version of Xcacls.vbs to support Windows Vista. The line **Case "5.0", "5.1", "5.2"** must be replaced by **Case "5.0", "5.1", "5.2", "6.0"**.

Xcacls.vbs enables you to set all Microsoft Windows Explorer file-system security options from the command line. The following example uses Xcacls.vbs from the command line to assign the Modify permission to AdrianLannin for the C:\Logs directory.

```
cscript xcacls.vbs C:\Logs /E /G \\FOURTHCOFFEE\AdrianLannin:M
```

The CScript host is used in this example, so that error and status messages will be displayed at the command line.

The following code achieves the same result as the preceding example, but Xcacls.vbs has been called from within another script.

```
Dim oShell, strCommand

Set oShell = CreateObject("WScript.Shell")
strCommand = "C:\Logs /E /G \\FOURTHCOFFEE\AdrianLannin:M"
oShell.Run "cscript \\london.fourthcoffee.com\tools\xcacls.vbs" & strCommand
```

In this script, an instance of the **WScript.Shell** object is created, and its **Run** method is used to execute Xcacls.vbs from a server share. This approach requires much less code than using ADSI to assign file and folder permissions.

## Parameters

Running Xcacls.vbs /? displays detailed help for the tool. The following shows the Xcacls.vbs syntax.

```
XCACLS filename [/E] [/G user:perm;spec] [...] [/R user [...]]
[/F] [/S] [/T]
[/P user:perm;spec [...]] [/D user:perm;spec] [...]
[/O user] [/I ENABLE/COPY/REMOVE] [/N]
```

[*/L filename*] [*/Q*] [*/DEBUG*]

You can use wildcard characters, multiple users in a single command, and the combination of multiple access rights in Xcacls.vbs. The following table describes some of these parameters.

| Parameter                  | Description                                                                                                                                                                                                                   |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Filename</i>            | [Required] If used alone, it displays ACLs.<br>Filename can be a file name, directory name, or wildcard characters, and can include the entire path. If the path is missing, it is assumed to be under the current directory. |
| <i>/T</i>                  | Causes Xcacls to work recursively through the current directory and all of its subdirectories.                                                                                                                                |
| <i>/E</i>                  | Edits the ACL instead of replacing it. The default is to replace the ACL.                                                                                                                                                     |
| <i>/G user:permissions</i> | Grants security permissions. See below for more details.                                                                                                                                                                      |
| <i>/R user</i>             | Revokes all access rights for the specified user.                                                                                                                                                                             |
| <i>/P user:perm;spec</i>   | Replaces access rights for the user. The rules for specifying perm and spec are the same as for the /G option. Some examples are given below in this documentation.                                                           |
| <i>/D user</i>             | Denies user access to the file or directory.                                                                                                                                                                                  |
| <i>/O user</i>             | Changes the ownership to this user or group.                                                                                                                                                                                  |

**Note:** For more information about Xcacls.vbs parameters, see “How to use Xcacls.vbs to modify NTFS permissions” on the Microsoft Help and Support Web site.

## /G in Detail

The */G* switch is the most complex part of the Xcacls tool. This switch typically takes the following parameters:

- User. This is the name of the account on which the permissions are being set. Acceptable name formats are:
  - UserName or DomainName\UserName
  - GroupName or DomainName\GroupName
- Permissions. The permissions option is used to assign rights, and can be any of the following standard values:
  - F (Full control)
  - M (Modify)
  - X (read and eXecute)
  - L (List folder contents)
  - R (Read)

- W (Write)

You can also specify advanced permissions such as:

- D (Take Ownership)
- C (Change Permissions)
- B (Read Permissions)
- A (Delete)

## Examples

This command replaces the access control list (ACL) of all files and folders found in the current folder without scanning any subfolders.

```
CScript Xcacls.vbs *.* /G administrator:RW
```

This command edits the ACL of all files and folders found in the current folder without scanning any subfolders, and adds List folder contents to the existing ACLs.

```
CScript Xcacls.vbs *.* /G \\FOURTHCOFFEE\JoBerry:L /E
```

This command revokes all permissions for a folder.

```
CScript Xcacls.vbs C:\Logs /R \\FOURTHCOFFEE\JoBerry
```

## Lab B: File and E-mail Scripts



The icon depicts a computer monitor displaying a file folder, with several blue cylinders representing hard drives connected by dashed lines. A small figure of a person is sitting at the desk, and an open book is on the surface.

- **Exercise 1**  
**Documenting Your Computer Drives and Folders**
- **Exercise 2**  
**Sending a Message and Attachment**

---

After completing this lab, you will be able to:

- Use the **FileSystemObject** object to create and write to files, and extract information about the folders on each of the drives on your computer.
- Use the **CDO.Message** to send a text file by e-mail.

Estimated time to complete this lab: 30 minutes

### Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Make sure that the 2433B-LON-DC1 virtual machine is still running, and that you are still logged on as **Administrator** after Lab A.
- Make sure that the 2433B-VISTA-CL1-07 virtual machine is still running, and that you are still logged on as **FOURTHCOFFEE\Administrator** after Lab A.

### Lab Scenario

You are the administrator of the Fourth Coffee corporate network, and it is often necessary for you to audit the contents of your users' computers. This can be a time-consuming and tedious task. You have decided that scripting the audit and recording the contents of the computer drives in a text file would suit your requirements. In this lab, you will write a script that documents all of the folders on all of the drives of a computer. The script will record the details in a text file.

You also want to collect every audit report and send it to the helpdesk. You have decided to use CDO to automatically generate the status e-mail messages with the audit reports as attachments.

## Exercise 1

### Documenting Your Computer Drives and Folders

In this exercise, you will use the **FileSystemObject** object to create and write to a file that contains information about the folders and hard disk drives on your computer.

The principal tasks for this exercise are as follows:

- To create a script that documents the drives on your computer.
- To run the DriveReport script.

| Task                                                              | Supporting information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. To create a script that documents the drives on your computer. | <ul style="list-style-type: none"> <li>• On the 2433B-VISTA-CL1-07 virtual machine, start PrimalScript.</li> <li>• Using the <b>Script Files</b> template list, create a new VBScript file.</li> <li>• Edit the template to declare the following variables:           <ul style="list-style-type: none"> <li>• oFS</li> <li>• oDrive</li> <li>• oFileText</li> <li>• sOutPut</li> </ul> </li> <li>• Insert E:\Labfiles\Starter\CreateTextFile.snippet.</li> <li>• Edit the inserted snippet to resemble the following code.</li> </ul> <pre>Set oFS = CreateObject("Scripting.FileSystemObject") ' Replace "Drive:\Path\Filename.extension" with your own data Set oFileText =   oFS.CreateTextFile("E:\Labfiles\Starter\DriveData.txt")</pre> <ul style="list-style-type: none"> <li>• Use the Snippets Browser to insert a <b>For...Each</b> loop.</li> <li>• Edit the <b>For...Each</b> loop statement to resemble the following code.</li> </ul> <pre>For Each oDrive In oFS.Drives     With oDrive         sOutPut = "Drive: " &amp; .DriveLetter         sOutPut = sOutPut &amp; " Type: " &amp;         GetDriveString(.DriveType)         If .IsReady Then             sOutPut = sOutPut &amp; " Format: "         &amp; .FileSystem         sOutput = sOutput &amp; " Label: "         &amp; .VolumeName         If .DriveType &lt;&gt; 3 Then             oFileText.WriteLine sOutPut             WriteFolder oDrive.RootFolder,         " "         End If     Else         oFileText.WriteLine sOutPut &amp; " not     ready"     End If     End With Next WScript.Echo "All Done!!" oFileText.Close WScript.Quit</pre> |

|                                   |                                                                                                                                                                                                                                            |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                   | <ul style="list-style-type: none"><li>• Insert E:\Labfiles\Starter\GetDriveStringFunction.snippet</li><li>• Insert E:\Labfiles\Starter\WriteFolderSub.snippet.</li><li>• Save the script as E:\Labfiles\Starter\DriveReport.vbs.</li></ul> |
| 2. To run the DriveReport script. | <ul style="list-style-type: none"><li>• Run the script in PrimalScript.</li><li>• Use Windows Explorer to open E:\Labfiles\Starter\DriveData.txt.</li><li>• Review the file contents, and then close the DriveData.txt file.</li></ul>     |

**Note:** The line continuation character (→) indicates that the text following it should be written on the same line as the code that precedes it.

## Questions

Q: Which object model exposes **FileSystemObject**?

Q: Why must you be careful when using scripts to manipulate the file system?

## Exercise 2

### Sending a Message and Attachment

In this exercise, you will add a new section to the previous script that sends a message to the helpdesk, with the text file as an e-mail attachment.

The principal tasks for this exercise are:

- To create the SendReport script.
- To run the DriveReport script.

| Task                                | Supporting information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. To create the SendReport script. | <ul style="list-style-type: none"> <li>• Switch to PrimalScript.</li> <li>• Using the <b>Script Files</b> template list, create a new VBScript file.</li> <li>• Edit the template to declare the variable <code>oMessage</code>.</li> <li>• Create the CDO message object, using the following code.</li> </ul> <pre>Set oMessage = CreateObject("CDO.Message")</pre> <ul style="list-style-type: none"> <li>• Create the following message details: <ul style="list-style-type: none"> <li>• <code>oMessage.Subject</code> set to <b>Drive Report</b></li> <li>• <code>oMessage.Sender</code> set to <b>adrianlannin@fourthcoffee.com</b></li> <li>• <code>oMessage.To</code> set to <b>administrator@fourthcoffee.com</b></li> <li>• <code>oMessage.TextBody</code> set to <b>The drive report is attached</b></li> </ul> </li> <li>• Create the e-mail attachment, using the following code.</li> </ul> <pre>oMessage.AddAttachment "E:\Labfiles\Starter\DriveData.txt"</pre> <ul style="list-style-type: none"> <li>• Add <code>oMessage.Send</code>, to send the message.</li> <li>• Use <b>WScript.Echo</b> to add a status message to the end of the script.</li> <li>• Save the script as E:\Labfiles\Starter\SendReport.vbs.</li> </ul> |
| 2. Run the DriveReport script.      | <ul style="list-style-type: none"> <li>• Run the script in PrimalScript.</li> <li>• Switch to the 2433B-LON-DC1 virtual machine.</li> <li>• Run Outlook Express.</li> <li>• In Outlook Express, click <b>Send/Receive</b>.</li> <li>• Verify that you have received an e-mail titled “Drive Report.”</li> <li>• Open the e-mail message and the attachment, and verify that the attachment contains the drive report.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

### Questions

**Q:** If no configuration information is set in a script when sending an e-mail message by means of CDO, where does the script get this information?

**Q:** What function in Visual Basic, Scripting Edition can you use to help manage **REG\_MULTI\_SZ** values in the registry?

**Note:** The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

## Lab Shutdown

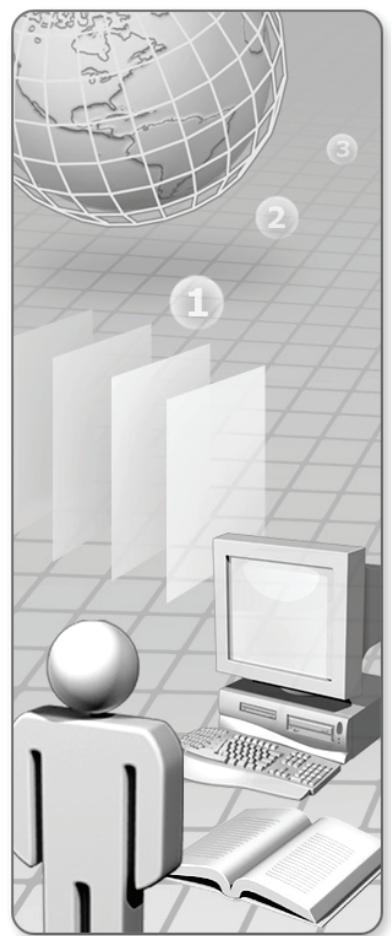
After you complete the lab, you must shut down the 2433B-LON-DC1, and 2433B-VISTA-CL1-07 virtual machines and discard any changes.

