OBJECTSTORE

WINDOWS VAR KIT

RELEASE 5

September 1997

ObjectStore Release 5 Windows VAR Kit

ObjectStore Release 5, September 1997

Object Design, Inc.
Twenty Five Mall Road
Burlington, MA 01803-4194

Contents

Chapter 1
Introduction

Purpose

The purpose of this document is to provide value-added resellers (VARs) with the information needed to either customize or replace the standard Windows ObjectStore 5 **install**, **setup**, and **uninstall** system when packaging an application with ObjectStore.

Audience

VARs who are deploying one or more ObjectStore applications.

ObjectStore Installation Components

The installation consists of two logically distinct sections:

- The user interface and flow of control elements are written as an InstallShield application.
- Most of the ObjectStore-specific elements of the system are provided in a DLL that is invoked from InstallShield.

Benefits

The two-part structure lets you generate installation applications that use a relatively stable set of ObjectStore-related APIs supplied in the DLL. This design shields you from underlying changes that might be made from one release to another. It also leaves you in control of the look and feel of your application installation.

Object Design encourages your feedback on the implementation and the capabilities provided by the DLL.

VAR kit files

The Windows ObjectStore 5 VAR kit includes a variety of files. The InstallShield script file (**SETUP.RUL**) and an associated header file (**OS_SETUP.H**) are provided in source form and can be modified to create a customized installation, or used as a reference when you develop completely new installation systems. This

release also provides two related files, **SETUP.50** and **OS_SETUP.50**. These files represent the state of the installation system as of Release 5 and are described in Overview of SETUP.50 (SETUP.RUL).

In future, you can compare the more recent files (**SETUP.RUL** and **OS_SETUP.H**) and the Release 5 versions to pinpoint changes that might have taken place since this document was completed.

The VAR kit also includes any other ObjectStore-supplied support files (such as bitmaps) and a makefile. If you have the appropriate tools (an appropriate version of InstallShield and the necessary Microsoft VC++ 5.0 tools) you should be able to recreate the standard ObjectStore installation system with the files supplied.

The **README.TXT** file included with the VAR kit contains late changes and other useful information. Be sure to check this file before beginning work on a custom installation system.

InstallShield Version Information

The ObjectStore 5 InstallShield application was built using the *International East* edition of InstallShield, Version 3.107. The specific component information, supplied by the InstallShield Version Checker, is as follows:

Component	***Version***	***Date***
InstallShield	**3.00.107**	*November 4, 1996*
compile.exe	**3.00.077**	*February 20, 1997*
icomp.exe	**3.00.062**	*January 15, 1996*
split.exe	**3.00.060**	*January 15, 1996*
packlist.exe	**3.00.060**	*January 15, 1996*
isverw.exe	**3.00.052**	*September 12, 1995*
isdbg.dll	**3.00.052**	*October 2, 1995*
unInstallShield	**2.20.920**	*November 5, 1996*

Chapter 2
ObjectStore Windows Installation System

The ObjectStore Windows installation and setup system is responsible for creating and maintaining the environment ObjectStore requires in order to operate properly. If you want to replace the standard tool, you must maintain the proper environment.

Installation

Installation requires that various directories and files be created or copied to the user's system, that the system environment be modified to point to the files, and that various entries (most described in more detail in Setup in this chapter) be established in the registry or in system services databases. Also, installations that include a Server generally require that the Server be initialized. This is also described in Setup.

Licensing options

The standard ObjectStore installation system has eight licensing options. Depending on the option selected, the installation system determines whether or not to install

- ObjectStore client files
- ObjectStore Server files
- ObjectStore development support files

These three license types are represented by flags presented to the entry **OS_GetInstallFlags**. The flags are described in Chapter 3, API Reference. These flags serve as a basis for the routines that decompress ObjectStore *zip* files. That is, the files installed are copied from the zip files to the appropriate place in the user's system. These flags determine which files are actually installed.

Optional components

In addition to variations based on licensing options, the installation system allows the following components to be installed or omitted:

- DBMS — The ObjectStore client-Server system
- ObjectStore / Single — The single machine standalone system
- Examples — For developers
- HTML documentation — Browsable ObjectStore documentation
- PostScript documentation — Documentation in PostScript or PDF format suitable for printing

Each component option is also represented by a flag presented to the entry **OS_GetInstallFlags**, described in Chapter 3, API Reference, which serves as a basis for the routines that decompress ObjectStore zip files.

Directories

Though some of the entries described in Chapter 3, API Reference, appear to allow a variety of target directories to be used for elements of an ObjectStore installation, in practice everything should be installed in various directories under a single **OS_ROOTDIR**. The directory structure should be as follows:

```
OS_ROOTDIR \
    bin
    binsngl
    doc \
        pscript
    etc
    examples
    include
    lib
```

While the contents of most of the directories are self-evident, the **etc** and **binsngl** directories need some explanation.

etc directory

The **etc** directory contains ObjectStore **catalog** files that are used to tailor messages for specific languages. If the installation includes development modules, the **etc** directory includes a **desktop.mak** file.

binsngl directory

Note that the **binsngl** directory is self-contained. Applications that run on a single machine can copy these files to any directory (most likely in the user's **PATH**) and need establish no other settings.

Environment settings

ObjectStore relies on a variety of environment variables. (There are minimal requirements for ObjectStore / Single, as noted previously.) You need the following environment variables, under some circumstances, for the full client-Server version:

OS_ROOTDIR Required. This is the parent directory of the **bin**, **lib**, and such directories.

OS_TMPDIR Recommended when you are running a Server or Cache Manager service. This is the directory where these processes store their text history files, **osserver.txt** and **oscmgr.txt**. If no value is found, the files are stored in the root directory of the boot drive.

