Internet Component Download

Last updated: 5/30/96

DRAFT

This document provides a description of the mechanism for downloading and installing code for COM Objects (components) using the Microsoft Active Internet Platform (Sweeper). This mechanism is used internally by the Microsoft Internet Explorer for downloading OLE Controls inserted in HTML pages.

Executive Summary

- Internet Component Download is a mechanism for downloading and installing code for COM objects[1].
- Details are presented for how OLE Control authors should package their objects to be downloaded and installed automatically.
- A new system API CoGetClassObjectFromURL is presented for downloading COM components. Other "safe code-download" needs can be met using the lower-level HREF="wintrust.doc service, or possibly using a high-level "Setup" OLE Control. Future releases may expose a more sophisticated component download interface.
- Internet Component Download makes use of an Internet Search Path to search various "Object Stores" for download-able code.
- Internet Component Download installs code in a permanent store. This document details a migration strategy for future releases to convert this store into a cache that discards unpopular components.

[1] **Note:** Internet Component Download as specified will *not* download anything other than OLE Objects. This document does not list steps needed to download/certify other entities. For other code-download needs see documentation for HREF="wintrust.doc.

1Introduction

Internet Component Download is a system service for downloading, certificate checking, and installing COM component code from the Internet. This service is used by applications (e.g. web browsers) to automatically download and install COM Objects from code repositories on the Internet. This document explains how code authors should prepare their components for automatic download. It then describes the interface for the component download mechanism, and finally provides some additional implementation details, including a description of the Internet Search Path service which allows searching for downloadable code from a series of "Object Stores".

Component Download is used within Microsoft Internet Explorer in order to automatically download OLE Controls inserted inside HTML pages using the `<OBJECT>` element in HTML[2]. The mechanism for downloading components is exposed in an API that may be used in various other OLE containers. OLE Control developers should follow the guidelines outlined below to package their controls so that they can be downloaded automatically by any container that uses the Component Download mechanism.

2Packaging component code for automatic download

ISVs and authors of COM Objects for the internet should package their implementations so that they may be downloaded automatically by web browsers such as the Microsoft Internet Explorer. Such objects will be downloaded, for instance, when parsing the OBJECT tag in HTML[3]. For details, see the HREF="WD-OBJECT.html.

2.1Interpreting the "CODEBASE" URL

The "CODEBASE" attribute in an OBJECT tag contains a URL pointing to the implementation of a given COM object. This URL is of critical importance for component download, because it must specify all files necessary to implement a particular COM object. HTML authors can author "CODEBASE" URL to point to one of three file types. Component developers should choose one of the packaging schemes below for their COM Objects:

1. A single **PE** (portable executable, e.g. an .OCX, a .DLL, or a .EXE): This single executable is downloaded, installed, and registered in one fell swoop. This is the simplest way to package a single-file OLE control, but (a) it will not use file compression, (b) it will not be platform independent except with HTTP.
2. A **.CAB** (cabinet) file: This file contains one or more files, all of which are downloaded together in a compressed cabinet.[4] Exactly one file in the cabinet is an .INF file providing further installation information. This .INF file may refer to files in the .CAB as well to files at other URLs. This mechanism requires authoring of a .INF and packaging of a .CAB file, but in return it provides file compression. It will not be platform independent, however, except with HTTP format negotiation.
3. A stand-alone **.INF** file: This file specifies various files that need to be downloaded and setup for the OCX to run. The syntax of the .INF file allows (a) URLs pointing to files to download, and (b) platform independence (by enumerating files for various platforms). This mechanism provides platform independence for non-HTTP servers.

[2] In future releases code for Document Object components will likewise be downloaded and installed automatically.
[3] Note: The <OBJECT> tag used to be called the <INSERT> tag. This change was decided on by the W3C on 2/13.

[4] Care must be taken so that the cabinet file contains only those files that must *necessarily* be downloaded (e.g. the OCX executable itself). Any additional helper DLLs (e.g. MFC) may have already been installed and if so should not be bundled into the cabinet.