**Important:** If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

## Module 8: WMI

**Table of Contents**

|                                                              |      |
|--------------------------------------------------------------|------|
| Module Overview                                              | 8-1  |
| Lesson 1: WMI Overview                                       | 8-2  |
| Lesson 2: WMI Scripting Basics                               | 8-8  |
| Lesson 3: Common Issues in WMI Scripts                       | 8-17 |
| Lesson 4: Scripting Common Administrative Tasks by Using WMI | 8-29 |
| Lab: Writing WMI Scripts                                     | 8-38 |
| Course Evaluation                                            | 8-43 |



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Module Overview

- **WMI Overview**
- **WMI Scripting Basics**
- **Common Issues in WMI Scripts**
- **Scripting Common Administrative Tasks by Using WMI**

As a system administrator, it is vital to have a consistent and uniform platform on which to manage, control, and monitor the systems across your enterprise. Windows® Management Instrumentation (WMI) provides such a platform for Windows operating systems. You can use WMI to query, change, and monitor configuration settings on desktop and server systems, applications, networks, and other enterprise components. You can write scripts that use the WMI Scripting library to work with WMI and create a wide range of systems management and monitoring scripts.

### Objectives

After completing this module, you will be able to:

- Describe how WMI works.
- Describe the three basic steps that most WMI scripts use.
- Understand what a WMI provider is.
- Know how to use the PrimalScript WMI Wizard to create WMI scripts.
- Use WMI scripts to perform common management tasks.
- Use WMI providers to manage Windows-based services.

## Lesson 1: WMI Overview

- **What Is WMI?**
- **Benefits of WMI**
- **WMI Architecture**

---

Before you write code that interacts with WMI, you must understand how WMI works and how it can help you to create administrative scripts.

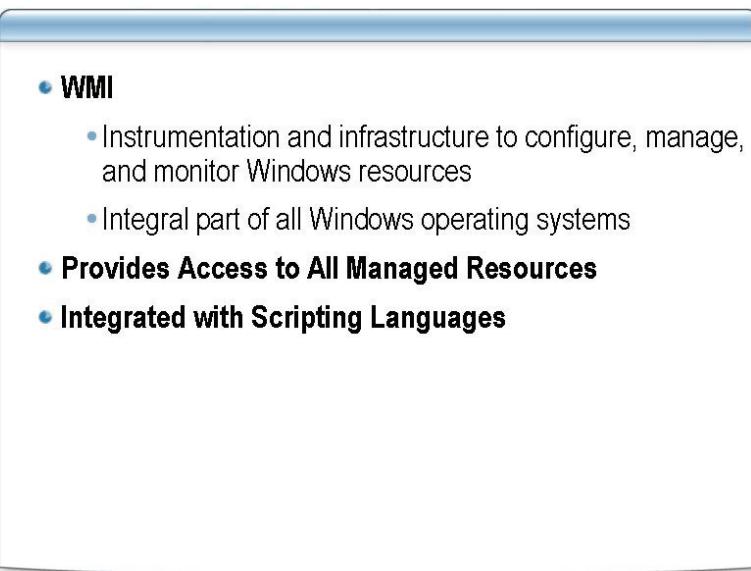
This lesson describes the uses of WMI and the benefits of using WMI in your administrative scripts. In addition, this lesson describes the architecture and security of WMI.

### **Objectives**

After completing this lesson, you will be able to:

- Describe WMI.
- Describe the benefits of WMI.
- Describe the architecture of WMI.

## What Is WMI?



---

WMI is the instrumentation and infrastructure through which you can access, configure, manage, and monitor most Windows-based resources.

### **WMI**

WMI was originally released in 1998 as an add-on component with Windows NT® 4.0 Service Pack 4. WMI is now an integral part of all Windows operating systems, including Windows® XP, Windows Vista®, and Windows Server® 2003.

### **Provides Access to All Managed Resources**

WMI provides a consistent model of the managed system environment. For each managed resource, there is a corresponding WMI class. You can think of a WMI class as a succinct description of the properties of a managed resource and the actions that WMI can perform to manage that resource. A managed resource is any object—for example, computer hardware, computer software, or a user account—that you can manage by using WMI.

### **Integrated with Scripting Languages**

You can write scripts that access WMI classes by using Windows Script Host (WSH), Microsoft® Visual Basic®, Scripting Edition (VBScript), Microsoft JScript®, or any scripting language that supports Automation (for example, ActivePerl from ActiveState Corporation).

**Note:** You can also use WMI with the Windows PowerShell™ command-line interface. For more information about Windows PowerShell, see the Windows PowerShell Technology Center Web site.

## Benefits of WMI

- **Single Management Technology**
- **Scripting Support Across the Enterprise**
- **Thorough Management Capabilities**

---

WMI offers numerous benefits to system and network administrators.

### **Single Management Technology**

WMI offers a single scripting interface to manage any resource that supports Automation. This means that, instead of learning how to use numerous administrative tools, it is only necessary to learn how to write scripts that use the WMI Scripting library. You can then use the same approach to manage resources as disparate as disk drives, event logs, and installed software.

### **Scripting Support Across the Enterprise**

You can write scripts that use WMI to configure, manage, and monitor nearly all aspects of your enterprise. These include administrative tasks associated with the following:

- Computers running Microsoft Windows® 2000, Windows XP, Windows Server 2003, and Windows Vista
- Networks
- Real-time health monitoring
- Windows Server system applications

A resource must include a WMI provider for you to write WMI scripts for that resource.

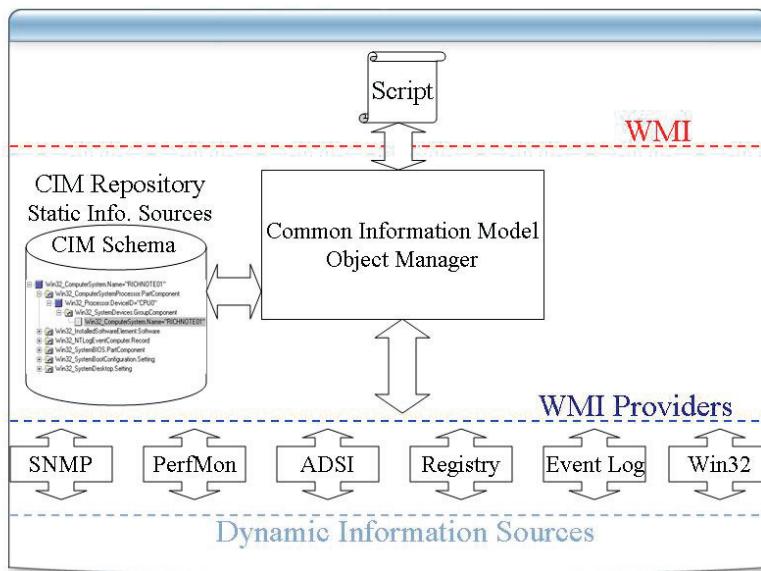
### **Thorough Management Capabilities**

In some cases, the capabilities found in WMI replicate capabilities found in command-line tools or graphical user interface (GUI) applications. In other cases, however, WMI provides management capabilities not readily available anywhere else. For example, by using WMI you can easily write a script to retrieve the total amount of physical memory

MCT USE ONLY. STUDENT USE PROHIBITED

installed in a remote Windows-based computer. Before WMI, there was no way to script this task without using a third-party tool. In fact, before WMI, the only operating system tool that enabled you to determine the amount of memory installed in a computer was the **System Properties** dialog box. Although this approach works for manually retrieving the memory configuration on the local computer, you cannot use it to automatically retrieve the memory configuration, or to obtain memory information from a remote computer.

## WMI Architecture



WMI consists of several components that work together to enable WMI client applications such as management programs or scripts to query, recover, and change information from hardware, software, or the operating system.

The basic components of WMI are:

- Common Information Model Object Manager (CIMOM)

CIMOM manages any calls that a WMI client application makes to WMI. It determines whether the request involves static data that is stored in the Common Information Model (CIM) repository or dynamic data supplied by a provider. These WMI providers specialize in providing specific features to WMI for a specific data source such as the Windows registry. When a provider finishes processing a request, it returns the results to CIMOM. CIMOM forwards these results to the WMI client application.

- CIM repository

The CIM repository is a central storage area that is managed by CIMOM. It stores the class definitions for static classes and stores static instances. If the data is regularly updated or dynamic, the WMI providers manage the updates.

- WMI providers

WMI providers are Component Object Model (COM) objects that are interpreters for WMI. WMI providers enable WMI to communicate with specific objects in a manageable and scalable manner. The following providers are available as part of Windows XP, Windows Vista, and Windows Server 2003:

- Win32

- Registry
- Directory services (Microsoft Active Directory®)
- Event Viewer logs
- Performance Monitor (PerfMon)
- Simple Network Management Protocol (SNMP)
- Internet Information Services (IIS)

Additionally, most Microsoft applications include WMI providers.

WMI is an extensible interface. If a third-party developer wants to expose management information about his or her program, the developer can write a WMI provider and publish information about the provider's capabilities to WMI. After the provider is published, WMI client applications can access this management information.

## Lesson 2: WMI Scripting Basics

- **Common Steps in WMI Scripts**
- **WMI Providers**

---

WMI interfaces increase the functionality of your administrative scripts.

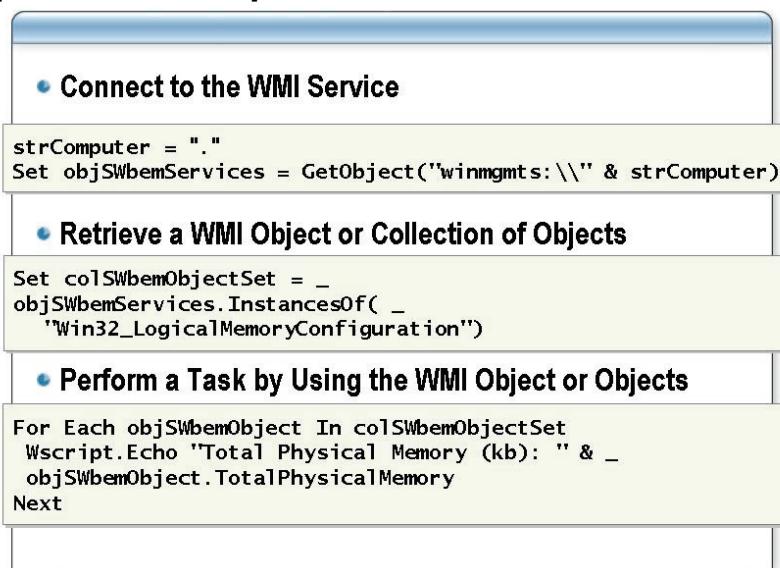
This lesson describes how to use WMI in your scripts.

### Objectives

After completing this lesson, you will be able to:

- Describe the three basic steps common to most WMI scripts.
- Understand how to write a WMI script that accesses a WMI provider.

## Common Steps in WMI Scripts




---

Most WMI scripts consist of three basic steps:

1. Connecting to the WMI service.
2. Retrieving a WMI object or a collection of WMI objects.
3. Performing a task by using the WMI object or collection.

