Component Categories

OLE Program Management
June 19~~January 2~~, 1996 Draft

The categorization of a COM class – roughly what interface(s) and related usage it supports – is currently exposed via registry keys such as Insertable and NotInsertable, Control, and DataSource. This mechanism is ill-defined and unsuitable for larger numbers of categories expected in a complex COM-based system. A more robust and well-defined mechanism for categorizing COM objects is defined in this document

1Introduction

Being able to group COM classes based on their category is extremely useful. Building lists of available classes for a particular task which the user may choose from, an object which automatically chooses to aggregate in the closest/smallest/most-efficient class from those available, and type browsers for COM development tools are just some examples of using such categorization information.[1]

When we really speak of the category of a COM class, though, we are talking about more than just the interfaces an instance will support (its 'interface signature'). We are far more interested in the way in which an instance may be used – in the class's richer usage capabilities. The interface signature alone is often insufficient; knowing, for example, which classes support IDataObject doesn't help you decide which ones are actually IDataObject-based text converters. In essence, using just interface signatures to convey categorization precludes convenient reuse of generic interfaces, which is a key goal of COM.[2] In addition, because creating an instance of a component can be an expensive operation, there is often the requirement that identification of a class's "richer usage capabilities" be made without instantiation.

The current ad-hoc categorization mechanism for COM classes is to add a human-readable sub-key to the CLSID of the class's registry information. For example, the HKEY_CLASSES_ROOT\CLSID\{...}\Insertable key is used to indicate that a class is insertable as an OLE object (meaning that an instance of the class is expected to support at least the IOleObject, IPersistStorage, IDataObject, as well as a handful of other, interfaces, used in a well-defined way).

Using such human-readable keys is convenient for the component builder but is unsuitable for installations with large numbers of categories, where collisions in human-readable keys invented by multiple parties would cause confusion at least and failures at worst.

With the widespread use of COM components and "container applications" for those components, there also arises the requirement for being able to identify what container functionality a component requires and this identification has to be made prior to instantiating the component.[3]

For these reasons, and to better consolidate component categories under each class's CLSID root-key, a set of registry entries are defined, and a set of APIs, to wrap the manipulation of these keys are introduced; which we call Component Categories.

1.1Categorizing By Component Capabilities

As mentioned above, in many user interface scenarios containers need to be able to display to the user some subset of all of the components installed on the machine. In the past this has been accomplished by defining registry keys such as "Insertable" or "Control". Component categories provides a richer, extensible, and more robust mechanism for doing the same thing. Each component category is identified by a GUID, referred to as a Category ID or CATID. Each category ID has a list of locale tagged human readable names associated with it. A list of the category IDs, along with the human readable name, is stored in a well known location in the system registry.

For example, all components that implement the functionality required to be used in OLE Document embedding scenarios can be classified within a component category. Historically, these objects have been identified by the "Insertable" key in the registry. Using component categories instead, there would be the following information in the registry:

```
HKEY_CLASSES_ROOT\Component Categories\{40FC6ED3-2438-11cf-A3DB-080036F12502}
                        = (default)[4] - ""
                        = 409 - "OLE Document Embeddings"
```

Each class that implements the functionality corresponding to a component category lists the category ID for that category within it's CLSID key in the registry. Because a single component can support a wide range of functionality, components may belong to multiple component categories. For example, a particular OLE Control might support all of the functionality required to participate as OLE Document embed-

[1] In the categorization model discussed in this document, there is no explicit type hierarchy. Note, however, that a hierarchy can be externally imposed on the category information when those classes that support more than one Component Category.

[2] This is another aspect in which the Component Category model can be seen to differ from a formal type hierarchy. Formal types are typically based purely on base interfaces and derived or additional interfaces. Component Categories are designed to capture an author's intention that one object be used interchangeably with another.

[3] Instantiation needs to be avoided primarily for performance/user interface reasons.

[4] The Registry supports a "default" named value on a given key. We use the notation "(default)" to indicate the default named value. The default named value usually (but not always) is nameless.

ding, Visual Basic data binding, *and* internet functionality. Such a control would have the following information stored within its CLSID key in the registry:

```
; The CLSID for "My Super OLE Control" is {12345678-ABCD-4321-0101-00000000000C}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}\Implemented Categories

; The CATID for "Insertable" is {40FC6E D3-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}\Implemented Categories\{40FC6E D3-
        2438-11cf-A3DB-080036F12502}

; The CATID for "Control" is {40FC6E D4-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}\Implemented Categories\{40FC6E D4-
        2438-11cf-A3DB-080036F12502}

; The CATID for an internet aware control is {...CATID_InternetAware...}
HKEY_CLASSES_ROOT\CLSID\{12345678-ABCD-4321-0101-00000000000C}\Implemented Categories\{...
        CATID_InternetAware...}
```

With this information, a container can enumerate the controls installed on a users system and only display the controls that support some functionality the container requires. In other words, component categories provides a way to categorize components by the *implemented functionality* of the component.