2.1.1 Registry settings and self-extracting .exes

It's recommended to use self-registering code for Internet Component Download, because the .INF format used by Internet Component Download (see below) does *not* provide syntax for changing registry information (for security reasons). For .DLLs, EXEs a"OleSelfRegister" in the Version resource, Internet Component Download will try to run self-registration. For .DLLs, this means loading the .DLL library and calling the DllRegisterServer entry point, if available. For .EXEs, this means *running* the .EXE with the runtime parameter of "/RegServer". This ensures that COM Objects implemented as local servers (e.g. winword.exe) are registered correctly. If an object is not marked as "OleSelfRegister" but registration is necessary, or if it is desired to over-ride the "OleSelfRegister" flag, one can add the following to an .INF file (see .INF setup-script format below):

```
[foo.ocx]
RegisterServer=no ; don't register even if marked OleSelfRegister
      or
RegsiterServer=yes ; register this even if not marked OleSelfRegister. This is the typical workaround for getting old
                   ; controls to register
```

Code that is downloaded via Internet Component Download may be a self-extracting .EXE because Internet Component Download ignores the "OleSelfRegister" flag if the main URL for code download points directly at a .EXE file. In this case it is assumed that this is a self-registering .EXE, and this enables self-extracting .EXEs to work correctly as long as they ignore the "/regsvr" command-line parameter. Supporting self-extracting .EXEs enables very complex setup mechanisms to be launched automatically via Internet Component Download. However, if a self-extracting .EXE is called via this mechanism, then any components that it installs will not be automatically tracked by Internet Component Download (see Appendix on ModuleUsage section in registry). Such components are permanently installed and are not marked by Internet Component Download for future cleanup.

2.1.2 Including version number in the "CODEBASE" URL

Besides the actual address of code, the "CODEBASE" URL may also include an optional version number using the following syntax: "CODEBASE=http://www.foo.com/bar.ocx#Version=a,b,c,d". The Internet Component Download mechanism will download and install the file only if the specified version number is *more recent* than any existing version of the same file currently installed in the system. (see Appendix on registry details for more information). If a version number is not specified with a file, it is assumed that any version installed on the system is recent enough. [5]

If the version number specified in the CODEBASE attribute is "-1,-1,-1,-1", then Internet Component Download will *always* try to download the *latest* version of the desired component. Note that this can be a costly effort involving many network transactions, especially if the Internet Search Path is searched for newest versions of an object (see below for details on Internet Search Path). Note also that because of the Internet Search Path, it is possible for the Component Download service to try to download code in the absence of a CODEBASE attribute. In fact, if the CODEBASE attribute is the URL fragment "#Version=-1,-1,-1,-1", then there is no specific location to download code from, but the Internet Search Path will still be searched to find the *latest* version of an object. [6]

2.1.3 Platform independence and HTTP

When code to downloaded is on an HTTP server, the HTTP Accept header MIME request type may be used to specify which platform the code is to run on, thus allowing platform independence of the "CODEBASE" URL.

[5] Note that Internet Component Download makes the assumption that a *newer* version of an object with the same ClassID is *always backwards compatible* with previous versions. A newer version of an object may be used to replace older versions without worry of losing functionality. If a newer version of an object is *not* backwards compatible with previous versions, it is advised to assign a new ClassID to the incompatible implementations in order to avoid one overwriting the other resulting in loss of functionality.

[6] If Internet Search Path is used to find the *latest* version of an object, Component Download will search the path, querying servers, and it will use the *first matching component* that is a newer version than the existing version installed on the system (if any). For more details see the below documentation on Internet Search Path.

The following MIME types will be used to describe PEs (portable executables - .EXE, .DLL, .OCX), cabinet files (.CAB), and setup scripts (.INF):[7]

File description	MIME Type
PE (portable executable) - .EXE, .DLL, .OCX	application/x-pe-%opersys%-%cpu%
Cabinet files - .CAB	application/x-cabinet-%opersys%-%cpu%
Setup scripts - .INF (platform independent)	application/x-setupscript

%opersys% and %cpu% above will specify the operating system and CPU for the desired platform downloaded components will be executed on. For example, the MIME type for a Win32 cabinet file running on an Intel ® x86-architecture processor would be application/x-cabinet-win32-x86.

The following are valid values for %opersys% and %cpu%:

Valid values for %opersys%	Meaning
win32	32-bit Windows ® operating systems (Windows95 or Windows NT)
mac	Macintosh ® operating system
<other>	will be defined as necessary

Valid values for %cpu%	Meaning
x86	Intel ® x86 family of processors
ppc	Motorola ® PowerPC architecture
mips	MIPS ® architecture processors
alpha	DEC ® Alpha architecture

When the code is on a non-HTTP server (e.g. at a local LAN location), a .INF file can be used to achieve platform independence by specifying different URLs for files to be downloaded for different platforms. (see the section below on platform independence in .INF files)

2.2.CAB format

The .CAB format used for Internet Component Download is a non-proprietary format based on Lempel-Ziv compression. The Microsoft Internet SDK includes a free tool called “diantz.exe” that will package cabinet files into this non-proprietary format. There no specification of this .CAB format publicly available, although such a specification will be distributed as soon as possible.