### Connect to the WMI Service

There are two steps to connecting to the WMI service. The first is to define the computer on which you are connecting. The typical technique is to define a strComputer variable.

```
strComputer = "computername"
```

To define the local computer, write the following line of code.

```
strComputer = "."
```

To define a remote computer in the local computer's domain, use the remote computer's name.

```
strComputer = "alt-dc-01"
```

After you set the strComputer variable to the name of the computer that you want to manage, you must write script to connect to the WMI service on that computer. There are two techniques that you can use.

## Using the SWbemLocator Class

The first technique is to create an instance of the **SWbemLocator** class. After object creation, call the **ConnectServer** method to log on to the WMI namespace that contains the object or objects you want to administer.

The following example logs on to the **root\cimv2** namespace on the current server.

```
strComputer = "."
Set objLocator = CreateObject("WbemScripting.SWbemLocator")
Set objService = objLocator.ConnectServer(strComputer, "root\cimv2")
```

If necessary, you can define any security settings or impersonation levels required by the object to which you are connecting by calling the **SWbemServices.Security\_** method.

## Using WMI Monikers

The second technique is to get the WMI object you want to work with by using a moniker string. This technique is far more common in most WMI scripts because it only requires one line of code.

WMI monikers consist of up to three parts: one mandatory component and two optional components. The mandatory component is the "Winmgmts:" prefix. All WMI monikers must begin with "Winmgmts:" as shown in the following example.

```
Set objSWbemServices = GetObject("winmgmts:")
```

The moniker in this code is the string "Winmgmts:", which is passed to the **GetObject** function. Specifying a moniker that consists only of the "Winmgmts:" prefix is the most basic form of WMI moniker you can use. The result is always a reference to a **SWbemServices** object, which represents a connection to the WMI service on the local computer.

The second part of WMI monikers is the optional security settings component. You can specify various security settings on this component, which connect to and communicate with WMI. The security settings that you can control as part of the moniker string include:

- Impersonation level, as shown in the following example.

```
"winmgmtsw:{impersonationLevel=Value}"
```

- Authentication level, as shown in the following example.

```
"winmgmts:{authenticationLevel=Value}"
```

- Authenticating authority, as shown in the following examples.

```
"winmgmts:{authority=ntlmDomain:DomainName}"
```

or

```
"winmgmts:{authority=kerberos:DomainName\ServerName}"
```

- Privileges to grant or deny, as shown in the following examples.

```
"winmgmts:{(Security, !RemoteShutdown)}"
```

The following example shows how to use the security settings component.

```
Set objSWbemServices = GetObject _
 ("winmgmts:{impersonationLevel=impersonate,(LockMemory, !IncreaseQuota)}")
```

The third component of WMI monikers is the WMI object path. Like the security settings component, the object path component is optional. However, the object path provides a great deal of flexibility, including the ability to identify remote computers. You use the WMI object path to uniquely identify one or more of the following target resources:

- A remote computer. If you do not specify the computer in the object path, the script will connect to the WMI service on the local computer.
- A WMI namespace. If you do not specify this, the default namespace is used.
- WMI class in a namespace. If you specify a class in a moniker, you must separate the class from the computer name and the namespace by a colon.
- Specific instance or instances of a WMI class in a namespace.

For example, the following moniker binds directly to the **Win32\_OperatingSystem** class in the **root\cimv2** namespace on a computer named atl-dc-01.

```
Set objSWbemObject = GetObject _
 ("winmgmts:\\atl-dc-01\root\cimv2:Win32_OperatingSystem")
```

### Retrieve a WMI Object or Collection of Objects

There are two ways to retrieve WMI objects in your scripts. The first is to call the **InstancesOf** method of the **SWbemServices** object. The **InstancesOf** method returns all instances of the defined WMI. The following example returns a collection of all instances of the **Win32\_Service** class on a computer.

```
Set colServices = objSWbemObject.InstancesOf("Win32_Service")
```

Alternatively, you can create a WMI query using WMI Query Language (WQL). Using WQL (and the **ExecQuery** method) rather than **InstancesOf** provides you with the flexibility to create scripts that return only the items that are of interest to you. For example, you can use a basic WQL query to return all properties of all instances of a given class; this is the same information that is returned by the **InstancesOf** method. However, you can also create targeted queries using WQL. Targeted queries do such things as:

- Return only selected properties of all the instances of a class.
- Return all the properties of selected instances of a class.
- Return selected properties of selected instances of a class.

Creating targeted queries can noticeably increase the speed with which data is returned. Targeted queries also make it easier to work with the returned data. For example, suppose you want only events from the Application event log with **EventCode** 0. Using a targeted

query returns only those items, whereas **InstancesOf** returns all the events. This would then mean that you would have to individually examine each event and determine whether it 1) came from the Application event log, and 2) has **EventCode** 0. Although this can be done, it is less efficient and requires additional coding.

Targeted queries can also reduce the amount of data that is returned, which is an important consideration for scripts that run over the network.

The following example is a simple WQL query that returns the value of the **State** property from all **Win32\_Service** classes running on a computer.

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set colServices = objSWbemServices.ExecQuery("SELECT State FROM Win32_Service")
```

You can use clauses and keywords in WMI queries to limit the amount of information that the script returns.

**Note:** WQL queries have similar syntax to SQL queries, although WQL is much simpler and only provides a few clauses and keywords. Regardless of whether you are familiar with SQL, WQL syntax is very easy to learn.

You can retrieve selected instances of a class by using the **WHERE** clause. The following example shows the generic representation of such a query.

```
"SELECT * FROM ClassName WHERE PropertyName = PropertyValue"
```

The following query retrieves all instances of the **Win32\_Service** class that are stopped.

```
"SELECT * FROM Win32_Service WHERE State = 'Stopped'"
```

In this query, **Win32\_Service** is the class being queried, **State** is the class property, and Stopped is the value of the **State** property.

**Note:** Stopped is a string value, so it must be included in the single quote marks. If you use a Boolean value (**True** or **False**) in a query string, do not include single quote marks.

By using the **AND** and **OR** keywords, you can create more complex queries. These queries either narrow or expand the range of a query that uses a **WHERE** clause.

You can limit the scope of a query by using the **AND** keyword. For example, suppose you want a list of autostart services that are stopped. This query requires two parameters in the **WHERE** clause: the **State** property, which must be equal to Stopped, and the **StartMode**, which must be equal to Auto. To create a query such as this, include both parameters, separating them with the **AND** keyword. In this query, an instance will be returned only if it meets both criteria.

```
"SELECT * FROM Win32_Service WHERE State = 'Stopped' AND StartMode = 'Auto'"
```

By contrast, you can expand the scope of a query by using the **OR** keyword. For example, suppose you require a list of services that are either stopped or paused. In this case, there are two criteria, but a service must meet only one of these criteria to qualify.

To create an expanded query, use the **OR** keyword to separate the two parameters. This query returns all services that are either stopped or paused.

```
"SELECT * FROM Win32_Service WHERE State = 'Stopped' OR State = 'Paused'"
```

### Perform a Task by Using the WMI Object or Collection

After you have retrieved an object or collection of objects, you must decide what to do with the data. A typical choice is to write the data to a destination using the **WScript.Echo** method. The following example incorporates a query from the previous section to display the value of each object's **State** property at the command prompt.

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set colServices = objSWbemServices.ExecQuery("SELECT State FROM Win32_Service")
For Each objService In colServices
 Wscript.Echo objService.State
Next
```

You can also perform more complex tasks after you have retrieved data. The following script queries for a disk volume on the local computer that is assigned to drive K, and defragments the volume.

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" -
 & strComputer & "\root\cimv2")
Set colVolumes = objWMIService.ExecQuery _
 ("Select * from Win32_Volume Where Name = 'K:\\\'")
For Each objVolume in colVolumes
 errResult = objVolume.Defrag()
Next
```

## WMI Providers

- Active Directory
- BitLocker Drive Encryption
- Distributed File System
- Event Log
- Internet Information Services
- Performance Counter
- Policy
- Security
- WMIPerfClass
- WMIPerfInst

A WMI provider is a COM server that communicates with managed objects to access data and event notifications from a variety of sources such as the system registry or an SNMP device. Providers forward this information to the CIMOM for integration and interpretation. The CIMOM forwards the results to the WMI client application.

**Note:** WMI providers represent managed objects that are not included in the CIM repository. If software installed on a Windows operating system does not also install a WMI provider, the software resource is not available to WMI. However, this is not common.

All WMI providers send error and warning messages to the event log. You can write WMI scripts that access those messages.

The following table lists a subset of all WMI providers available on the Windows operating system. Microsoft products and third-party software also install WMI providers and their classes onto your computer.

| Provider                   | Description                                                                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Active Directory           | Maps Active Directory objects to WMI.                                                                                                                                                        |
| BitLocker Drive Encryption | Provides configuration and management for an area of storage on a hard disk drive, represented by an instance of <b>Win32_EncryptableVolume</b> , that can be protected by using encryption. |
| Distributed File System    | Supplies DFS functions that logically group shares on multiple servers, and links them transparently in a tree-like structure in a single namespace.                                         |

| Provider                      | Description                                                                                                                                              |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Event Log                     | Provides access to data from the event log service to notifications of events.                                                                           |
| Internet Information Services | Exposes programming interfaces that can be used to query and configure IIS.                                                                              |
| Performance Counter           | A high-performance provider that is the preferred source of raw performance data for Windows XP and Windows Server 2003.                                 |
| Policy                        | Provides extensions to group policy, and permits refinements in the application of policy.                                                               |
| Security                      | Retrieves or changes security settings that control ownership, auditing, and access rights to files, directories, and shares.                            |
| WMIPerfClass                  | Creates the WMI Performance Counter Classes in Windows Vista. Data is dynamically supplied to these WMI performance classes by the WMIPerfInst provider. |
| WMIPerfInst                   | Supplies raw and formatted performance counter data dynamically from WMI Performance Counter Class definitions in Windows Vista.                         |

**Note:** There are many other providers installed on Windows operating systems. For a complete list, see the WMI documentation on MSDN.

You can access the information offered by these providers using the **SWbemLocator** class or a WMI moniker.

The following example connects to the IIS provider and obtains all instances of the **WorkerProcess** class on the server. The script then displays the number of currently executing requests in each worker process on the server. The script notifies you if there are no currently executing requests in the worker process.

```

strComputer = "."
Set objSWbemServices = GetObject("winmgmts:\\" & strComputer & _
 "\root\WebAdministration")
Set objWorkerProcesses = oWebAdmin.InstancesOf("WorkerProcess")

For Each objWorkerProcess In objWorkerProcesses

 ' Place the requests queued for a process in an array variable.
 objWorkerProcess.GetExecutingRequests arrReqs

 ' Show the number of requests queued.
 If IsNull(arrReqs) Then
 WScript.Echo "No currently executing requests."
 Else
 ' Display the number of requests.
 End If
Next

```

```
WScript.Echo "Number of currently executing requests: " & _
 UBound(arrReqs) + 1

End If
Next
```

**Note:** For more examples of using some of the providers installed with Windows Vista, Windows XP, and Windows Server 2003, see Lesson 4 later in this module.

## Lesson 3: Common Issues in WMI Scripts

- Dates and Times
- Converting a UTC Date to Standard Date-Time Format
- Converting a Standard Date to UTC Date-Time Format
- Monitoring Resources by Using Event Notifications
- Differences Between Event Notification Queries and WMI Queries

WMI is a powerful and wide-ranging scripting technology. However, there are a few challenges and techniques that you must keep in mind when you create WMI scripts.

Firstly, WMI uses the Coordinated Universal Time (Greenwich Mean Time) date-time format, while the Windows operating system uses the Standard date-time format. When you use dates in WMI scripts, you must often convert them to and from Coordinated Universal Time (UTC) format.

Secondly, you can write WMI scripts that notify you when a computer suffers from poor performance, or when a computer uses more memory than an amount you specify. This approach is called event notification, and you can use scripts to monitor the health and performance of computers on your network.

Finally, you can use pre-written scripts as templates for your own scripts. There are a number of script templates on the Microsoft Windows 2000 Scripting Guide.

### Objectives

After completing this lesson, you will be able to:

- Describe some common challenges presented by WMI.
- Convert WMI dates and times to and from standard date-time format.
- Monitor resources with WMI scripts by using WMI event notifications.

## Dates and Times

### • WMI Uses UTC Date Format

- WMI will not convert to and from UTC automatically

---

One of the more confusing aspects of WMI is the way that WMI handles dates and times.

### **WMI Uses UTC Date Format**

When you write a script that results in a date and time, WMI returns that information in UTC format. WMI also does not recognize other date formats as input.

The following script returns the date when the operating system was installed on the local computer.

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set colOS = objSWbemServices.ExecQuery("SELECT * FROM Win32_OperatingSystem")
For Each objOS in colOS
 Wscript.Echo objOS.InstallDate
Next
```

This script then returns a value similar to the following example.

```
20061224113047.000000-480
```

This is the date, in UTC format, when the operating system was installed on the computer.