PATH The **PATH** environment variable should include the ObjectStore **bin** directory.

INCLUDE If this is a development machine, the **INCLUDE** variable should specify the ObjectStore **include** directory.

LIB

If this is a development machine, the **LIB** variable should include the ObjectStore **lib** directory.

The **OS_ROOTDIR** and **OS_TMPDIR** values need to be set in the system environment section on Windows NT, to ensure that they are available for the Server and Cache Manager when they run as services.

The environment values are stored in the registry on Windows NT, but they are set through **AUTOEXEC.BAT** on Windows 95. This can cause some problems on dual-boot machines, so the standard ObjectStore installation creates an **OS_AUTO.BAT**, which is called from **AUTOEXEC.BAT**. This construct sets the values for Windows 95, but skips them when Windows NT processes **AUTOEXEC**.

Registration data in the registry

The standard ObjectStore installation sets a variety of values in the registry concerning product registration. These values are used by the **install** and **setup** system and might not be needed for a VAR installation. Most values are stored in

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Registration

but one is stored in

HKEY_CURRENT_USER\Software\Object Design Inc.\ObjectStore.4.0\Registration

Note that **ObjectStore.4.0** refers to the *server generation*, not to the current ObjectStore release number, and is used for both Release 4.*x* and 5.*x*.

Values stored in **HKEY_LOCAL_MACHINE** include

- Company
- Install From
- License Type
- ObjectStore Install Directory
- ObjectStore / Single Directory
- ObjectStore Examples Directory
- ObjectStore HTML Documentation Directory
- ObjectStore PostScript Documentation Directory
- Program Folder Name

The value stored in **HKEY_CURRENT_USER** is User Name.

See **OS_GetRegistration** and **OS_SetRegistration** in Chapter 3, API Reference, for information about using these functions to retrieve and set these values.

Setup

Setup operations are typically performed on a preexisting (or just created) ObjectStore installation. Setup operations generally involve

- Setting ObjectStore Server parameter values
- Creating or modifying RAWFS partition information
- Initializing or reinitializing the Server and Server log file
- Setting up the Server and Cache Manager as system services, and, optionally, starting them immediately

Setting ObjectStore Server parameter values

ObjectStore Management contains descriptions of specific Server parameters in Chapter 2, Server Parameters. For Windows systems, values are stored in the registry under the key

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

Individual parameters are stored using value names that correspond to the parameter name (for example, **"Authentication Required"**). All values are stored as strings **(REG_SZ)**.

Note that you can use the **OS_GetServerParameter** and **OS_SetServerParameter** entries, described in Chapter 3, API Reference, to get and set individual parameters. You can also use the **OS_SetServerParams** entry to run the Server Parameters dialog and update the registry appropriately.

Creating or modifying RAWFS partition information

Partition information is also stored in the registry, under the same key:

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

The RAWFS might contain multiple components, named *Partition0*, *Partition1*, ..., *PartitionN*. Partitions must start from number 0 and no gaps are allowed in the sequence. Each partition descriptor has the following format:

<type> <path> <expandability>

where

<type> is either **FILE** (Windows NT or 95) or **PARTITION** (Windows NT only).

<path> is the name of a file system file (for **FILE**) or a partition name in the form

\\.\A:

or a physical drive name in the form

\\.\PhysicalDrive1

<expandability> is either **EXPANDABLE** or **NONEXPANDABLE**. Only file system files can be **EXPANDABLE**. Partitions and physical drives are always **NONEXPANDABLE**.

If you specify a file system file, the installation must create the file. The Server overwrites the contents of the file during initialization, but it does not create the file if it does not exist. If a file system file is designated **NONEXPANDABLE**, the installation must also presize the file before initializing the Server.

Note that you can use the **OS_SetupRawfs** entry, described in Chapter 3, API Reference, to run the standard RAWFS setup dialog and update the registry.

Note also that most changes in RAWFS configurations require you to reinitialize the Server. The only exception is when you add a new component (file, partition, or disk) to an existing RAWFS. You can do this without reinitializing.

Initializing or reinitializing the Server and Server log file

You must reinitialize the ObjectStore Server following an installation or following most changes to a RAWFS configuration. You can use the entry **OS_InitLog**, described in Chapter 3, API Reference, to initialize the Server. Note that there are several entries in the registry that are relevant to Server initialization under the key

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

They are

Log File	Pathname of the Server log file, or a null string if the log is being maintained in the RAWFS
Log Version	**300**
LogInPartition	**0** to indicate that the log is in the RAWFS, **1** to indicate that it is in a file
Server Initialized	**1** after the Server has been initialized

Use the entry **OS_InitLog**, described in Chapter 3, API Reference, to initialize the Server and set the relevant values in the registry.

Setting up the Server and Cache Manager as system services

You can use the entry **OS_UpdateStartup**, described in Chapter 3, API Reference, to set up system services. This task requires two steps. First, you must update the system services databases. Then set the **Auto Start Server** value in the registry under

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

Note that system service updating follows different rules on Windows NT, which supports substantive system services, and Windows 95, which does not.

The following paragraphs characterize the components and sequence of an ObjectStore installation and setup.