2.2.1Use of the DIANTZ.EXE tool for creating .CAB cabinet files

The DIANTZ.EXE tool takes a .DDF “directive file” specifying which files to combine into a cabinet. The syntax for using this tool from a command line is:

```
DIANTZ.EXE /f <directive file.ddf>
```

The example directive file below, CIRC3Z.DDF, would be used for creating a cabinet file containing two files - circ3.inf and circ3.ocx. It should be straightforward to add to this list of files…

```
; DIAMOND directive file for CIRC3.OCX+CIRC3.INF
.OPTION EXPLICIT                        ; Generate errors on variable typos
.Set CabinetNameTemplate=CIRC3Z.CAB
;** The files specified below are stored, compressed, in cabinet files
.Set Cabinet=on
.Set Compress=on
```

[7] Note: The MIME scheme described here is *temporary*. Obviously this scheme results in too many MIME types. Eventually, MIME attributes will be used for the purpose of describing platform-dependent code (e.g. application/x-cabinet; os=win32 cpu=x86). Until more HTTP servers support such requests, the temporary scheme described above should suffice.

```
circ3.INF
circ3.OCX
```

Note: it is possible to use the "code-signing" utilities to sign entire cabinet files using a digital certificate. However, in order to do this, it is necessary to add the following linesto the .DDF file before the list of files for inclusion in the cabinet.

```
:Reserve space for PKS#7 Code Signature
.Set ReservePerCabinetSize=2048
```

If a cabinet file is signed, it is assumed that *every* file inside the cabinet is trusted, including .INF and .INI files. This has two advantages:

1. It is now possible to include powerful .INFs inside a trusted cabinet
2. By signing an entire cabinet the time for verifying digital certificates can be sped up due to the cabinet compression

2.3.INF setup-script format

Here is a sample .INF file that demonstrates the syntax that is understood by the Component Download service. **Note: Only the .INF syntax below may be used to write setup scripts for Internet Component Download. Due to security reasons the system standard (SetupX) .INF setup is *not* called to install components with setup scripts, and instead the limited INF syntax below is the only legal format for Internet Component Download.** See Future Directions below for plans for eventually supporting other .INF formats.

```
;Sample INF file for CIRC3.OCX
[Add.Code]
circ3.ocx=circ3.ocx
random.dll=random.dll
mfc40.dll=mfc40.dll
foo.ocx=foo.ocx

[circ3.ocx]
; lines below specify that the specified circ3.ocx (clsid, version) needs to be installed on
; the system. If doesn't exist already, can be downloaded from the given location (a .CAB)
; note: if "thiscab" is specified instead of the file location, it is assumed that the
; desired file is present in the same .CAB cabinet that the INF originated from
; otherwise, if the location pointed to is a different .CAB, the new cabinet is also downloaded and
; unpacked in order to extract the desired file
file=http://www.code.com/circ3/circ3.cab
clsid={9DBAFCCF-592F-101B-85CE-00608CEC297B}
FileVersion=1,0,0,143

[random.dll]
; lines below specify that the random.dll needs to be installed in the system
; if this doesn't exist already, it can be downloaded from the given location.
file=http:// www.code.com/circ3/random.dll
; Note that the FileVersion is option, and it may also be left empty, meaning that any version is ok.
FileVersion=
DestDir=10


; DestDir can be set to 10 or 11 ( LDID_WIN or LDID_SYS by INF convention)
; this places files in \windows or \windows\system, respectively
; if no dest dir specified (typical case), code is installed in the fixed occache directory.

[mfc40.dll]
; leaving the file location empty specifies that the installation
; needs mfc40 (version 4,0,0,5), but it should not be downloaded.
; if this file is not already present on the client machine, component download fails
file=
FileVersion=4,0,0,5

[foo.ocx]
; leaving the file location empty specifies that the installation
; needs the specified foo.ocx (clsid, version), but it should not be downloaded.
; if this file is not already present on the client machine, component download fails
file=
clsid={DEADBEEF-592F-101B-85CE-00608CEC297B}
```

```
FileVersion=1,0,0,143
```

2.3.1 Platform independence in .INF files

It is possible to create platform-independent setup scripts that pull files from different locations depending on the desired platform. Internet Component Download .INF files will use a scheme similar to the one described above under “Platform Independence and HTTP”. Specifically, a sample platform-independent .INF file would include a text such as the following:

```
[circ3.ocx]
; lines below specify that the specified circ3.ocx (clsid, version) needs to be installed on
; the system. If doesn't exist already, can be downloaded from the given location (a .CAB)
file-win32-x86=file://products/release/circ3/x86/circ3.cab
file-win32-mips=file://products/release/circ3/mips/circ3.cab
file-mac-ppc=ignore
      ; the 'ignore' keyword means that this file is not needed for this platform

clsid={9DBAFCCF-592F-101B-85CE-00608CEC297B}
FileVersion=1,0,0,143
```

Thus the “file=” syntax used in the .INF file is expanded to “file-%opersys%-%cpu =”, allowing the .INF file to specify multiple locations where various platform-dependent modules can be found and downloaded. See the section above for valid values for %opersys% and %cpu%.