In UTC format, dates display as yyyyymmddHHMMSS.xxxxxx±UUU, where:

- yyyy represents the year. (Positions 1 through 4 in the UTC string.)
- mm represents the month. (Positions 5 and 6.)
- dd represents the day. (Positions 7 and 8.)
- HH represents the hour in 24-hour format. (Positions 9 and 10.)
- MM represents the minutes. (Positions 11 and 12.)
- SS represents the seconds. (Positions 13 and 14.)

- xxxxxx represents the milliseconds. (Positions 16 through 22.)
- UUU represents the difference, in minutes, between the local time zone and UTC. (Positions 24 through 26.)

Consequently, the value 20061224113047.000000-480 is translated like this:

- 2006 is the year.
- 12 is the month (December).
- 24 is the day.
- 11 is the hour of the day (in 24-hour format).
- 30 is the minutes.
- 47 is the seconds.
- 000000 is the milliseconds.
- 480 is the number of minutes less than UTC.

The UTC format creates several problems for both script writers and script users. Firstly, it is difficult to determine dates and times at a glance. Secondly, querying for items based on date-time values is not a straightforward process. Suppose you want to retrieve a list of all the folders on a computer that were created after September 3, 2005 and read them in standard format. You must convert the value returned from the **CreationDate** property in the **Win32\_Directory** class from a UTC date to a standard date.

## Converting a UTC Date to Standard Date-Time Format

### • Four Steps to Convert a UTC Date to a Standard Date Format

- Extract the month
- Extract the day
- Extract the year
- Combine them in a standard date format

---

To convert a UTC date to a standard date-time format, you select the date-time components (such as month, day, and year) that you want and construct a date-time string of your own.

### Four Steps to Convert a UTC Date to a Standard Date Format

The process of converting a UTC date to standard date-time format takes four steps. For example, with the UTC value 20060710113047.000000-420, you must:

- Extract the month (07).
- Extract the day (10).
- Extract the year (2006).
- Combine them in a standard date format: July 10, 2006. You can extract the individual date components using functions such as **Left** and **Mid** in Visual Basic, Scripting Edition. For example, to extract the day (Positions 7 and 8), you use code similar to that shown in the following example, in which **dtmInstallDate** is a variable used to represent the date being converted.

```
Mid(dtmInstallDate, 7 ,2)
```

The following script defines a function named **WMIDateStringToDate** that converts a UTC date to standard date format.

```
Function WMIDateStringToDate(dtmInstallDate)
 WMIDateStringToDate = CDate(Mid(dtmInstallDate, 5, 2) & "/" & _
 Mid(dtmInstallDate, 7, 2) & "/" & Left(dtmInstallDate, 4) & " " & Mid(dtmInstallDate, 9, 2) & ":" & _
 Mid(dtmInstallDate, 11, 2) & ":" & Mid(dtmInstallDate, 13, 2))
```

**End Function**

The code does this by defining dtmInstallDate as the input parameter. If a UTC date is passed to this function, the function converts the date using these steps in the following order:

1. Extracts the month.
2. Appends a backslash (/).
3. Extracts the day.
4. Appends a backslash.
5. Extracts the year.
6. Adds a space.
7. Extracts the hour.
8. Appends a colon (:).
9. Extracts the minutes.
10. Appends a colon.
11. Extracts the seconds.
12. Converts the resultant string to a date using the **CDate** function.

The following script uses the **WMIDateStringToDate** function to convert the install date from the **Win32\_OperatingSystem** class from UTC to standard date format. Then the script echoes the standard date to the screen.

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set objOS = objSWbemServices.ExecQuery("SELECT * FROM Win32_OperatingSystem")
For Each strOS in objOS
 dtmInstallDate = strOS.InstallDate
 strReturn = WMIDateStringToDate(dtmInstallDate)
 Wscript.Echo strReturn
Next
```

## Converting a Standard Date to UTC Date-Time Format

The diagram illustrates the two-step process for converting a standard date to UTC:

- Determine the offset between UTC and your time zone**

```
Set colTimeZone = objSWbemServices.ExecQuery _
("SELECT * FROM Win32_TimeZone")
For Each objTimeZone in colTimeZone
 strBias = objTimeZone.Bias
Next
```
- Convert the date to a UTC value**
  - Isolate individual components of date using date functions in VBScript: Year, Month, Day
  - Ensure that all months and days have two-digit values

```
dtmMonth = Month(dtCurrentDate)
If Len(dtmMonth) = 1 Then
 dtmMonth = "0" & dtmMonth
End If
```

  - Set time to string and subtract offset
  - Concatenate the string values to construct a UTC value

---

The preceding topic explains that you cannot use standard date-time formats—such as October 18, 2006—when writing WMI queries. Instead, you must convert any dates used in your queries to UTC format. This requires two steps:

1. You must determine the offset (difference in minutes) between your time zone and UTC.
2. You must convert the date to a UTC value.

The WMI **Win32\_TimeZone** class includes a **Bias** property that returns the UTC offset. The following script demonstrates how to access this property.

```
strComputer = ".."
Set objSWbemServices = GetObject("winmgmts:" _
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colTimeZone = objSWbemServices.ExecQuery _
("SELECT * FROM Win32_TimeZone")
For Each objTimeZone in colTimeZone
 Wscript.Echo "Offset: "& objTimeZone.Bias
Next
```

The following example shows the value that is echoed to the command window when this script runs in Microsoft CScript on a computer operating on Pacific Time.

```
Offset: -480
```

After you determine the UTC offset, you must then convert a standard date such as October 18, 2006 to a UTC date. To convert a standard date to a UTC date, you can use date functions in Visual Basic, Scripting Edition, such as **Year**, **Month**, and **Day**, to isolate the individual components that make up a UTC date. After you have individual

values for these components, you can concatenate them in the same manner as you would any other string value.

**Note:** UTC dates are treated as strings because the UTC offset must be appended to the end.

For example, the standard date October 18, 2006 has the following components:

- Year: 2006
- Month: 10
- Day: 18

Your script must combine these three values, the string "113047.000000" (representing the time, including milliseconds), and the UTC offset to create a UTC date.

**Note:** You can use the functions **Hour**, **Minute**, and **Second** in Visual Basic, Scripting Edition to convert the time portion of a UTC date. Thus, a time such as 11:30:47 A.M. would be converted to 113047.

There is one complicating factor. The month must take up Positions 5 and 6 in the string; the day must take up Positions 7 and 8. This is not a problem with month 10 and day 18. But how do you get July 5 (month 7, day 5) to fill up the requisite positions?

The answer is to add a leading zero to each value, thus changing the 7 to 07 and the 5 to 05. To do this, use the VBScript **Len** function to check the length (number of characters) in the month and the day. If the length is 1 (meaning that there is just one character), add a leading zero.

```
If Len(dtmMonth) = 1 Then
 dtmMonth = "0" & dtmMonth
End If
```

The following script determines the UTC offset, and then converts a specified current date (in this case, October 18, 2006) to UTC date-time format. After the date has been converted, that value is used to search a computer and returns a list of all the folders that were created after October 18, 2006.

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" _
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colTimeZone = objSWbemServices.ExecQuery _
("SELECT * FROM Win32_TimeZone")
For Each objTimeZone in colTimeZone
 strBias = objTimeZone.Bias
Next

dtmCurrentDate = "10/18/2006"
dtmTargetDate = Year(dtmCurrentDate)

dtmMonth = Month(dtmCurrentDate)
```

```
If Len(dtmMonth) = 1 Then
 dtmMonth = "0" & dtmMonth
End If

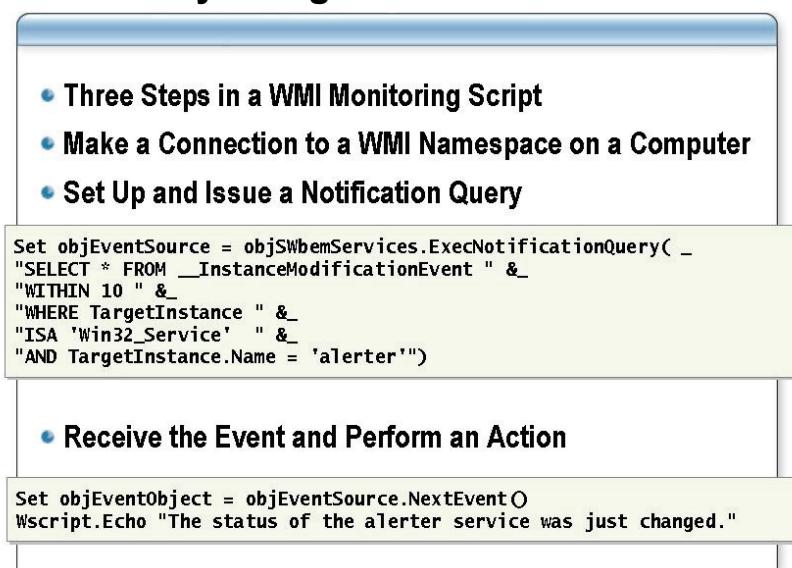
dtmTargetDate = dtmTargetDate & dtmMonth

dtmDay = Day(dtmCurrentDate)
If Len(dtmDay) = 1 Then
 dtmDay = "0" & dtmDay
End If

dtmTargetDate = dtmTargetDate & dtmDay & "000000.000000"
dtmTargetDate = dtmTargetDate & Cstr(strBias)

Set colFolders = objSWbemServices.ExecQuery _
 ("SELECT * FROM Win32_Directory WHERE CreationDate < '" & _
 dtmTargetDate & "'")
For Each objFolder in colFolders
 Wscript.Echo objFolder.Name
Next
```

## Monitoring Resources by Using Event Notifications




---

By using WMI event notifications, you can monitor the state of any WMI-managed resource and respond to an issue much earlier, perhaps before your users even notice.

For example, instead of waiting for a problem related to insufficient free disk space on a server to be brought to your attention, you can set up a WMI event notification that notifies you by e-mail when the free disk space on a server falls below a specified value. By knowing about the potential problem before it occurs, you can solve it before any productivity is lost—in this case, perhaps by archiving some of the data on the hard disk or by installing an additional hard disk.

### Three Steps in a WMI Monitoring Script

Just as there are three primary steps in a WMI script that retrieves and displays the properties of a managed resource, you follow three steps to create a WMI monitoring script.