Installation System Files

The ObjectStore installation and setup for Windows is an InstallShield application delivered in a number of different files. For ObjectStore Release 5, the files are as follows:

client.zip **debug.zip** **devo.zip**	Compressed forms of the ObjectStore product files.
examples.zip	ObjectStore example files.
runtime.zip **server.zip**	
single.zip	ObjectStore / Single files.
doc.zip	ObjectStore HTML documentation files.
pscript.zip	ObjectStore PostScript or PDF format documentation.
msvcrt.dll	Redistributed Microsoft C run-time DLL.
install.bat	Batch file replacing **install.exe** (executes the command **setup -install**).
readme.ico **uninst.ico**	Program manager icons for **README.txt** and for **uninstall** (which executes **setup -uninstall**).
readme.txt	
setup.bmp **setup16.bmp**	Bitmaps of the installation *splash* screen (**setup** is 256 color, **setup16** is 16 color).
setup.exe **_inst32i.ex_** **_setup.dll**	InstallShield-supplied executables.

setup.ini	InstallShield startup initialization file.
setup.ins	Compiled ObjectStore InstallShield application.
_setup.lib	InstallShield support library, including both InstallShield-supplied components and Object Design-supplied components. The components are listed in the next table.

_setup.lib components

CTL3D32.DLL **CTL3D32S.DLL** **_isres.dll** **corecomp.ini** **uninst.exe**	InstallShield-supplied components
odi.bmp	Object Design logo bitmap
ostore.bmp	ObjectStore logo bitmap
oscp437.dll	Resource DLL used by **os_setup.dll**
os_setup.dll	Support DLL called from ObjectStore InstallShield application

Components of Installation and Setup

The major ObjectStore-supplied components for the installation and setup are **SETUP.INS**, **OS_SETUP.DLL**, and **OSCP437.DLL**. **SETUP.INS** is the main InstallShield application program. **OS_SETUP.DLL** provides a wide variety of ObjectStore-specific support routines, and **OSCP437.DLL** is a resource DLL supplying English-language resources used by **OS_SETUP.DLL**. **OS_SETUP.DLL** and **OSCP437.DLL** are supplied in object form. **SETUP.INS** is derived from the source files **SETUP.RUL** and **OS_SETUP.H** that are supplied with this VAR kit.

OS_SETUP.DLL and **OSCP437.DLL**

Chapter 3, API Reference, describes the available entries in **OS_SETUP.DLL**. You should use these entries when modifying the ObjectStore installation or when creating your own installation and setup. The VAR kit might contain a text file with updates for the API documentation included in Chapter 3, API Reference.

The **OSCP437.DLL** is used internally by **OS_SETUP.DLL** and should not need to be changed unless the installation is to be performed in a language other than English. Contact Object Design Technical Support for assistance if a non-English system is required.

SETUP.INS: SETUP.RUL and OS_SETUP.H

SETUP.INS is the control program for ObjectStore **install**, **setup**, and **uninstall**. It was generated by the InstallShield compiler from the source files **SETUP.RUL** (that contains code) and **OS_SETUP.H** (that contains definitions for the various strings displayed by InstallShield) and for a variety of constants used by **SETUP.RUL**.

The VAR kit includes **SETUP.RUL** and **OS_SETUP.H** in source form. The descriptions of these files are based on the initial versions used with ObjectStore Release 5. Since changes might occur in subsequent versions, Object Design has included two files:

- **SETUP.50**
- **OS_SETUP.50**

Compare these files to **SETUP.RUL** and **OS_SETUP.H,** respectively, to see if any changes have occurred since ObjectStore Release 5.

See Chapter 4, SETUP.RUL Routines, for specific information about how these routines work.

Uninstall

Using **uninstall** results in the removal of all the various pieces of the ObjectStore installation from the system. This typically means the removal of

- **%OS_ROOTDIR%** and all its contents
- All ObjectStore-related entries from the registry
- On Windows 95, any ObjectStore-specific entries in **AUTOEXEC.BAT**

ObjectStore might also have installed updated Microsoft C++ run-time **DLL**s, but there is seldom a reason to remove these during **uninstall**.

Chapter 3
API Reference

The API descriptions that follow are based on the ObjectStore Release 5 version of **OS_SETUP.DLL**. The descriptions are organized alphabetically. Check the **README.TXT** file supplied with the VAR kit for the latest information about these APIs.

OS_BuildUnzipList

EXPORT BOOL OS_BuildUnzipList (char *InstallDir, char *InstallFrom, BOOL Upgrading, long Flags, char *ZipFileName, int iZipFile, long Mask)

Creates an internal list of files to be extracted from a given zip file.

InstallDir	Pathname of the target directory.
InstallFrom	Pathname of the directory containing zip files.
Upgrading	**TRUE** if upgrading an existing installation (the InstallShield script always sets this to **FALSE**).
Flags	Flags corresponding to the options being installed (as returned by **OS_GetInstallFlags**).
ZipFileName	Name of the zip file being unzipped.
iZipFile	Index of the zip file being unzipped (not currently used).
Mask	Mask associated with files in the zip file (as returned by **OS_GetZipFile**).
Return	**TRUE** if there are files to be extracted from this zip file, **FALSE** otherwise.

You must call this entry before attempting to extract files from a given zip file.

OS_CheckLogFile

EXPORT int OS_CheckLogFile (CHAR *FileName, int FileNameSize)

Attempts to determine if the Server log file has been initialized.

FileName	Pointer to a buffer to hold the Server log file pathname.
FileNameSize	Size of the Server log file pathname buffer.
Return	**0** if a log file is not found.
	1 if the log file is found in a file system file.
	2 if the log is being maintained in the RAWFS partition.

OS_CheckRAWFS

EXPORT int OS_CheckRAWFS ()

Determines if any RAWFS partitions exist.

Return	**TRUE** if a RAWFS partition is found in the registry, **FALSE** otherwise.

OS_CheckServerStartup

EXPORT int OS_CheckServerStartup ()

Determines if the ObjectStore Server is currently set for autostartup.