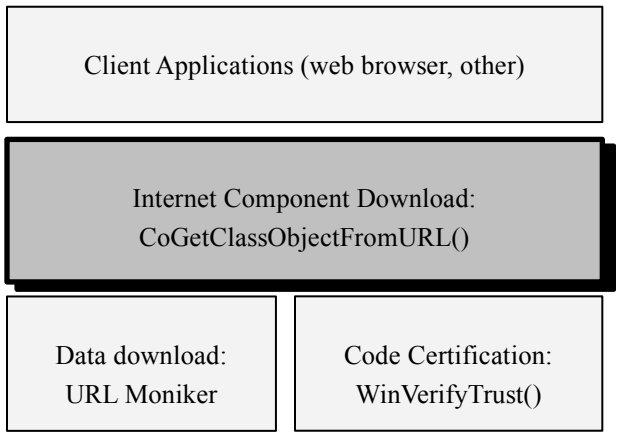
3 The Internet Component Download interface

The Internet Component Download service is exposed via a single function, CoGetClassObjectFromURL(). This system function is called by an application that wishes to download, verify, and install code for an OLE component. The function is used in the implementation of Microsoft ® Internet Explorer. The implementation uses HREF="urlmon.doc to asynchronously download code, and it uses the HREF="wintrust.doc service to verify validity of the code.

Related Documents	Filename
HREF="urlmon.doc	urlmon.doc
HREF="asyncmon.doc	asyncmon.doc
HREF="wintrust.doc	wintrust.doc

3.1 Architecture

The diagram below shows the implementation architecture for the Internet Component Download mechanism and its relation to other system services:



3.2 Technical Details

This section describes technical details of the Internet Component Download API used by applications (e.g. web browsers) to download and install COM Object code.

3.2.1 CoGetClassObjectFromURL

```
STDAPI CoGetClassObjectFromURL ( [in] REFCLSID rclsid, [in] LPCWSTR szCodeURL, [in] DWORD dwFileVersionMS,
                [in] DWORD dwFileVersionLS, [in] LPCWSTR szContentTYPE, [in] LPBINDCTX pBindCtx,
                [in] DWORD dwClsContext, [in] LPVOID pvReserved, [in] REFIID riid, [out] VOID **ppv );
```

This function will return a factory object for a given rclsid. If no CLSID is specified (CLSID_NULL), this function will choose the appropriate CLSID for interpreting the Internet MIME type specified in szContentType. If the desired object is installed on the system, it is instantiated. Otherwise, the necessary code is downloaded and installed from the location specified in szCodeURL or from an Object Store on the Internet Search Path (see below).

This "download and install" process involves the following steps:

1. Downloading the necessary file(s) (.CAB, .INF, or executable) using URL Moniker(s).
2. Call WinVerifyTrust to ensure that all downloaded files are safe to install.
3. Complete self-registration of all COM components[8]
4. Add registry entries to keep track of downloaded code (see Appendix on Registry Details)
5. Call CoGetClassObject for the desired rclsid.

CoGetClassObjectFromURL accepts the following arguments:

[8] Internet Component Download accomplishes self-registration using the /regserver command-line argument for .EXE files, and DLLRegisterServer() for other executables (.DLL, .OCX)