### Make a Connection to a WMI Namespace on a Computer

The first step is the same as the first step in most WMI scripts. A connection is made to the namespace where the WMI class corresponding to the resource being monitored is located.

```

strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" &
"{impersonationLevel=impersonate}!" &
"\\" & strComputer & "\root\cimv2")

```

### Set Up and Issue a Notification Query

In the second step, a notification query is issued using WQL. You create an event source object by calling the **ExecNotificationQuery** method and passing it the query string as

an argument. The query looks similar to the WQL statements used previously in this chapter, although there are several new elements such as **WITHIN** and **ISA**.

```
Set objEventSource = objSWbemServices.ExecNotificationQuery(_
"SELECT * FROM __InstanceModificationEvent " &
"WITHIN 10 " &
"WHERE TargetInstance " &
"ISA 'Win32_Service' " &
"AND TargetInstance.Name = 'alerter'")
```

### Receive the Event and Perform an Action

In the last step, you create an instance of the event object by calling the **NextEvent** method on the event source object. When you execute the script, it pauses at the **NextEvent** method call until the event you have registered for occurs. Then the script performs the action you specify. In this example, the script writes a line that to the screen that indicates the status of the alerter service has changed.

```
Set objEventObject = objEventSource.NextEvent()
Wscript.Echo "The status of the alerter service was just changed."
```

## Differences Between Event Notification Queries and WMI Queries

- **Event Classes**
- **WITHIN Keyword**
- **TargetInstance Object**
- **ISA Keyword**

---

Despite the similarities between standard WMI queries and event notification queries, there are some significant differences.

### Event Classes

Instead of selecting instances from a WMI class that represents a managed resource, a WQL event notification query selects instances from an event class. The

**\_InstanceModificationEvent** class represents the event that occurs when an instance is modified. There are also event classes that represent the events that occur when an instance is created or deleted: **\_InstanceDeletionEvent** and **\_InstanceCreationEvent**. Each of these three classes derives from the more general **\_InstanceOperationEvent** class, which contains events generated whenever an instance is created, deleted, or modified.

### WITHIN Keyword

Because the **Win32\_Service** class does not have a corresponding WMI event provider, you must use the **WITHIN** keyword to signify that the WMI polling mechanism must be used with a polling interval of 10 seconds. This polling function checks the CIM repository every 10 seconds to see whether any new changes have been made to the Alerter service. At most, you are notified in 10 seconds if the Alerter service is modified.

If you do not include the **WITHIN** keyword when the resource being monitored does not have a corresponding event provider, you receive an error message.

## TargetInstance Object

You can specify the instances of WMI classes about which you want to be notified by using the **TargetInstance** object. Using **TargetInstance** provides a way to refer to the instances you would like your query to return.

**TargetInstance** is an object, created in response to an event, that has the same properties (and values) of the object that triggered the event. For example, if the Alerter service is stopped, the **TargetInstance** object will be an instance of the **Win32\_Service** class that has the **Name** property set to the name of the Alerter service and the **State** set to **Stopped** (plus any other properties of the Alerter service).

## ISA Keyword

You can check whether a particular object belongs to a certain class by using the **ISA** keyword, which is roughly equivalent to the equal sign. For example, if you want to capture all instances of the **Win32\_Processor** class on a computer, you could use the **ISA** keyword to limit the **TargetInstance** object to search only for **Win32\_Processor** objects.

The following code example demonstrates how to construct a query that uses the **\_\_InstanceModificationEvent** class. The query uses the **WHERE** and **ISA** keywords to define the **Win32\_Processor** class. It also uses the **AND** keyword to specify that the **DeviceID** property must have the value CPU0 and that the **LoadPercentage** property must have a value greater than 90. The **ExecNotificationQuery** method registers the script's interest in the event, and the **NextEvent** method pauses the query until the **LoadPercentage** property reaches 90 percent.

```
Set objSWbemServices = _
GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")

strWQL = "SELECT * FROM __InstanceModificationEvent " &_
"WITHIN 5 " &_
"WHERE TargetInstance ISA 'Win32_Processor' " &_
"AND TargetInstance.DeviceID='CPU0' " &_
"AND TargetInstance.LoadPercentage > 90"

Set objEventSource = objSWbemServices.ExecNotificationQuery(strWQL)
Set objEventObject = objEventSource.NextEvent()
Wscript.Echo "Load Percentage on CPU0 exceeded 90%."
```

**Note:** You can include additional constraints on the values of any of the properties of the class, making use of the **AND** and **OR** keywords to combine the constraints.

For more examples of event notification scripts, see the Microsoft Script Center.

## Lesson 4: Scripting Common Administrative Tasks by Using WMI

- Accessing Event Log Information
- Sample WMI Event Log Scripts
- Setting File and Folder Permissions

There are a number of common administrative tasks that you can automate by using WMI scripts. This lesson describes two: accessing information from the event log and setting file and folder permissions.

**Note:** For examples of WMI scripts that perform other common tasks, see the Microsoft Script Center.

### Objectives

After completing this lesson, you will be able to:

- Write scripts that access information from the event log.
- Understand how to set permissions on a file or folder using WMI scripts.

## Accessing Event Log Information

- **Win32\_NTEventLogFile and Win32\_NTLogEvent Classes**
- **Create WMI Scripts to Automate Event Log Analysis**
  - Periodically return any instances of a particular event
  - Run daily to analyze and store event log entries
- **Suggested Techniques**
  - Specify a single event log
  - Return a subset of events
  - Return one day's events
  - Asynchronously return event records
  - Copy events to a database

---

You can use WMI scripts to query event logs on local and remote computers.

### **Win32\_NTEventLogFile and Win32\_NTLogEvent Classes**

You can obtain information from the Windows event log by using the **Win32\_NTEventLogFile** or the **Win32\_NTLogEvent** classes in your WMI queries. The **Win32\_NTLogEvent** class contains several properties that enable you to access the information that is stored in an event log file.

**Note:** To access the Security event log, your scripts must have **SeSecurityPrivilege** permissions, otherwise “Access denied” is the message returned to the application.

The following example shows how to use the **Win32\_NTLogEvent** class in a query that obtains information from the System log file.

```
Set colLoggedEvents = objWMIService.ExecQuery _
 ("SELECT * FROM Win32_NTLogEvent WHERE Logfile = 'System'")
```

The **Win32\_NTEventLogFile** class represents an instance of an event log file. In addition to properties that you can use to access event log data, **Win32\_NTEventLogFile** has a number of methods that enable you to manage event logs. You can use an instance of this class to configure event log properties, and back up and clear event logs. The following example shows how to use the **Win32\_NTEventLogFile** class in a query that obtains information from the Application event log.

```
Set colLogfiles = objWMIService.ExecQuery _
 ("Select * from Win32_NTEventLogFile " _
 & "Where LogFileName='Application'")
```

## Create WMI Scripts to Automate Event Log Analysis

The **Win32\_NTLogEvent** and **Win32\_NTEventLogFile** classes enable you to write WMI scripts that automate analysis tasks on the event log. Examples of some of the tasks that you can automate include:

- Periodically returning any instances of a particular event (for example, a failed logon or failed service).
- Running the script in the morning to retrieve all of the events that occurred on the previous day. The script can then copy the event records to a database, where you can easily sort, filter, analyze, and print them.

The following table shows several key event log properties that you can access.

| Property            | Description                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Category</b>     | The classification of the event as determined by the source.                                                          |
| <b>ComputerName</b> | The name of the computer on which the event occurred.                                                                 |
| <b>EventCode</b>    | The identification number for a particular kind of event.                                                             |
| <b>Message</b>      | The event description.                                                                                                |
| <b>RecordNumber</b> | The record number for the event.                                                                                      |
| <b>SourceName</b>   | The application that generated the event.                                                                             |
| <b>TimeWritten</b>  | The time when the record was written to the event log in UTC format.                                                  |
| <b>Type</b>         | The error type. Values are:<br>1 = Error<br>2 = Warning<br>4 = Information<br>8 = Audit success<br>16 = Audit failure |
| <b>User</b>         | The user account on which the event occurred.                                                                         |

## Suggested Techniques

Because of the time that it can take an event log query to run and the large amount of information that can be returned, it is important that you carefully design your queries for retrieving data from the event logs. Unless you must return all of the events from all of the event logs on a computer, write your queries so that they do one or more of the following:

- Specify a single event log. If events are recorded in only one event log, it will not be necessary to search the other logs for these records. For example, computer startup

and shutdown events are recorded only in the System event log. If you are querying for these records, there is no reason to search the Application or DNS event logs.

- Return a subset of events. When analyzing event logs, it is rarely necessary to examine every event. The vast majority of records in a typical event log report the success of a particular operation. If you are interested only in operations that failed, there is no reason to examine hundreds of records that report operations that succeeded. Instead, you can focus on a subset of events: those records reporting an operation that failed.
- Return one day's events. For most computers, this kind of query returns only a few hundred records, compared with thousands of records stored in a large event log. If you want to view only the event records for last Tuesday, construct your query so that only the records from that day are returned.
- Asynchronously return event records. If a dataset is large, asynchronous queries will usually run faster than semi-synchronous queries.
- Copy events to a database. Unlike databases, event logs are not optimized for data retrieval and analysis. If you must perform a number of queries on your event logs, or if you must combine events from multiple event logs, it is recommended that you copy those events to a database and then use the database and its tools to analyze the records. In addition to the speed differential, databases typically provide more sophisticated tools for analyzing data and calculating statistics than event logs.

## Sample WMI Event Log Scripts

- Query an Event Log for a Subset of Events

```

strComputer = "."
Set objWMIService = GetObject("winmgmts:"_
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colLoggedEvents = objWMIService.ExecQuery _
("SELECT * FROM Win32_NTLogEvent WHERE Logfile = 'System' AND " _
& "EventCode = '6008'")
Wscript.Echo "Improper shutdowns: " & colLoggedEvents.Count

```

- Retrieve Event Log Records from a Specified Day

```

dtmStartDate = "20061219000000.000000-480"
dtmEndDate = "20061220000000.000000-480"
strComputer = "."
Set objWMIService = GetObject("winmgmts:"_
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colEvents = objWMIService.ExecQuery _
("Select * from Win32_NTLogEvent Where TimeWritten >= '" _
& dtmStartDate & "' and TimeWritten < '" & dtmEndDate & "'")
For each objEvent in colEvents
 ' Perform tasks
Next

```

A number of WMI scripts are available on the Microsoft Script Center. The following sections examine sample scenarios.

### Query an Event Log for a Subset of Events

The following script queries the **Win32\_NTLogEvent** class. The query uses the **LogFile** property to gather entries only from the System event log; it also uses the **EventCode** property to limit the events for which it is searching. In this case, the query specifies the 6008 event code. After it has gathered the data, the script echoes the number of events in the returned collection to the screen.

```

strComputer = "."
Set objWMIService = GetObject("winmgmts:"_
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colLoggedEvents = objWMIService.ExecQuery _
("SELECT * FROM Win32_NTLogEvent WHERE Logfile = 'System' AND " _
& "EventCode = '6008'")
Wscript.Echo "Improper shutdowns: " & colLoggedEvents.Count

```

### Retrieve Event Log Records from a Specified Day

The following example shows how to obtain all event log records between December 19, 2006 and December 22, 2006. The dates are **dtmStartDate** and **dtmEndDate** variables in UTC format. The script queries all instances of the **Win32\_NTLogEvent** class on the local computer. The script uses the **Win32\_NTLogEvent TimeWritten** property along with the date variables to establish the date range for the query.