1.2Categorizing By Container Capabilities

The previous section discussed using component categories to allow a container to enumerate the controls installed on a users system and only display the controls that support some functionality the container requires. However components often require certain functionality *from the container*, and simply will not work with a container that does not provide the support. It is inappropriate to force the end user to choose a component only to find out that it is not suitable for the task at hand. Instead, the user interface should be able to filter out components that require functionality the container does not support. For this reason, the concept of categorizing components by *required container functionality* is introduced.

Simple frame OLE Controls are a good example of components that require functionality from the container (i.e. container must implement ISimpleFrameSite) and do not work in containers that do not support that functionality.

Categorizing by container capabilities is accomplished through an additional registry key within the component's CLSID key: Required Categories. To illustrate, below are the relevant registry entries for a simple frame control:

```
; The CLSID for "Revo's Simple Frame Control" is {123456FF-ABCD-4321-0101-00000000000C}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}\Implemented Categories

; The CATID for "Control" is {40FC6E D4-2438-11cf-A3DB-080036F12502}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}\Implemented Categories\{40FC6E D4-
        2438-11cf-A3DB-080036F12502}

; The CATID for simple frame controls is {...CATID_ SimpleFrameControl...}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}\Implemented Categories\
        {...CATID_SimpleFrameControl...}
HKEY_CLASSES_ROOT\CLSID\{123456FF-ABCD-4321-0101-00000000000C}\Reqruied Categories\
        {...CATID_SimpleFrameControl...}
```

As illustrated by this example, a component can belong to component categories that indicate supported functionality *and* component categories that indicate required functionality.

> The use of required categories should be used with care because it severely restricts the ways in which a component can be used. While implemented categories are hints to a container about what functionality a component supports, required categories indicate that the component *can only be used in a very specific scenario.* In general, components should be coded such that they are useful regardless of what services the container/user provides; gracefully degrading their functionality if some container provided service is not available at runtime.

A complete example using OLE Controls. Assume the following component categories are defined for OLE Controls:

Category	Impl.	Req.	Description
CATID_Control	Yes	No	Indicates that the component is an OLE Control. The minimum requirements for this are support for IOleControl, IPersistStreamInit, and IDispatch.
CATID_VBDataBound	Yes	Yes	Indicates that the component supports VB data binding. If specified as a required category, indicates that the component will only work in a container that supports VB data binding.
CATID_SimpleFrame	Yes	Yes	Indicates that the component is a simple frame control. If specified as a required category, indicates that the component will only work in a container that supports VB data binding.

Assume that installed on the users system are the following controls:

```
; The button control is just a generic OLE Control that supports no additional functionality.  It will
; work in any OLE Control container.
HKCR[5]\CLSID\{...CLSID_Button...}\Implemented Categories
HKCR\CLSID\{...CLSID_Button...}\Implemented Categories\{...CATID_Control...}

; The MyDBControl can use VB data binding if the container supports it.  However, it's been coded
; such that it will work in containers that do not support VB data binding (perhaps via a different
; database API)
HKCR\CLSID\{...CLSID_MyDBControl...}\Implemented Categories
HKCR\CLSID\{...CLSID_MyDBControl...}\Implemented Categories\{...CATID_Control...}
HKCR\CLSID\{...CLSID_MyDBControl...}\Implemented Categories\{...CATID_VBDatabound...}

; The GroupBox control is a simple frame control.  It relies on the container implementing the
; ISimpleFrameSite interface, and thus will only work correctly in such containers.
HKCR\CLSID\{...CLSID_GroupBox...}\Implemented Categories
HKCR\CLSID\{...CLSID_GroupBox...}\Implemented Categories\{...CATID_Control...}
HKCR\CLSID\{...CLSID_GroupBox...}\Implemented Categories\{...CATID_SimpleFrame...}
HKCR\CLSID\{...CLSID_GroupBox...}\Required Categories\{...CATID_impleFrame...}
```

In our example, a container that supports VB data bound controls, but not simple frame controls would specify CATID_Control, and CATID_VBDatabound to the insert control user interface. The list of controls displayed to the user would contain CLSID_Button, CLSID_MyDbControl, but not CLSID_GroupBox.

1.3Default Classes and Associations

For certain categories, users and tools want to associate a single class as the default class to choose in any scenario requiring that category of object. The manner in which file-extensions are currently associated with executables is an example of this sort of behavior and user interface. Extending this to components, a package may ship with its own e-mail editor or web browser, however, having designed these features as components to standard interfaces, and having introduced a CATID that identifies exchangeable component, they would like the user to be able to choose their preferred editor or browser in all UI situations where one is to be created.

Note that this capability is not useful for all component categories: establishing a default Insertable class, or some sort of default graphic filter class is not at generally useful or necessary. Generally speaking, establishing a default class is useful when some class must be loaded from a list of possible classes without intervention by an interactive user and yet the consumer of the service provided by the class does not want to be hard-wired to a particular implementation. Users and/or administrators can externally define which class is to be used by such consumers by manipulating the registry. For example, a UI-less background multimedia rendering tool may need to load an MPEG filter component and there are a dozen such filter components available on the market and perhaps several on a given machine. In that case, as part of the installation process the rendering tool would ask the user to specify which filter component to use by default.