Argument	Type	Description
rclsid	REFCLSID	CLSID of the COM object that needs to be installed. If value is CLSID_NULL, then szContentType is used to determine the CLSID.
szCodeURL	LPCWSTR	URL pointing to the code for the COM object. This may point to an executable, to an .INF file, or to a .CAB file (see below for details). If this value is NULL, then Internet Component Download will still attempt to download the desired code from an Object Store on the Internet Search Path.
dwFileVersionMS	DWORD	Major version number for the object that needs to be installed. If this value and the next are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired, which means that Internet Component Download will *always* attempt to download new code.
dwFileVersionLS	DWORD	Minor version number for the object that needs to be installed. If this value and the previous one are both 0xFFFFFFFF, then it is assumed that the latest version of the code is always desired, which means that Internet Component Download will *always* attempt to download new code.
szContentType	LPCWSTR	MIME type that needs to be understood by the installed COM object. If rclsid is CLSID_NULL, this string is used to determine the CLSID of the object that must be installed. Note: this parameter is only useful when trying to download a viewer for a particular media type, if the MIME type of the media is known but the CLSID is not.
pBindCtx	LPBINDCTX	A bind context to use for downloading/installing component code. The client should register its IBindStatusCallback in this bind context to receive callbacks during the download and installation process. (See HREF="asyncmon.doc specification for details)
dwClsContext	DWORD	Values taken from the CLSCTX enumeration specifying the execution context for the class object.
pvReserved	LPVOID	Reserved value, must be set to NULL.
riid	REFIID	The interface to obtain on the factory object (typically IClassFactory).
ppv	VOID **	Pointer in which to store the interface pointer upon return if the call is synchronous.
Returns	S_OK	Success. ppv contains the requested interface pointer.
	MK_S_ASYNCHRONOUS	Component code will be downloaded and installed asynchronously. The client will receive notifications through the IBIndStatusCallback interface it has registered on pBindCtx..
	E_NOINTERFACE	The desired interface pointer is not available. Other CoGetClassObject error return values are also possible here..

In the common web-browser scenario, the values for parameters passed to this function are read directly from an HTML OBJECT tag. For example, the szCodeURL, dwFileVersionMS, and dwFileVersionLS are specified inside an <OBJECT> tag as "CODEBASE=http://www.foo.com/bar.ocx#Version=a,b,c,d", where szCodeURL is "http://www.foo.com/bar.ocx", dwFileVersionMS is MAKELONG(b, a), and dwFileVersionLS is MAKELONG(d, c).

Because the downloading and installation of code occurs asynchronously, CoGetClassObjectFromURL will often return immediately with a return value of E_PENDING. At this point, the IBindStatusCallBack mechanism is used to communicate the status of the download operation to the client[9]. To participate in this communication, the client *must* implement IBindStatusCallback and register this interface in the pBindCtx passed into CoGetClassObjectFromURL using RegisterBindStatusCallback. The client can expect to be called

[9] See the HREF="asyncmon.doc specification for further details.

with callback notifications for OnStartBinding (providing an IBinding for controlling the download), OnProgress (reporting progress), OnObjectAvailable (which returns the desired object interface pointer), and OnStopBinding (which returns error codes in case of an error). For further negotiations, the client *must* also implement ICodeInstall as described below.

Note: the initial (beta) implementation of CoGetClassObjectFromURL will not handle system-wide simultaneous downloads of the *same* code. Similarly, it will not handle cases where different simultaneous downloads refer to the same piece of dependent code.

3.2.2IBindStatusCallback::OnProgress

The client of CoGetClassObjectFromURL will receive notification about the download / install process via the provided IBindStatusCallback interface. During the download process , the following additional values (from the BINDSTATUS enumeration) may be passed back as the ulStatusCode parameter for IBindStatusCallback::OnProgress.

Value	Description
BINDSTATUS_BEGINDOWNLOADCOMPONENTS	The download operation has begun downloading code for COM components that will be installed before the object may be instantiated. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the component being downloaded.
BINDSTATUS_INSTALLINGCOMPONENTS	The download operation has downloaded code and is installing it. The szStatusText accompanying IBindStatusCallback::OnProgress() provides the display name of the component being installed.
BINDSTATUS_ENDDOWNLOADCOMPONENTS	The download operation has finished downloading and installing all necessary code. The szStatusText accompanying OnProgress() provides the display name of the newly installed component.

3.2.3ICodeInstall

A code install operation requires additional services from the client in order to complete the negotiation necessary for a download operation. Such services are requested using IBindStatusCallback::QueryInterface. The specific inter

face requested in IBindStatusCallback::QueryInterface is ICodeInstall. This interface must be implemented by a client of Internet Component Download.

```
interface ICodeInstall : IUnknown {
    HRESULT     GetWindow( [out] HWND* phwnd);
    HRESULT     OnCodeInstallProblem( [in] ULONG ulStatusCode, [in] LPCWSTR szDestination,
                                      [in] LPCWSTR szSource, [in] DWORD dwReserved);
    };
```

ICodeInstall::GetWindow

This function is called when Component Download needs to display user interface for verification of downloaded code[10]. When a client is called with this function, it has the opportunity to clear the message queue of it's parent window before allowing UI to be displayed. If the client does not wish to display UI, code verification may continue, but components may fail to be installed. Note: ICodeInstall actually inherits from the IWindow interface (see documentation for URL Monikers), which consists of the single member function GetWindow. However, as far as any client code is concerned, the interface definition in this specification is still accurate.