After the script has gathered the data, it echoes the data to the screen, including the name of the Windows event log from which the script has gathered the data.

```

dtmStartDate = "20061219000000.000000-480"
dtmEndDate = "20061220000000.000000-480"

```

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:"_
& "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2")
Set colEvents = objWMIService.ExecQuery _
("Select * from Win32_NTLogEvent Where TimeWritten >= '" _
& dtmStartDate & "' and TimeWritten < '" & dtmEndDate & "'")
For each objEvent in colEvents
Wscript.Echo "Category: " & objEvent.Category
Wscript.Echo "Computer Name: " & objEvent.ComputerName
Wscript.Echo "Event Code: " & objEvent.EventCode
Wscript.Echo "Message: " & objEvent.Message
Wscript.Echo "Record Number: " & objEvent.RecordNumber
Wscript.Echo "Source Name: " & objEvent.SourceName
Wscript.Echo "Time Written: " & objEvent.TimeWritten
Wscript.Echo "Event Type: " & objEvent.Type
Wscript.Echo "User: " & objEvent.User
Wscript.Echo objEvent.LogFile
Next
```

## Setting File and Folder Permissions

- Create a Win32\_LogicalFileSecuritySetting Object
- Call the GetSecurityDescriptor Method
- Get the DACL from the Security Descriptor
- Modify the ACEs
- Call the SetSecurityDescriptor Method

---

Another common task that you can automate with WMI scripts is setting file and folder permissions. There are five essential steps that your script must follow to perform this task.

### Create a Win32\_LogicalFileSecuritySetting Object

The first step is to create a WMI object by using the **Win32\_LogicalFileSecuritySetting** class and specify the path to the folder or file on which you want to set permissions.

```
Set wmiFileSecSetting = GetObject("winmgmts:Win32_LogicalFileSecuritySetting." & _
 "path='c:\\testfolder'")
```

### Call the GetSecurityDescriptor Method

The second step is to call the **GetSecurityDescriptor** method. This gets the security descriptor that you must modify to set folder permissions. The following example shows how to call this method.

```
' Obtain the existing security descriptor for folder
RetVal = wmiFileSecSetting. _ GetSecurityDescriptor(wmiSecurityDescriptor)
If Err <> 0 Then
 WScript.Echo "GetSecurityDescriptor failed" & _
 VBCRLF & Err.Number & VBCRLF & Err.Description
 WScript.Quit
Else
 WScript.Echo "GetSecurityDescriptor succeeded"
End If
```

### Get the DACL from the Security Descriptor

The third step is to get the discretionary access control list (DACL) from the security descriptor for the folder. The DACL is an array of access control entries (ACEs). In WMI, a **Win32\_ACE** object represents each ACE.

```
DACL = wmiSecurityDescriptor.DACL
```

### Modify the ACEs

The fourth step is to change the property values of one or more ACEs. The following code changes the **AccessMask** property, and then echoes the new value to the screen.

```
' Set read access to the owner, group,
' and DACL of the security descriptor (131072)
wmiAce.AccessMask = 131072
wscript.echo "Access Mask: " & wmiAce.AccessMask
```

### Call the SetSecurityDescriptor Method

The final step is to call the **SetSecurityDescriptor** method to apply the new security settings to the folder.

```
RetVal = wmiFileSecSetting.SetSecurityDescriptor(wmiSecurityDescriptor)
Wscript.Echo "ReturnValue is: " & RetVal
```

The following example is the complete script that changes the permissions on a folder named testfolder on drive C of the local computer.

```
' Connect to WMI and get the file security
' object for the testfolder directory
Set wmiFileSecSetting = GetObject(_
 "winmgmts:Win32_LogicalFileSecuritySetting." &
 "path='c:\\\\testfolder'")

' Use the Win32_LogicalFileSecuritySetting Caption
' property to create a simple header before
' clearing the discretionary access control list (DACL).
Wscript.Echo wmiFileSecSetting.Caption & ":" & vbCrLf

' Obtain the existing security descriptor for folder
RetVal = wmiFileSecSetting.
 GetSecurityDescriptor(wmiSecurityDescriptor)
If Err <> 0 Then
 WScript.Echo "GetSecurityDescriptor failed" &
 VBCRLF & Err.Number & VBCRLF & Err.Description
 WScript.Quit
Else
 WScript.Echo "GetSecurityDescriptor succeeded"
End If

' Retrieve the content of Win32_SecurityDescriptor
' DACL property.
' The DACL is an array of Win32_ACE objects.
DACL = wmiSecurityDescriptor.DACL

' Display the control flags in the descriptor.
Wscript.Echo "Control Flags: " &
 wmiSecurityDescriptor.ControlFlags

' Obtain the trustee for each access
' control entry (ACE) and change the permissions
' in the AccessMask for each ACE to read, write, and delete.
For each wmiAce in DACL
```

MCT USE ONLY. STUDENT USE PROHIBITED

```
' Get Win32_Trustee object from ACE
 Set Trustee = wmiAce.Trustee
' wscript.echo "Trustee Domain: " & Trustee.Domain
' wscript.echo "Trustee Name: " & Trustee.Name
' wscript.echo "Access Mask: " & wmiAce.AccessMask
' Set read access to the owner, group,
' and DACL of the security descriptor (131072)
 wmiAce.AccessMask = 131072
 wscript.echo "Access Mask: " & wmiAce.AccessMask
Next

' Call the Win32_LogicalFileSecuritySetting.
' SetSecurityDescriptor method
' to write the new security descriptor.
RetVal = wmiFileSecSetting. _
 SetSecurityDescriptor(wmiSecurityDescriptor)

Wscript.Echo "ReturnValue is: " & RetVal
```

## Lab: Writing WMI Scripts



- Exercise 1  
Using the PrimalScript WMI Wizard
- Exercise 2  
Managing Remote Resources by  
Using WMI
- Exercise 3  
Limiting Event Log Data Returned to a  
Script

---

After completing this lab, you will be able to:

- Understand how to use the PrimalScript WMI Wizard to help you to create WMI scripts.
- Write WMI scripts that retrieve information from managed resources in your enterprise.
- Write WMI scripts that access the event log and limit the amount of data returned from the event log.

Estimated time to complete this lab: 45 minutes

### Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the 2433B-LON-DC1 virtual machine first and then start the 2433B-VISTA-CL1-08 virtual machine.
- Log on to the 2433B-VISTA-CL1-08 virtual machine as **FOURTHCOFFEE\Administrator** with a password of **Pa\$\$w0rd**. Do not log on to the 2433B-LON-DC1 virtual machine.

### Lab Scenario

You are the administrator of the Fourth Coffee corporate network. To monitor the Application event log and memory usage of the domain controller, you must write WMI scripts that access the domain controller from the client computer.

## Exercise 1

### Using the PrimalScript WMI Wizard

In this exercise, you will use the PrimalScript WMI Wizard to create a WMI script. The wizard creates a WMI code template for connecting to and scripting against objects that you can manage by using WMI.

The principal tasks for this exercise are as follows:

- To create a WMI script by using the PrimalScript WMI Wizard.

| Task                                                            | Supporting information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. To create a WMI script by using the PrimalScript WMI Wizard. | <ul style="list-style-type: none"><li>• Start PrimalScript.</li><li>• Using the <b>ScriptFiles</b> template list, create a new VBScript file.</li><li>• On the <b>Script</b> menu, point to <b>Wizards</b>, and then click <b>0 WMI Wizard</b>.</li><li>• Review the list of classes in the <b>Classes</b> list, and choose a class to script against.</li><li>• Copy the code generated by the wizard and paste it into the VBScript file you created.</li><li>• Test the script by using the <b>Run Script</b> command from the <b>Script</b> menu and reviewing the results in the output pane.</li><li>• The WMI Wizard always uses the <b>SWbemLocator</b> class to connect to the WMI service on the specified computer.</li></ul> |

#### Questions

**Q:** List five WMI providers that ship with Windows XP, Windows Vista, or Windows Server 2003.

**Q:** What date format do WMI objects use?

## Exercise 2

### Managing Remote Resources by Using WMI

In this exercise, you will create a WMI script that manages resources on the domain controller, LON-DC1. You will write a script that determines the memory usage statistics on the server.

The principal tasks for this exercise are:

- To connect to the WMI service on a remote computer.
- To gather memory information from the remote computer by using WMI.

| Task                                                                   | Supporting information                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. To connect to the WMI service on a remote computer.                 | <ul style="list-style-type: none"><li>• Using the <b>ScriptFiles</b> template list, create a new VBScript file.</li><li>• Connect to the WMI service on the domain controller by using the WMI moniker.</li></ul>                                                                                                                                                             |
| 2. To gather memory information from the remote computer by using WMI. | <ul style="list-style-type: none"><li>• Construct WMI queries to gather memory usage information from the server by using the <b>Win32_ComputerSystem</b> and <b>Win32_OperatingSystem</b> classes.</li><li>• Save the file and use the <b>Run Script</b> command from the <b>Script</b> menu to test the script.</li><li>• Observe the results in the output pane.</li></ul> |

### Questions

**Q:** What WMI component determines whether the request involves static data stored in the Common Information Model (CIM) repository or dynamic data supplied by a provider?

## Exercise 3

### Limiting Event Log Data Returned to a Script

In this exercise, you will write a WMI script that accesses the Application event log on the domain controller, LON-DC1. You will write code that limits the amount of data returned from the event log by the date when the event occurred.

The principal tasks for this exercise are:

- To establish the date range.
- To connect to the WMI service on the remote computer.
- To query the Application event log and return property values.

| Task                                                              | Supporting information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. To establish the date range.                                   | <ul style="list-style-type: none"> <li>• Using the <b>ScriptFiles</b> template list, create a new VBScript file.</li> <li>• Assign two variables, <b>dtmStartDate</b> and <b>dtmEndDate</b>, to two <b>WbemScripting.SWbemDateTime</b> objects to establish a start date and end date between which to return event log data.</li> <li>• Establish the date to check that is relative to the current date. Set a variable to Date – 5 for five days or Date – 6 for six days.</li> <li>• Use the <b>SetVarDate</b> method to establish the start date for <b>dtmStartDate</b> and the end date for <b>dtmEndDate</b>.</li> </ul>                                                                                                                                             |
| 2. To connect to the WMI service on the remote computer.          | <ul style="list-style-type: none"> <li>• Connect to the WMI service on the domain controller by using the WMI moniker.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 3. To query the Application event log and return property values. | <ul style="list-style-type: none"> <li>• Create a WMI SELECT query that contacts the Application event log through the <b>Win32_NTLogEvent</b> class.</li> <li>• Include the following string in the query.           <pre>'Application' And TimeWritten &gt;= '"_&amp; dtmStartDate &amp; "' and TimeWritten &lt; '"_&amp; dtmEndDate &amp; "'"</pre> </li> <li>• Write code that accesses the <b>Category</b>, <b>ComputerName</b>, <b>EventCode</b>, <b>Message</b>, <b>RecordNumber</b>, <b>SourceName</b>, <b>TimeWritten</b>, <b>Type</b>, and <b>User</b> properties from the event log.</li> <li>• Save the file and use the <b>Run Script</b> command from the <b>Script</b> menu to test the script.</li> <li>• Observe the results in the output pane.</li> </ul> |

### Questions

Q: What two classes can you use to access event log information?

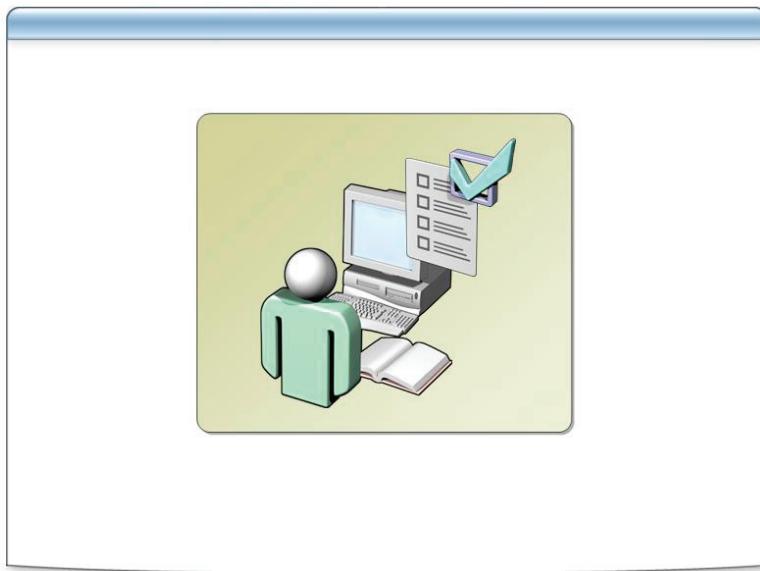
**Note:** The answers to the practices and labs are available in the **Student Course files** on the <http://www.microsoft.com/learning/companionmoc> Site.

## Lab Shutdown

After you complete the lab, you must shut down the 2433B-LON-DC1 and 2433B-VISTA-CL1-08 virtual machines and discard any changes.

**Important:** If the **Close** dialog box appears, ensure that **Turn off and delete changes** is selected and then click **OK**.

## Course Evaluation



---

Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

MCT USE ONLY. STUDENT USE PROHIBITED

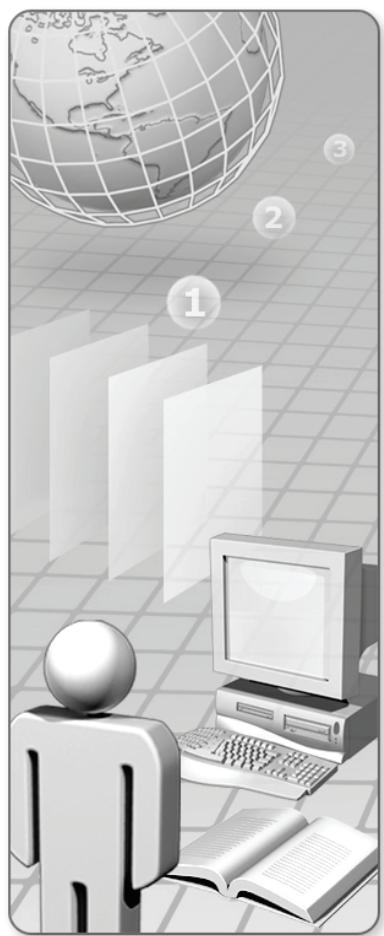
MCT USE ONLY. STUDENT USE PROHIBITED

## Appendix A: VBScript Runtime and Parsing Error Reference

### Table of Contents

Appendix A: VB Script Runtime and Parsing Error  
Reference

1



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Appendix A: VB Script Runtime and Parsing Error Reference