Return	**TRUE** if the Server is in autostart mode, **FALSE** otherwise.

OS_DecompressZipFile

EXPORT BOOL OS_DecompressZipFile (char *InstallDir, char *InstallFrom, BOOL Upgrading, long Flags, char *ZipFileName, int iZipFile, long Mask)

Decompresses an entire zip file. Not currently used for the InstallShield installation.This entry is obsolete.

InstallDir	Pathname of a target directory.
InstallFrom	Pathname of the directory containing zip files.
Upgrading	**TRUE** if upgrading an existing installation (the InstallShield script always sets this to **FALSE**).

Flags	Flags corresponding to the options being installed (as returned by **OS_GetInstallFlags**).
ZipFileName	Name of the zip file being unzipped.
iZipFile	Index of the zip file being unzipped (not currently used).
Mask	Mask associated with files in the zip file (as returned by **OS_GetZipFile**).
Return	**1** if files are extracted, **–1** otherwise.

OS_DeleteUninstRegKey

EXPORT BOOL OS_DeleteUninstRegKey (CHAR *szKeyPath, CHAR * szKey)

InstallShield adds an item to the list of *uninstallable* applications in the registry as one activity in the InstallShield ObjectStore installation process. However, ObjectStore cannot be uninstalled in the standard InstallShield mechanism, so Object Design deletes the key to avoid confusing the user. For this purpose, **szKeyPath** is

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall

and **szKey** is **ObjectStore**.

OS_DeleteUninstRegKey attempts to delete the key from **HKEY_LOCAL_MACHINE**.

Return	**TRUE** if the deletion succeeded, **FALSE** otherwise.

ObjectStore uses the InstallShield **VerUpdateFile** routine to install C++ run-time files. **VerUpdateFile** determines whether a reboot is required to complete installation. However, **VerUpdateFile** requires that the deinstall system be initialized (creating the **Uninstall** key) that **OS_DeleteUninstRegKey** later deletes.

OS_GetDefaultSrvrLogFile

EXPORT BOOL OS_GetDefaultSrvrLogFile(CHAR *pSvrLogFile)

Constructs a default pathname for the Server log file.

pSvrLogFile	Pointer to a 512-character buffer to hold the pathname. The pathname is based on the operating system (Win95 or WinNT), and on either the drive letter of the installation directory or the boot drive letter.

OS_GetDiskRequirement

EXPORT long OS_GetDiskRequirement (unsigned long Flags, char *InstallDir)

Returns an estimate of the amount of disk space required to install ObjectStore, based on the options represented by the **Flags** word.

Flags	As returned by **OS_GetInstallFlags**.
InstallDir	Pointer to a string containing the proposed installation directory.
Return	Estimated disk space requirements, in bytes.

OS_GetInstallFlags

XPORT long OS_GetInstallFlags (BOOL bLcClient, BOOL bLcServer, BOOL bLcDevo, BOOL bCompDBMS, BOOL bCompSingle, BOOL bCompExamples, BOOL bCompHTML, BOOL bCompPScript)

The entries that determine disk space requirements and lists of files to be installed use a flag, **long**, whose bits indicate which installation components should be copied, based on the various Boolean values passed in.

bLcClient	**TRUE** if a client is being installed, **FALSE** otherwise.
bLcServer	**TRUE** if a Server is being installed, **FALSE** otherwise.
bLcDevo	**TRUE** if this is a development installation, **FALSE** otherwise.
bCompDBMS	**TRUE** if ObjectStore itself is being installed, **FALSE** otherwise.
bCompSingle	**TRUE** if ObjectStore / Single is being installed, **FALSE** otherwise.
bCompExamples	**TRUE** if examples are being installed, **FALSE** otherwise.
bCompHTML	**TRUE** if HTML documentation is being installed, **FALSE** otherwise.
bCompPScript	**TRUE** if PostScript documentation is being installed, **FALSE** otherwise.
Return	**Flag** word with bits set as appropriate for files being installed.

OS_GetNextUnzip

EXPORT int OS_GetNextUnzip (char *UnzipName)

Gets the name of the next file to be unzipped.

UnzipName	Gets the name (relative to **OS_ROOTDIR**) of the next file to be extracted from the zip file. Back-slash characters are returned as forward slashes.
Return	**0** if there are no more files to be unzipped from this zip file, otherwise, the number of bytes of disk space required for this file.

OS_GetRegistration

EXPORT int OS_GetRegistration (CHAR *pRegisterType, CHAR *pRegEntry, int RegEntrySize, int LocalMachBool)

Returns ObjectStore *registration* registry information.

pRegisterType	Specific key value to be retrieved (see below).
pRegEntry	Pointer to a buffer to hold the value.
RegEntrySize	Size of the value buffer.
LocalMachBool	If **1**, the value is retrieved from **HKEY_LOCAL_MACHINE**; otherwise, the value is retrieved from **HKEY_CURRENT_USER**.
Return	Always returns **1**.

Values are retrieved from

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Registration

or

HKEY_CURRENT_USER\Software\Object Design Inc.\ObjectStore.4.0\Registration

Values stored in **HKEY_LOCAL_MACHINE** include all of these that are literal strings:

- Company
- Install From
- License Type
- ObjectStore Install Directory
- ObjectStore / Single Directory

- ObjectStore Examples Directory
- ObjectStore HTML Documentation Directory
- ObjectStore PostScript Documentation Directory
- Program Folder Name

Values stored in **HKEY_CURRENT_USER** consist only of

- User Name

OS_GetServerParameter

EXPORT int OS_GetServerParameter (const CHAR *Key, CHAR *Value)

Gets the current value of an ObjectStore Server parameter.