[10] Actually, this UI is displayed by the WinVerifyTrust mechanism that is used within Component Download.

Argument	Type	Description
phwnd	HWND *	Client-provided HWND of the parent window for displaying code verification UI. If this parameter is NULL, the desktop window is used. If the value is INVALID_HANDLE_VALUE, or if the return value is S_FALSE, then no code verification UI will be displayed, and certain necessary components may not be installed.
Returns	S_OK	Success.
	S_FALSE	No window is available.
	E_INVALIDARG	The argument is invalid.

ICodeInstall::OnCodeInstallProblem

This function is called when there is a problem with code installation. This notification gives the client a chance to resolve the problem, often by displaying UI, or by aborting the code installation process. **Note:** if the client does not understand the problem, it should return E_ABORT by default to abort the code installation process, because returning S_OK would imply retrying the operation.

Argument	Type	Description
ulStatusCode	ULONG	Status code describing what problem occurred. A member of CIP_STATUS.
szDestination	LPCWSTR	The name of the existing file that was causing a problem. This may be the name of an existing file that needs to be overwritten, the name of a directory causing access problems, or the name of a drive that is full.
szSource	LPCWSTR	Name of the new file to replace the existing file (if applicable).
dwReserved	DWORD	Reserved for future use.
Returns	S_OK	Continue the installation process. If there was an "access denied" or disk-full problem, retry the installation. If there was an existing file (newer *or* older version), overwrite it.
	S_FALSE	Skip this particular file, but continue with the rest of the code installation process. Note: this is the typical response for the CIP_NEWER_VERSION_EXISTS case.
	E_ABORT	Abort the code installation process.
	E_INVALIDARG	The given arguments are invalid.

The ulStatusCode parameter above is one of the following values:

```
typedef enum {
    CIP_DISK_FULL,
    CIP_ACCESS_DENIED,
    CIP_OLDER_VERSION_EXISTS,
    CIP_NEWER_VERSION_EXISTS,
    CIP_NAME_CONFLICT,
    CIP_TRUST_VERIFICATION_COMPONENT_MISSING
} CIP_STATUS;
```

Value	Description
CIP_DRIVE_FULL	The drive specified in szDestination is full..
CIP_ACCESS_DENIED	Access to the file specified in szDestination is denied.
CIP_OLDER_VERSION_EXISTS	An existing file (older version) specified in szDestination needs to be overwritten by the file specified in szSource.
CIP_NEWER_VERSION_EXISTS	A file exists (specified in szDestination) that is a newer version of a file to be installed (specified in szSource)
CIP_NAME_CONFLICT	A file exists (specified in szDestination) that has a naming conflict with a file to be installed (specified in szSource). The existing file is neither a newer nor an older version of the new file—they are mismatched but have the same file name.
CIP_TRUST_VERIFICATION_COMPONENT_MISSING	The code installation process cannot find the necessary component (WinVerifyTrust) for verifying trust in downloaded code. szSource specifies the name of the file that cannot be certified. The client should display UI asking the user whether or not to install the untrusted code, and should then return E_ABORT to abort the download, S_OK to continue anyway, or S_FALSE to skip this file but continue (usually dangerous).

4Storing / Caching Downloaded Code

Code Download installs most new code in a permanent store in windows\occache.[11] Some components (helper DLLs that need to be on the system PATH but currently are not) will also be installed in \windows and \windows\system. All downloaded code is registered using a new registry "Module Usage" section that keeps track of such code. Downloaded code is *not* removed automatically, but it is possible in the future to add UI to the Control Panel (or elsewhere) allowing a user to clean up this directory.

For future releases, it is also possible to convert this "permanent store" into a code cache that retains only popular downloaded code and deletes old unused code automatically. This migration plan justifies use of a permanent store for the first version. See Appendix for registry details on how downloaded code is listed in the registry and how a code cache could function in future releases..

5Internet Search Path

When Internet Component Download is called to download code, it traverses the *Internet Search Path* to look for the desired component. This path is a list of *Object Store* servers that will be queried every time components are downloaded using CoGetClassObjectFromURL. This way, even if an <OBJECT> tag in an HTML document does not specify a CODEBASE location to download code for an embedded OLE Control, the Internet Component Download will still use the Internet Search Path to find the necessary code.