```
' Runtime errors:

Const VBSERR_None = 0 'No error
Const VBSERR_IllegalFuncCall = 5 'Invalid procedure call or argument
Const VBSERR_Overflow = 6 'Overflow
Const VBSERR_OutOfMemory = 7 'Out of memory
Const VBSERR_OutOfBounds = 9 'Subscript out of range
Const VBSERR_ArrayLocked = 10 'This array is fixed or temporarily
 locked
Const VBSERR_DivByZero = 11 'Division by zero
Const VBSERR_TypeMismatch = 13 'Type mismatch
Const VBSERR_OutOfStrSpace = 14 'Out of string space
Const VBSERR_CantContinue = 17 'Can't perform requested operation
Const VBSERR_OutOfStack = 28 'Out of stack space
Const VBSERR_UndefinedProc = 35 'Sub or Function not defined
Const VBSERR_DLLLoadErr = 48 'Error in loading DLL
Const VBSERR_InternalError = 51 'Internal error
Const VBSERR_BadFileNameOrNumber = 52 'Bad file name or number
Const VBSERR_FileNotFound = 53 'File not found
Const VBSERR_Bad FileMode = 54 'Bad file mode
Const VBSERR_FileAlreadyOpen = 55 'File already open
Const VBSERR_IOError = 57 'Device I/O error
Const VBSERR_FileAlreadyExists = 58 'File already exists
Const VBSERR_DiskFull = 61 'Disk full
Const VBSERR_EndOfFile = 62 'Input past end of file
Const VBSERR_TooManyFiles = 67 'Too many files
Const VBSERR_DevUnavailable = 68 'Device unavailable
Const VBSERR_PermissionDenied = 70 'Permission denied
Const VBSERR_DiskNotReady = 71 'Disk not ready
Const VBSERR_DifferentDrive = 74 'Can't rename with different drive
Const VBSERR_Path FileAccess = 75 'Path/File access error
Const VBSERR_PathNotFound = 76 'Path not found
Const VBSERR_ObjNotSet = 91 'Object variable not set
Const VBSERR_IllegalFor = 92 'For loop not initialized
Const VBSERR_CantUseNull = 94 'Invalid use of Null
Const VBSERR_CantCreateTmpFile = 322 'Can't create necessary temporary
 file
Const VBSERR_NotObject = 424 'Object required
Const VBSERR_CantCreateObject = 429 'ActiveX component can't create
 object
Const VBSERR_OLENotSupported = 430 'Class doesn't support Automation
Const VBSERR_OLEFileNotFound = 432 'File name or class name not found
 during Automation operation
Const VBSERR_OLENoPropOrMethod = 438 'Object doesn't support this property
 or method
Const VBSERR_OLEAutomationError = 440 'Automation error
Const VBSERR_ActionNotSupported = 445 'Object doesn't support this action
Const VBSERR_NamedArgsNotSupported = 446 'Object doesn't support named
 arguments
Const VBSERR_LocaleSettingNotSupported= 447 'Object doesn't support current
 locale setting
Const VBSERR_NamedParamNotFound = 448 'Named argument not found
```

```

Const VBSERR_ParameterNotOptional = 449 'Argument not optional
Const VBSERR_FuncArityMismatch = 450 'Wrong number of arguments or invalid
 property assignment
Const VBSERR_NotEnum = 451 'Object not a collection
Const VBSERR_InvalidDllFunctionName = 453 'Specified DLL function not found
Const VBSERR_CodeResourceLockError = 455 'Code resource lock error
Const VBSERR_DuplicateKey = 457 'This key is already associated with
 an element of this collection
Const VBSERR_InvalidTypeLibVariable = 458 'Variable uses an Automation type not
 supported in VBScript
Const VBSERR_ServerNotFound = 462 'The remote server machine does not
 exist or is unavailable
Const VBSERR_InvalidPicture = 481 'Invalid picture
Const VBSERR_UndefVariable = 500 'Variable is undefined
Const VBSERR_CantAssignTo = 501 'Illegal assignment
Const VBSERR_NotSafeForScripting = 502 'Object not safe for scripting
Const VBSERR_NotSafeForInitializing = 503 'Object not safe for initializing
Const VBSERR_NotSafeForCreating = 504 'Object not safe for creating
Const VBSERR_InvalidReference = 505 'Invalid or unqualified reference
Const VBSERR_ClassNotDefined = 506 'Class not defined
Const VBSERR_ComponentException = 507 'An exception occurred
Const VBSERR_ElementNotFound = 32811 'Element not found
Const VBSERR_NeedRegExp = 5016 'Regular Expression object expected
Const VBSERR_RegExpSyntax = 5017 'Syntax error in regular expression
Const VBSERR_RegExpBadQuant = 5018 'Unexpected quantifier
Const VBSERR_RegExpNoBracket = 5019 'Expected ']' in regular expression
Const VBSERR_RegExpNoParen = 5020 'Expected ')' in regular expression
Const VBSERR_RegExpBadRange = 5021 'Invalid range in character set

' Parse errors

Const VBSERR_noMemory = 1001 ' Out of memory
Const VBSERR_syntax = 1002 ' Syntax error
Const VBSERR_noColon = 1003 ' Expected ':'
Const VBSERR_noLparen = 1005 ' Expected '('
Const VBSERR_noRparen = 1006 ' Expected ')'
Const VBSERR_noBrack = 1007 ' Expected ']'
Const VBSERR_noIdent = 1010 ' Expected identifier
Const VBSERR_noEq = 1011 ' Expected '='
Const VBSERR_noIf = 1012 ' Expected 'If'
Const VBSERR_noTo = 1013 ' Expected 'To'
Const VBSERR_noEnd = 1014 ' Expected 'End'
Const VBSERR_noFnc = 1015 ' Expected 'Function'
Const VBSERR_noSub = 1016 ' Expected 'Sub'
Const VBSERR_noThen = 1017 ' Expected 'Then'
Const VBSERR_noWend = 1018 ' Expected 'Wend'
Const VBSERR_noLoop = 1019 ' Expected 'Loop'
Const VBSERR_noNext = 1020 ' Expected 'Next'
Const VBSERR_noCase = 1021 ' Expected 'Case'
Const VBSERR_noSelect = 1022 ' Expected 'Select'
Const VBSERR_noExpr = 1023 ' Expected expression
Const VBSERR_noStmt = 1024 ' Epected statement
Const VBSERR_noEOS = 1025 ' Expected end of statement
Const VBSERR_noIntCns = 1026 ' Expected integer constant
Const VBSERR_noWhUn = 1027 ' Expected 'While' or 'Until'

```

MCT USE ONLY. STUDENT USE PROHIBITED

```
Const VBSERR_noWhUnEOS = 1028 ' 'Until' or end of statement)
Const VBSERR_noWith = 1029 ' Expected 'With'
Const VBSERR_idTooLong = 1030 ' Identifier too long
Const VBSERR_badNumber = 1031 ' Invalid number
Const VBSERR_illegalChar = 1032 ' Invalid character
Const VBSERR_noStrEnd = 1033 ' Unterminated string constant
Const VBSERR_noCmtEnd = 1034 ' Unterminated comment
Const VBSERR_badMeUse = 1037 ' Invalid use of 'Me' keyword
Const VBSERR_noDo = 1038 ' 'loop' without 'do'
Const VBSERR_badExit = 1039 ' Invalid 'exit' statement
Const VBSERR_badForVar = 1040 ' Invalid 'for' loop control variable
Const VBSERR_redefName = 1041 ' Name redefined
Const VBSERR_not1stOnLine = 1042 ' Must be first statement on the line
Const VBSERR_asgByRef = 1043 ' Cannot assign to non-ByVal argument
Const VBSERR_badParens = 1044 ' Cannot use parentheses when calling
 a Sub
Const VBSERR_notConst = 1045 ' Expected literal constant
Const VBSERR_noIn = 1046 ' Expected 'In'
Const VBSERR_noClass = 1047 ' Expected 'Class'
Const VBSERR_inClass = 1048 ' Must be defined inside a Class
Const VBSERR_noPropSpec = 1049 ' Expected Let or Set or Get in
 property declaration
Const VBSERR_noProp = 1050 ' Expected 'Property'
Const VBSERR_wrongArgsNum = 1051 ' Number of arguments must be
 consistent across properties specification
Const VBSERR_redefDefault = 1052 ' Cannot have multiple default
 property/method in a Class