Key	Points to the parameter name.
Value	Points to a 512-character buffer to receive the value.
Return	**0** if the parameter is known, **–1** otherwise.

Server parameters are stored under the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

Note that a parameter can be known but have a null value. In this case, the **OS_GetServerParameter** returns **0** but the value buffer will contain a null string.

OS_GetZipFile

EXPORT int OS_GetZipFile(char *ArchiveName, int Which, long *Mask)

Gets the name of the next zip file and a mask, corresponding to the *flags* (see **OS_ZipLogFiles**) associated with the files in the zip file.

ArchiveName	Pointer to a 256-character buffer to hold the name of the zip file.
Which	Should start at zero and be incremented after each call to **OS_GetZipFile**.
Mask	Pointer to a **long**, which is set to the flags of the files stored in this zip file.
Return	**>= 0** if an archive corresponding to **Which** exists, **–1** otherwise.

OS_InitDLL

EXPORT int OS_InitDLL (HWND hWnd, CHAR *pAppPath)

Initialize **DLL** for execution	The first DLL entry called.
hWnd	Main installation window **InstallShieldGetWindowHandle** (**HWND_INSTALL**): used when displaying installation, setup, and error dialogs.
pAppPath	Directory containing **OSCP437.DLL** (or equivalent). The **OSCP***xxx***.DLL** should correspond to the OEM code page of the system running **setup**.
Return	Always returns **0**.

OS_InitLog

EXPORT int OS_InitLog (CHAR *Logfile, BOOL bSilent)

Initializes the ObjectStore Server and the Server log file.

Logfile	Pathname of the log file. This is ignored unless **bSilent** is **TRUE**, in which case it *must* be a file system path. It is not currently possible to silently initialize the Server if the Server log file is stored in the RAWFS.
bSilent	Set to **TRUE** to initialize the log without interacting with the user (used for a **Typical** installation).
Return	**TRUE** if the log file was initialized, **FALSE** if the user canceled from dialog or if initialization failed. Note that an error dialog is displayed by **OS_InitLog** if the initialization fails.

OS_IsUserAdmin

EXPORT BOOL OS_IsUserAdmin ()

Attempts to determine if the current user is an administrator for this system. For InstallShield users, Object Design recommends that you use the InstallShield **IS (USER_ADMINISTRATOR)** call instead. Some installation processes, especially those involving

the registry, might not be possible if the user is not running as administrator.

Return **TRUE** if the user is administrator, **FALSE** otherwise.

OS_NeedCheckpoint

EXPORT int OS_NeedCheckpoint ()

Determines if a Server log file or a RAWFS partition exists on the system. If they do, the user should be queried to see if they need to run checkpoint before proceeding.

Return **TRUE** if a checkpoint might be needed, **FALSE** otherwise.

OS_NeedCheckpoint examines the registry under

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

and looks for either a *Log File* entry or a *Partition0* entry. It does not check to see if either of these entries corresponds to actual data on the system.

OS_OStoreInstalled

EXPORT int OS_OStoreInstalled (CHAR *pOSRootDir, CHAR *pSingleDir, CHAR *pExamplesDir, CHAR *pHTMLDir, CHAR *pPScriptDir)

Attempts to determine if ObjectStore is currently installed on the system. If it is, it attempts to determine pathnames for directories being used. This entry should be called very early in the setup process. All the directories involved are obtained from the corresponding registry entries found in **pOSRootDir**, which points to a 512-byte buffer that holds the current **OS_ROOTDIR**.

pSingleDir Points to a 512-byte buffer that holds the current single directory (the directory **binsngl**).

pExamplesDir Points to a 512-byte buffer that holds the current examples directory (the directory above **examples**).

pHTMLDir Points to a 512-byte buffer that holds the current HTML documentation directory (the directory above **doc**).

pPScriptDir Points to a 512-byte buffer that holds the current PostScript documentation directory (the directory above **doc\pscript**).

Return Additive combinations of

1: If the **OS_ROOTDIR** directory is found

2: If the Single directory is found

4: If the Examples directory is found

8: If the HTML documentation directory is found

16: If the PostScript documentation directory is found

Each of the return values depends on the corresponding directory's being named in the registry data and on the existence of the named directory in the system.

An internal default installation directory pathname might be set based on what directories are found. At any given time, only one installation directory is permitted; therefore, if multiple directories are encountered, the internal setting corresponding to the lower return value (that is, **OS_ROOTDIR** is preferred) is used.

OS_OStoreRunning

EXPORT BOOL OS_OStoreRunning ()

Attempts to determine if ObjectStore is currently running. Looks for any services (Server or Cache Manager) that might be running and for ObjectStore DLLs that might be active.

Return **TRUE** if it appears that ObjectStore is running, **FALSE** otherwise.

OS_rename_dir

EXPORT BOOL OS_rename_dir (char *Path, char *NewPath)

Renames a directory (or file).

Path Original name of file or directory.

NewPath New name.

Return **TRUE** if the rename succeeds, **FALSE** otherwise.

OS_RenameToLongFiles

EXPORT BOOL OS_RenameToLongFiles (char *InstallDir)

Renames files to their correct long (non-8.3) names.

InstallDir **OS_ROOTDIR** of where files have been installed.

Return Always returns **TRUE**.