5.1Internet Search Path syntax

The search path is specified in a string in the registry, under the key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Internet Settings\CodeBaseSearchPath. The value for this key is a string in the following format:

CodeBaseSearchPath = <URL_1>; <URL_2>; … <URL_m>; CODEBASE; <URL_{m+1}>; … <URL_{n-1}>; <URL_n>

[11] This directory location is hard-coded for initial releases. In future releases users may use a registry setting or a Control Panel applet to choose this directory. Component code will be installed in this directory unless a previous version exist. In such cases, the Component Download mechanism will attempt to replace the previous version and invoke ICodeInstall::OnCodeInstallProblem.

Where each of URL_1 through URL_n are absolute URLs pointing to HTTP servers acting as "Object Stores". When processing a call to CoGetClassObjectFromURL, the Internet Component Download service will *first* try downloading the desired code from the locations URL_1 through URL_m, it will *then* try the location specified in the szCodeURL parameter (corresponding to the CODEBASE attribute in the <OBJECT> tag), and will *finally* try the locations specified in locations URL_{m+1} through URL_n.

Note that if the CODEBASE keyword is not included in the CodeBaseSearchPath key, then calls to CoGetClassObjectFromURL will never check the szCodeURL location for downloading code. By removing the CODEBASE keyword from the CodeBaseSearchPath, corporate intranet administrators can effectively disable Internet Component Download for corporate users.

5.2 Object Stores

An *Object Store* on the Internet Search Path is an HTTP server that services requests for download-able code[12]. During a call to CoGetClassObjectFromURL, Internet Component Download will try to download code from the various Object Stores on the search path. Specifically, an Object Store will receive an HTTP POST request with data in the format below.[13]

```
CLSID={class id}
Version=a,b,c,d
MIMETYPE=mimetype
```

All the values above are optional, although one of CLSID or MIMETYPE *must* be present. The Object Store should parse this information, check an internal database, and either fail the call, or ***redirect*** the HTTP request to the download-able code Cabinet file (.CAB), setup script (.INF), or portable executable (.EXE/.DLL/.OCX).

The POST parameters should be processed by the Object Store as follows:

If CLSID is provided with no version number, then the most recent object matching the CLSID will be returned. If the CLSID is provided with Version, then the object matching the CLSID and with the *largest version number greater than or equal to* Version will be provided. If no object is available that matches the CLSID with a large enough version number, then the 404 error will be returned. MIMETYPE will be ignored when CLSID is provided.

If no CLSID is provided, but MIMETYPE is provided, then the first object found in the database that matches the MIMETYPE will be returned. Version, if provided, is treated *exactly* as above. If neither CLSID or MIMETYPE is provided then the error return code 400 Bad Request will be returned.

In addition to the POST data described above, queries to Object Stores will also include HTTP headers for Accept (MIME type) and Accept-Language, thus specifying the desired platforms (see above for Platform Independence and HTTP) and language-localized implementation for a component. Note that these HTTP headers are added to all HTTP requests made by Internet Component Download. This allows Object Stores to serve different code implementations for differing platforms or even different languages.

Note: Internet Component Download will use the *first successful response* from a server on the Internet Search Path. Component Download will *not* continue searching for newer versions of components.

5.3 Uses for Internet Search Path

The Internet Search Path can be used in two ways:

1. Object Store servers at the beginning of the path will be asked for code *before* checking the location specified in the szCodeURL parameter for CoGetClassObjectFromURL. Servers at the beginning of the search path will thus be checked *before* trying the location specified in the CODEBASE attribute of an <OBJECT> tag. This is a useful feature for corporate intranets, because it allows intranet administrators to set up a local Object Store that is used to serve code for download by employees. (in fact, it is possible to disable the CODEBASE attribute for <OBJECT> tags by removing the CODEBASE keyword from the search path.

[12] Currently Object Stores *must* be HTTP servers that can serve content dynamically, for instance via ISAPI. In future versions a mechanism will be introduced allowing non-HTTP Object Stores.

[13] Note: an HTTP ***POST*** request is used, not a GET request. This is because the number of parameters involved is large enough that a GET request may exceed the maximum URL length of 1024 characters.

2. "Object Store" servers at the end of the search path will be asked for code *after* trying the location specified in the szCodeURL parameter for CoGetClassObjectFromURL, and thus *after* trying the location specified in the CODEBASE attribute. This allows registration of default Object Store locations on the World Wide Web, where browsers can find code when no CODEBASE location is explicitly specified.