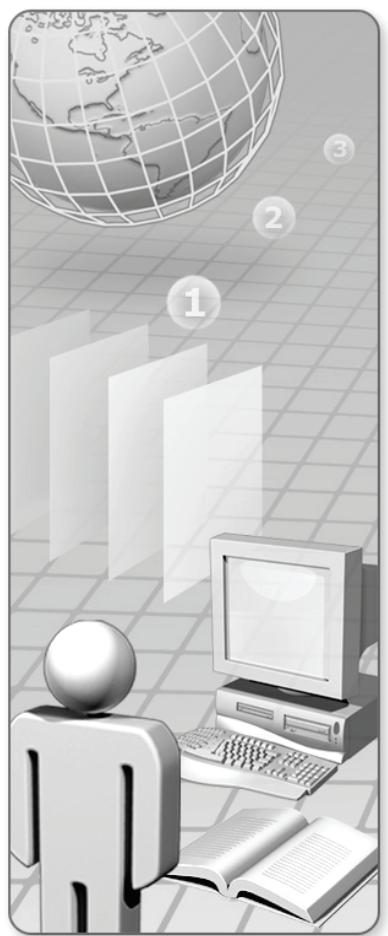
Const VBSERR_noArgs = 1053 ' Class initialize or terminate do
 not have arguments
Const VBSERR_doArg = 1054 ' Property set or let must have at
 least one argument
Const VBSERR_badNext = 1055 ' Unexpected 'Next'
Const VBSERR_badDefault = 1056 ' 'Default' can be specified only on
 'Property' or 'Function' or 'Sub'
Const VBSERR_illegalDefault = 1057 ' 'Default' specification must also
 specify 'Public'
Const VBSERR_defaultOnGet = 1058 ' 'Default' specification can only be
 on Property Get
```

MCT USE ONLY. STUDENT USE PROHIBITED

## Appendix B: Reference Error Codes for ADSI 2.5

### Table of Contents

Appendix B: Reference Error Codes for ADSI 2.5      1



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, ActiveX, BitLocker, BizTalk, Excel, Front Page, Internet Explorer, Jscript, MSDN, Outlook, PowerPoint, SQL Server, Visual Basic, Visual C++, Visual Studio, Win32, Windows, Windows Media, Windows NT, Windows PowerShell, Windows Server and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Appendix B: Reference Error Codes for ADSI 2.5

| ADSI Error Code | LDAP message                   | Win32 message                      | Description                                          |
|-----------------|--------------------------------|------------------------------------|------------------------------------------------------|
| 0L              | LDAP_SUCCESS                   | NO_ERROR                           | Operation succeeded.                                 |
| 0x80070005L     | LDAP_INSUFFICIENT_RIGHTS       | ERROR_ACCESS_DENIED                | The user has insufficient access right.              |
| 0x80070008L     | LDAP_NO_MEMORY                 | ERROR_NOT_ENOUGH_MEMORY            | The system is out of memory.                         |
| 0x8007001fL     | LDAP_OTHER                     | ERROR_GEN_FAILURE                  | Unknown error occurred.                              |
| 0x800700eaL     | LDAP_PARTIAL_RESULTS           | ERROR_MORE_DATA                    | Partial results and referrals received.              |
| 0x800700eaL     | LDAP_MORE_RESULTS_TO_RETURN    | ERROR_MORE_DATA                    | More results are to be returned.                     |
| 0x800704c7L     | LDAP_USER_CANCELLED            | ERROR_CANCELLED                    | The user has cancelled the operation.                |
| 0x800704c9L     | LDAP_CONNECT_ERROR             | ERROR_CONNECTION_REFUSED           | Cannot establish the connection.                     |
| 0x8007052eL     | LDAP_INVALID_CREDENTIALS       | ERROR_LOGON_FAILURE                | The supplied credential is invalid.                  |
| 0x800705b4L     | LDAP_TIMEOUT                   | ERROR_TIMEOUT                      | The search was timed out.                            |
| 0x80071392L     | LDAP_ALREADY_EXISTS            | ERROR_OBJECT_ALREADY_EXISTS        | The object already exists.                           |
| 0x8007200aL     | LDAP_NO_SUCH_ATTRIBUTE         | ERROR_DS_NO_ATTRIBUTE_OR_VALUE     | Requested attribute does not exist.                  |
| 0x8007200bL     | LDAP_INVALID_SYNTAX            | ERROR_DS_INVALID_ATTRIBUTE_SYNTAX  | The syntax is invalid.                               |
| 0x8007200cL     | LDAP_UNDEFINED_TYPE            | ERROR_DS_ATTRIBUTE_TYPE_UNDEFINED  | Type is not defined.                                 |
| 0x8007200dL     | LDAP_ATTRIBUTE_OR_VALUE_EXISTS | ERROR_DS_ATTRIBUTE_OR_VALUE_EXISTS | The attribute exists or the value has been assigned. |
| 0x8007200eL     | LDAP_BUSY                      | ERROR_DS_BUSY                      | The server is busy.                                  |
| 0x8007200fL     | LDAP_UNAVAILABLE               | ERROR_DS_UNAVAILABLE               | The server is not available.                         |
| 0x80072014L     | LDAP_OBJECT_CLASS_VIOLATION    | ERROR_DS_OBJ_CLASS_VIOLATION       | There was an object class violation.                 |
| 0x80072015L     | LDAP_NOT_ALLOWED_ON_NONLEAF    | ERROR_DS_CANT_ON_NON_LEAF          | Operation is not allowed on a non leaf object.       |

| ADSI<br>Error Code | LDAP message                    | Win32 message                       | Description                                      |
|--------------------|---------------------------------|-------------------------------------|--------------------------------------------------|
| 0x80072016L        | LDAP_NOT_ALLOWED_ON_RDN         | ERROR_DS_CANT_ON_RDN                | Operation is not allowed on RDN.                 |
| 0x80072017L        | LDAP_NO_OBJECT_CLASS_MODS       | ERROR_DS_CANT_MOD_OBJ_CLASS         | Cannot modify object class.                      |
| 0x80072020L        | LDAP_OPERATIONS_ERROR           | ERROR_DS_OPERATIONS_ERROR           | Operations error occurred.                       |
| 0x80072021L        | LDAP_PROTOCOL_ERROR             | ERROR_DS_PROTOCOL_ERROR             | Protocol error occurred.                         |
| 0x80072022L        | LDAP_TIMELIMIT_EXCEEDED         | ERROR_DS_TIMELIMIT_EXCEEDED         | Time limit has exceeded                          |
| 0x80072023L        | LDAP_SIZELIMIT_EXCEEDED         | ERROR_DS_SIZELIMIT_EXCEEDED         | Size limit has exceeded                          |
| 0x80072024L        | LDAP_ADMIN_LIMIT_EXCEEDED       | ERROR_DS_ADMIN_LIMIT_EXCEEDED       | Administration limit on the server has exceeded. |
| 0x80072025L        | LDAP_COMPARE_FALSE              | ERROR_DS_COMPARE_FALSE              | Compare yielded FALSE.                           |
| 0x80072026L        | LDAP_COMPARE_TRUE               | ERROR_DS_COMPARE_TRUE               | Compare yielded TRUE.                            |
| 0x80072027L        | LDAP_AUTH_METHOD_NOT_SUPPORTED  | ERROR_DS_AUTH_METHOD_NOT_SUPPORTED  | The authentication method is not supported.      |
| 0x80072028L        | LDAP_STRONG_AUTH_REQUIRED       | ERROR_DS_STRONG_AUTH_REQUIRED       | Strong authentication is required.               |
| 0x80072029L        | LDAP_INAPPROPRIATE_AUTH         | ERROR_DS_INAPPROPRIATE_AUTH         | Authentication is inappropriate.                 |
| 0x8007202aL        | LDAP_AUTH_UNKNOWN               | ERROR_DS_AUTH_UNKNOWN               | Unknown authentication error occurred.           |
| 0x8007202bL        | LDAP_REFERRAL                   | ERROR_DS_REFERRAL                   | Referral                                         |
| 0x8007202cL        | LDAP_UNAVAILABLE_CRIT_EXTENSION | ERROR_DS_UNAVAILABLE_CRIT_EXTENSION | Critical extension is unavailable.               |
| 0x8007202dL        | LDAP_CONFIDENTIALITY_REQUIRED   | ERROR_DS_CONFIDENTIALITY_REQUIRED   | Confidentiality is required.                     |
| 0x8007202eL        | LDAP_INAPPROPRIATE_MATCHING     | ERROR_DS_INAPPROPRIATE_MATCHING     | There was an inappropriate matching.             |
| 0x8007202fL        | LDAP_CONSTRAINT_VIOLATION       | ERROR_DS_CONSTRAINT_VIOLATION       | There was a constrain violation.                 |
| 0x80072030L        | LDAP_NO SUCH OBJECT             | ERROR_DS_NO_SUCH_OBJECT             | Object does not exist.                           |
| 0x80072031L        | LDAP_ALIAS_PROBLEM              | ERROR_DS_ALIAS_PROBLEM              | The alias is invalid.                            |
| 0x80072032L        | LDAP_INVALID_DN_SYNTAX          | ERROR_DS_INVALID_DN_SYNTAX          | The distinguished name has an invalid syntax.    |
| 0x80072033L        | LDAP_IS LEAF                    | ERROR_DS_IS_LEAF                    | The object is a leaf.                            |
| 0x80072034L        | LDAP_ALIAS_DEREF_PROBLEM        | ERROR_DS_ALIAS_DEREF_PROBLEM        | Can not dereference the alias.                   |

| ADSI<br>Error Code | LDAP message                 | Win32 message                     | Description                                     |
|--------------------|------------------------------|-----------------------------------|-------------------------------------------------|
| 0x80072035L        | LDAP_UNWILLING_TO_PERFORM    | ERROR_DS_UNWILLING_TO_PERFORM     | The server is unwilling to perform.             |
| 0x80072036L        | LDAP_LOOP_DETECT             | ERROR_DS_LOOP_DETECT              | Loop was detected.                              |
| 0x80072037L        | LDAP_NAMING_VIOLATION        | ERROR_DS_NAMING_VIOLATION         | There was a naming violation.                   |
| 0x80072038L        | LDAP_RESULTS_TOO_LARGE       | ERROR_DS_OBJECT_RESULTS_TOO_LARGE | Results returned are too large.                 |
| 0x80072039L        | LDAP_AFFECTS_MULTIPLE_DSAS   | ERROR_DS_AFFECTS_MULTIPLE_DSAS    | Multiple directory service agents are affected. |
| 0x8007203aL        | LDAP_SERVER_DOWN             | ERROR_DS_SERVER_DOWN              | Cannot contact the LDAP server.                 |
| 0x8007203bL        | LDAP_LOCAL_ERROR             | ERROR_DS_LOCAL_ERROR              | Local error occurred.                           |
| 0x8007203cL        | LDAP_ENCODING_ERROR          | ERROR_DS_ENCODING_ERROR           | Encoding error occurred.                        |
| 0x8007203dL        | LDAP_DECODING_ERROR          | ERROR_DS_DECODING_ERROR           | Decoding error occurred.                        |
| 0x8007203eL        | LDAP_FILTER_ERROR            | ERROR_DS_FILTER_UNKNOWN           | The search filter is bad.                       |
| 0x8007203fL        | LDAP_PARAM_ERROR             | ERROR_DS_PARAM_ERROR              | A bad parameter was passed to a routine.        |
| 0x80072040L        | LDAP_NOT_SUPPORTED           | ERROR_DS_NOT_SUPPORTED            | The feature is not supported.                   |
| 0x80072041L        | LDAP_NO_RESULTS_RETURNED     | ERROR_DS_NO_RESULTS_RETURNED      | Results are not returned.                       |
| 0x80072042L        | LDAP_CONTROL_NOT_FOUND       | ERROR_DS_CONTROL_NOT_FOUND        | The control was not found.                      |
| 0x80072043L        | LDAP_CLIENT_LOOP             | ERROR_DS_CLIENT_LOOP              | Client loop was detected.                       |
| 0x80072044L        | LDAP_REFERRAL_LIMIT_EXCEEDED | ERROR_DS_REFERRAL_LIMIT_EXCEEDED  | The referral limit has exceeded                 |

MCT USE ONLY. STUDENT USE PROHIBITED