OS_SetInstallDirectories

EXPORT int OS_SetInstallDirectories (CHAR *pOSRootDir, CHAR *pSingleDir, CHAR *pExamplesDir, CHAR *pHTMLDir, CHAR *pPScriptDir)

Sets registry values corresponding to the directory names passed in. All names passed in should be the same, but directories corresponding to components that have not been installed should be set to **""**. Do *not* pass in NULL pointers.

pOSRootDir	Points to **OS_ROOTDIR** value.
pSingleDir	Points to path of parent of **binsngl** directory.
pExamplesDir	Points to path of parent of **examples** directory.
pHTMLDir	Points to path of parent of **doc** directory.
pPScriptDir	Points to path of parent of **doc\pscript** directory.
Return	Always returns **0**.

OS_SetRegistration

EXPORT int OS_SetRegistration (CHAR *pRegisterType, CHAR *pRegEntry, int LocalMachBool)

Sets the ObjectStore registration registry information.

pRegisterType	Specific key value to be set.
pRegEntry	Pointer value string.
LocalMachBool	If **1**, the value is set in **HKEY_LOCAL_MACHINE**; otherwise, the value is set in **HKEY_CURRENT_USER**.
Return	Always returns **1**.

See **OS_GetRegistration** for details of registration keys used by ObjectStore.

OS_SetServerParameter

EXPORT int OS_SetServerParameter (const CHAR *Key, CHAR *Value)

Sets an ObjectStore Server parameter value.

Key	Points to the parameter name.
Value	Points to the new value.

Return **0** if the parameter is known, **–1** otherwise.

Server parameters are stored under the registry key

HKEY_LOCAL_MACHINE\SOFTWARE\Object Design Inc.\ObjectStore.4.0\Server

OS_SetServerParams

EXPORT int OS_SetServerParams ()

Runs the ObjectStore Server Parameters dialog and sets any parameters selected by the user.

Return **0** if the user selected **Cancel**, **1** if the user selected **OK**.

OS_SetupRawfs

EXPORT int OS_SetupRawfs (int *bPartChanged)

Runs the ObjectStore RAWFS partition dialogs and sets registry values appropriately.

bPartChanged When the dialog returns, the installation checks to see if the partitions have changed. If so, the utility **OR**s a changed bit into the **int** pointed to by this argument.

Return **0** if the user selected **Cancel**, **1** if the user selected **OK**.

OS_ShutdownOStore

EXPORT int OS_ShutdownOStore ()

Attempts to shut down any running ObjectStore services (Server or Cache Manager).

Return **TRUE** if all processes appear to have been stopped, **FALSE** otherwise.

OS_status_update_bar_num

EXPORT int OS_status_update_bar_num (char *CurrFile)

Gets the total disk space used by files installed to this point.

CurrFile **OS_ROOTDIR**-relative name of the next file to be installed.

Return	Number of bytes installed before **curr_file**.

OS_TermDLL

EXPORT int OS_TermDLL ()

Terminate **DLL**	Frees resources after execution. The last **DLL** entry called.
Return	Always returns **0**.

OS_Uninstall

EXPORT int OS_Uninstall (char *OSRootDir, BOOL bUninstallRAWFS)

Removes ObjectStore from the system; attempts to remove all vestiges of ObjectStore.

OSRootDir	Current **OS_ROOTDIR** value.
bUninstallRAWFS	**TRUE** to remove RAWFS files in addition to ObjectStore executables.
Return	Always returns **TRUE**.

OS_update_environment

EXPORT BOOL OS_update_environment (char *OSRootDir, BOOL bRuntimeOnly, char *OSSchemaDir)

Sets various environment values.

OSRootDir	Pointer to **OS_ROOTDIR** value.
bRuntimeOnly	**TRUE** if this installation does not include development support.
OSSchemaDir	Pointer to a schema directory, if this is a client-only development installation.
Return	**TRUE** if the environment was modified, **FALSE** otherwise.

Environment values that can be set or modified by this call include

OS_ROOTDIR	Always set.
OS_LIBDIR	Set if **OSSchemaDir** is nonnull.
OS_TMPDIR	Always set.
PATH	Always set.

INCLUDE	Set if **bRuntimeOnly** is **FALSE**
LIB	Set if **bRuntimeOnly** is **FALSE**

For Windows NT, settings are made in the registry. For Windows 95, settings are made in **OS_AUTO.BAT**, which is called from **AUTOEXEC.BAT**.

OS_UpdateStartup

EXPORT int OS_UpdateStartup (BOOL StartServer, CHAR *InstallDir, BOOL StartNow)

Updates Server autostartup values in the registry and in system services databases.

StartServer	**TRUE** to have Server start automatically, **FALSE** otherwise.
InstallDir	Pointer to **OS_ROOTDIR** string.
StartNow	**TRUE** if user should be queried about starting the Server immediately (assuming that **StartServer** is also **TRUE**), **FALSE** to skip the query and delay starting the Server. It might be appropriate to delay startup if some elements of the installation (notably **MSVCRT.DLL**) are not to be installed until after the system reboots.
Return	Always returns **0**.

OS_UnzipOneFile

EXPORT INT OS_UnzipOneFile (char *UnzipName)

Unzips one file.

UnzipName	Name of the file to be extracted from the current zip file.
Return	Return **0** if no errors are encountered, nonzero otherwise.

Unzip error codes are defined in **os_setup.h**, with names beginning with **PK_** and values ranging from **1** to **51**.

OS_ZipLogFiles

EXPORT BOOL OS_ZipLogFiles (char *InstallDir)

Opens log files **OSUNZIP.LOG** and **OSUNZIP.ERR** in **InstallDir** and redirects **stdout** and **stderr** to those files. You can use this entry to direct unzip output to these files, but the InstallShield installation does not currently do this.

InstallDir	Directory into which the log files will be placed.
Return	Currently always returns **1**.

Chapter 4
SETUP.RUL Routines

This chapter provides overview and detailed information about the **SETUP.RUL** routines.