6Future directions

6.1Internet Search Path without HTTP

6.2The Internet Search Path assumes that all Object Stores on the search path are "Active" HTTP servers capable of handling HTTP POST requests and querying an object database. In future revisions, it is planned to allow Object Stores on FILE or FTP servers (or simple HTTP servers) in addition to the existing support for "Active" HTTP servers. No further details are available."Pluggable" Setup-script handlers

Although Internet Component Download currently supports a limited .INF setup-script syntax, future releases will take into consideration the need to support "hooks" that allow custom setup handlers to interact with the component download and installation process. For example, it would be desirable to use Win32 standard SetupX .INF files for installation. However, such scripts are not "safe", and allowing such installations would require signing of "trusted": setup scripts. Such work is being considered for future versions of Internet Component Download. No further details are available at this time.

7Needs that aren't met by Internet Component Download

There are various situations in which code needs to be downloaded with trust verification but the code is not an OLE Object. Such cases are not addressed by the current specification of the Internet Component Download mechanism. Solutions for these cases need to use the WinVerifyTrust mechanism directly, as detailed below:

- **<A HREF> tag in HTML:** It is possible in HTML to download and run .EXE files directly using the <A HREF> tag. The HTML parser uses URL Moniker directly to download this code, and it calls WinVerifyTrust to check validity.
- **Scripts:** scripting languages will need to define a mechanism for inserting certificates in the script (perhaps in special comments). Given such a mechanism, the WinVerifyTrust service will provide trust verification of any such scripts that are downloaded from the internet.
- **Full applications, other:** The existing Internet Component Download will not handle extremely complex download situations (e.g. download/install DOOM, register device drivers, reboot machine). Future releases will aim to allow hooking into the Component Download mechanism to provide more complicated setup routines.

8Appendix - Registry Details

The Internet Component Download service will keep registry entries for every new downloaded component. These registry entries will be useful for (a) writing a utility for cleaning up the code storage, or (b) migrating the Component Download service to use a code cache rather than a permanent store.[14]

8.1Why the existing "SharedDLL" mechanism is inadequate

To do correct code caching, the existing shared DLL ref. counting scheme will not suffice, because ref. counts are easily inflated. Specifically, any application that is re-installed increases the ref. count on a shared DLL even though that DLL already has a ref. count belonging to the particular application. (this is already broken for current ref. counting, but it will especially fail for Code Download, in which OCXs are used by multiple pages quite regularly, and there is no way of knowing which OCXs need reference counts.

8.2The new "ModuleUsage" mechanism in the registry for tracking usage of shared components.

To do ref. counting correctly, Component Download will maintain a ModuleUsage section in the registry which holds a list of "owners" and "clients" for each shared module. Thus the registry can keep track of *who* is using a shared module, not just *how many clients* that module has. The registry entries would use the following syntax:

```
[ModuleUsage]
    [<Fully Qualified Path&File Name>]
        .FileVersion=a,b,c,d
        .Owner = Friendly Name/ID of Owner
        <Client ID > = <info peculiar to this client>
        <Client ID > = <info peculiar to this client>
```

A ModuleUsage section in a sample registry would look something like the following:

```
Under My Computer\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion:

[ModuleUsage]
    [c:\windows/system/mfc40.dll]
        .FileVersion=0,4,0,0
        .Owner = Microsoft Internet Code Downloader
        Microsoft Internet Code Downloader= <any info, or default>
        AnotherAppID= <any info, or default>
```

Key name	Description
<Fully Qualified Path&File Name>	This is the full path of the shared module. This name has to use "/"s instead of "\"s because the "\" is an invalid char in a key name.
.Owner	The application that installs the shared module and creates the original ModuleUsage section will put some identifier in the Owner key section. If the DLL already existed on the system then and this Module Usage key did not exist then the .Owner key should be set to "Unknown" and the DLL should not be removed on uninstall. The owner should always also enlist itself as a client.
.File Version	The version number for the shared module.
<Client ID>	ID of a client who is using the shared module. The value corresponding to each client key contains client specific information. When the client is Internet Component Download, the <Client ID> is "Microsoft Internet Code Downloader", and the client-specific information is a number which serves as a reference count. For other clients, the client-specific information should be the full path

[14] Either of these would be intelligent about un-installing and un-registering component code using its existing self-registration mechanism.

	of the client, so that if the client is accidentally deleted it is possible to do garbage collection.

Every client of this module is expected to increment and decrement the existing SharedDLLs section in the registry as well (a client only increments this value once when it adds itself as a client under [ModuleUsage]). This is to allow a migration path for apps currently implementing only SharedDLLs scheme.

This registry information complements the reference counts in the SharedDLLs section by remembering which clients are actually using a shared module. This counting scheme will work correctly and allow caching of downloaded code. Furthermore, when downloading files, Internet Component Download can use this registry information as an efficient shortcut for verifying whether a file needs to be overwritten because it is an out-of-date version.