Overview of SETUP.50 (SETUP.RUL)

The paragraphs that follow describe each of the routines supplied in **SETUP.50**.

Two areas of that code (the main program, and the code that unzips files and updates the progress display) are complex. They are documented here in detail.

Main Program

The main program starts at the label **start:** following the keyword **program**. The code in the main program is somewhat complex because of the interactions between **install** and **setup**, and because it is serving as the main program for three different applications: **install**, **setup**, and **uninstall**.

The main program starts out with a call to the **Initialize** routine. **Initialize** performs a number of functions, such as loading **DLL**s, displaying bitmaps, and checking for a previous installation of ObjectStore. It also checks the command line arguments **-install** and **-uninstall**, which force those behaviors. After dealing with the possibility that ObjectStore might be running, the main program sets a **RunningInstall** flag, which is set to TRUE if no previous ObjectStore installation was detected or if the **SETUP.EXE** program was invoked with the **-install** command line argument.

Execution of the code starting at the **Reinstall:** is controlled by a number of Boolean variables:

- **RunningInstall**
- **RunningSetup**
- **AutoSetup**

If **RunningInstall** is set, execution proceeds directly to the installation welcome screen (the call to **SdWelcome**) and continues with the call to **RunInstall**. **RunInstall** returns a value indicating whether **setup** needs to be run, if the user selected **Custom install**. If the user selected **Typical install**, the **AutoSetup** flag is set if the DBMS server was installed and needs to be set up automatically.

The next block of code, starting with **if (RunningSetup) then**, takes different paths depending on the flag settings. The settings and their results are as follows:

RunningSetup	RunningInstall	AutoSetup	*Action*
TRUE	FALSE	Don't Care	Run **SetupOptions** to see what to do next, then restart at the **Reinstall:** label.
TRUE	TRUE	Don't Care	Run **OSSetup**, **OSFinish**, and then done.
FALSE	TRUE	TRUE	Reinitialize the log, if necessary.
FALSE	TRUE	FALSE	Run **OSFinish**, and then done.
FALSE	FALSE	Don't Care	Should not happen.

After this block of code, and after the **Terminate** routine has been invoked, the code checks the **BATCH_INSTALL** flag that is set if a conflict was encountered when attempting to install Microsoft C++ run-time files. If so, the user can choose to reboot immediately or to reboot later.

Unzipping Files and Updating the Progress Display

The second complex section of code in **SETUP.50** occurs in the routine **DecompressFiles**. This routine manages both decompressing (unzipping) files and updating the progress bar with percent complete and the name of the current file being decompressed.

DecompressFiles calls **OS_GetInstallFlags**, which returns a bit mask corresponding to the various types of files to be extracted. It then enters the outer loop and calls **OS_GetZipFile**, which returns the name of each available zip file, along with another mask corresponding to the types of files contained in that zip file. If there is a match between the two masks, **DecompressFiles** then calls **OS_BuildUnzipList** to initialize the list of files to be extracted from the current zip file, and enters an inner loop.

The inner loop processes one file at a time. It calls **OS_GetNextUnzip** to get the name and size of the next file to be decompressed. It then updates the progress display with the name of the file being unzipped. Note that the progress bar is not updated until the next time through the loop. This means that, at any given time, the display includes the name of the file being decompressed and the percent complete up to, but not including,

that file. After the display is updated, the size of the current file is added to **TotalCopied** and the file is decompressed.

(TotalCopied * 100) / TotalSpace

TotalSpace is calculated in the **Components** routine and is based on the licensing and installation options.

SETUP.RUL Descriptions

The paragraphs that follow describe the **SETUP.RUL** routines.

function CheckOStoreRunning ()

Returns **TRUE** if it appears thatObjectStore is currently running, **FALSE** otherwise.

function CheckUpgrade ()

Determines if the installation is an upgrade and, if so, queries the user about checkpointing the Server. Returns **TRUE** if the installation should continue, **FALSE** if the installation should be terminated.

function Components ()

Builds and displays the component options dialog and sets flags, and space requirements, based on results. Note that the component options dialog is more complex than most since the options presented to the user depend on the licensing option selected.

function ConfirmInstall ()

Generates and displays the installation confirmation dialog, customized by all the user's selections. Returns either **NEXT** or **BACK** depending on the user's button selection.

function CopyFiles (szInstallFrom)

Calls various routines to copy individual files. Supplies estimated file size for each file being copied. This function deals with files (such as those used by the installation) that are distributed outside the zip files.

function CopyOneFile (szFile, nFileSize)

Copies a single nonzipped file while updating the progress display.

function CreateOSProgramFolder ()

Creates and / or updates a program manager folder for ObjectStore.

function CreateRegAndDirs ()

Updates registry information under the registration key. See "Registration data in the registry", Chapter 2, ObjectStore Windows Installation System, for details.

function DecompressFiles (szInstallFrom)

Main routine for decompressing zip files.

function DeleteFiles ()

Called if the user selected the **delete before installing** option. It deletes *all* subdirectories under the selected **OS_ROOTDIR**. (It does not delete any files that might reside in the **OS_ROOTDIR** itself.)

function DeleteProgramFolderItem (szItemName)

Deletes a single item from a program manager folder.

function DeleteOSFolder ()

Deletes various ObjectStore items from a program manager folder and then attempts to delete the folder itself. This code is not completely effective as there is a conflict between InstallShield and Windows NT 4.0 (and, perhaps, Windows 95).

function DirExistsOptions (szExistingDir)

Displays a dialog asking the user what to do about a previously existing **OS_ROOTDIR** (overwrite, rename, or delete). Returns either **NEXT** or **BACK**, depending on the user's button selection.

function DisplayCopyFile (szFileStr, nPercent)

Updates the progress display status window for files being copied.

function DisplayUnzipFile (szDestLower, szTargetStr, nPercent)

Updates the progress display status window for files being unzipped.

function FixDisplayPath (szPathStr, szFixedStr)

Utility routine used when updating the progress display. This routine converts forward slashes to back slashes and, if necessary,

shortens the string being displayed so it will fit in the space available.

function GetOSLibDir (WhichButton)

Displays a dialog querying the user for an **OS_LIBDIR**. This function is used when you are installing a client-only development system because the various ObjectStore databases needed during development are not available on the local system. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function GetRenamePathname (oldpath, newPath, szSeqNum)

Constructs a pathname used in the **DirExistsOptions** routine as the target if the user chooses to rename an existing **OS_ROOTDIR**.

function HandleUnzipError (nErrorCode, szFileName)

Displays error messages for problems encountered during unzip operations, and allows the user to do one of the following:

- Ignore this error
- Ignore all unzip errors
- Abandon the installation attempt

function Initialize ()

Performs common initialization tasks.

- Checks operating system type and version
- Checks whether user is a system administrator
- Loads **OS_SETUP.DLL** and initializes it
- Initializes main display window
- Checks for **-install** and **-uninstall** arguments
- Checks to see if ObjectStore is already installed
- Sets up various strings, dialog titles, and internal lists

function InitializeAutoStart ()

Displays a dialog asking if the user wants the Server to start automatically. Note that this routine merely sets the **DoStartup** flag, which is then used later in the **OSFinish** routine.

function InitializeInstall ()

Performs basic initialization steps needed specifically for **install**.

function InitializeLog ()

Displays a dialog asking whether the user wants to initialize or reinitialize the Server log. If so, invokes the **OS_SETUP.DLL** routine to do the initialization. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function InstallationOptions ()

Displays the installation options (typical or custom) dialog. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function InstallFiles ()

Performs all file installations, which include

- Copying **setup**-related files to **OS_ROOTDIR**
- Decompressing appropriate zip files in **OS_ROOTDIR** subdirectories
- Installing updated Microsoft C++ run-time files
- Installing the ObjectStore MFC Wizard file

function LicenseOptions ()

Displays the license options dialog and sets flags based on user selection. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function OSFinish ()

Checks to see whether to start the Server automatically (if the log file is OK) and calls **OS_UpdateStartup**, which asks the user if the Server should be started immediately. Also displays a dialog allowing the user to choose to view the **README.TXT** file. Always returns **NEXT**.

function OSSetup (bAllowBack)

Steps through the **setup** dialogs: **Server Parameters**, **RAWFS Setup**, **Server Initialization**, and **AutoStart Initialization**.

function OSUninstall ()

Asks the user for **uninstall** confirmation and then performs the sequence of steps needed to remove ObjectStore from the system.

function ProcessInstall ()

Calls other routines to perform the basic steps required to actually install ObjectStore.

function ProcessSpecialFiles (szBinDir)

Attempts to install the ObjectStore MFC Wizard file in the Microsoft **MSDevDir** hierarchy.

function ProgramFolder (WhichButton)

Displays a dialog allowing the user to select a program manager folder name for ObjectStore use. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function QueryShutdownServices ()

Queries the user about shutting down currently running ObjectStore services. Returns **TRUE** if services should be shut down.

function RecheckInitializeLog ()

Checks to see if the Server log file needs to be initialized or reinitialized. If so, queries the user and, if necessary, invokes **OS_SETUP.DLL** to initialize the log.

function Registration ()

Displays the registration information dialog. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function RunInstall ()

This is the main routine for **install**. Steps through the **install**-specific dialogs (obtaining registration data, license options, installation options, component options, and so on), displays a confirmation dialog, invokes **ProcessInstall** to install files, and updates environment settings.

function RunSetupOptions ()

Displays the initial **setup** dialog, allowing the user to choose between **Install**, **Reinstall**, **Setup**, and **Uninstall**. In case of **Setup** and **Uninstall**, invokes the relevant routines immediately. Returns **TRUE** if the user selects the **Reinstall** option, **FALSE** otherwise.

function SetServerParams ()

Displays a dialog asking whether the user wants to set Server parameters. If so, invokes the **OS_SETUP.DLL** routine to display the Server parameters dialog. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function SetupRawfs ()

Displays a dialog asking whether the user wants to set up or modify the RAWFS configuration. If so, invokes the **OS_SETUP.DLL** routine to display the RAWFS dialog. Returns either **NEXT** or **BACK**, depending on the user's button selection.

function SourcePath ()

Attempts to find a usable installation source directory. **install** is usually invoked from a directory that contains the complete ObjectStore installation environment. But the user can select **reinstall** after invoking **setup.exe** in the **OS_ROOTDIR**. In this case files such as the **.zip** files are not immediately available. **SourcePath** looks in the registration data to see if there is an original source path. If there is, it verifies that the **Sourcepath** exists and contains **.zip** files.

function Terminate ()

Performs common termination tasks.

- Deletes extraneous registry key
- Calls **OS_SETUP.DLL** termination routine
- Unloads **OS_SETUP.DLL** and **OSCP437.DLL**

function UninstallSpecialFiles ()

Attempts to delete the ObjectStore MFC Wizard file, which might have been installed in the Microsoft C++ hierarchy.

function UpdateMSVCRT (szInstallFrom, szFileName)

Attempts to update **MSVCRT.DLL** and does so in such a way that a failure to update can be corrected at a subsequent system reboot.

Index

I

L

O

P

R

S