To associate a default class with a category, introduce a CLSID key whose CLSID is the same as the CATID of the component category in question. Add the TreatAs key to this key, with a value of the CLSID of the default class for this category. Tools (and applications the user is running) which want to use the default class for an component category simply use CoCreateInstance or CoGetClassObject and specify the CATID for the CLSID parameter, which will automatically redirect to the CLSID established as the default for this category via this key.

[5] Abbreviation for HKEY_CLASSES_ROOT

```
HKEY_CLASSES_ROOT\CLSID\{…catid…}\TreatAs      = {… default clsid…}
```

During installation, a component can check for the existence of any default class keys for its category(ies) and present the user with options for overriding their current settings. Further user interface to allow the user to select between compatible classes post-install can also be helpful: examples include a control panel application that allows the user to choose between compatible e-mail editors or browsers, or a drop-down list in an application's Options… dialog that allows the user to select among compatible text or graphic editors.[6]

1.4Defining Component Categories

In general the author of an component category definition will simply create a unique GUID (the CATID) which is published along with the definition. Other parties, object builders or tools, then know the definition of this type and can make use of classes which support it accordingly. Like the method signature of an interface which is immutable once shipped, a category's semantics should not be modified after being installed as part of a shipping product. It is preferable to maintain backward compatibility of the category by simply introducing a new category identifier with the revised semantics.

Because interface identifiers (IID) and component category identifiers (CATID) exist in different name spaces, it appears as though it is possible to use the same GUID value for both an IID and a CATID. However, because IIDs are often used for the CLSID of the interface's proxy/stub server there is the potential for conflict (especially in the default class scenario described below). Therefore using the same GUID for an IID and CATID is strictly prohibited.

1.5Associating Icons with a Category

In building a user interface for selecting component categories and the components within a category, the ability to display a meaningful image for a category is required. To facilitate this we leverage the DefaultIcon and ToolboxBitmap32 keys already defined by OLE Documents and OLE Controls.

To associate an icon with a component category create a key in HKEY_CLASSES_ROOT\CLSID for the category's CATID and populate that key with a DefaultIcon sub-key. For example:

```
HKEY_CLASSES_ROOT\CLSID\{…catid…}\DefaultIcon = "c:\hello\icons.dll,1"
```

The filename referenced by the DefaultIcon key can be a .EXE or .DLL file (and can be a resource only .DLL).

To associate small 16x16 "toolbox bitmap" with a component category create a key in HKEY_CLASSES_ROOT\CLSID for the category's CATID and populate that key with a ToolboxBitmap32 sub-key. For example:

```
HKEY_CLASSES_ROOT\CLSID\{…catid…}\ToolboxBitmap32      = "c:\goodbye\mycomponent.dll,42"
```

The filename referenced by the ToolboxBitmap32 key can be a .EXE or .DLL file (and can be a resource only .DLL).

2Component Categories API

To facilitate using component categories, an in-proc server will be provided that implements the interfaces described below. This component, called the Component Category Manager will have a CLSID of TBD. This component supports aggregation such that it can be extended by 3rd parties.[7]

```
// CLSID_StdComponentCategoriesyMgr: {0002E00500185-0000-0000-c000-000000000046}
```

The IDL file (comcat.idl) for Component Categories is given below.

```
//+-------------------------------------------------------------------------
//
```

[6] Further UI details TBD. Some visual examples as well as how you use the APIs to build the examples is probably a good idea.
[7] It is expected that OLE Controls will require an extension to the interfaces defined here to allow access to "Mandatory" function group information.

```
//  Copyright 1995 - 1996 Microsoft Corporation. All Rights Reserved.
//
//  Contents:   Component Categories Interfaces
//
//--------------------------------------------------------------------------

cpp_quote("//+-------------------------------------------------------------------------")
cpp_quote("//")
cpp_quote("//  Microsoft Windows")
cpp_quote("//  Copyright 1995 - 1996 Microsoft Corporation. All Rights Reserved.")
cpp_quote("//")
cpp_quote("//  File: comcat.h")
cpp_quote("//")
cpp_quote("//-------------------------------------------------------------------------")

#ifndef DO_NO_IMPORTS
import "unknwn.idl";
#endif

interface IEnumGUID;
interface IEnumCATEGORYINFO;
interface ICatRegister;
interface ICatInformation;

cpp_quote("")
cpp_quote("////////////////////////////////////////////////////////////////////////////")
cpp_quote("//  Classes (link with uuid3.lib)")
cpp_quote("")
cpp_quote("EXTERN_C const CLSID CLSID_StdComponentCategoriesMgr;")

cpp_quote("")
cpp_quote("////////////////////////////////////////////////////////////////////////////")
cpp_quote("//  Types")
   typedef GUID CATID;
   typedef REFGUID REFCATID;
cpp_quote("#define IID_IEnumCLSID            IID_IEnumGUID")
cpp_quote("#define IEnumCLSID                IEnumGUID")
cpp_quote("#define LPENUMCLSID               LPENUMGUID")
#define IEnumCLSID                  IEnumGUID
cpp_quote("#define CATID_NULL                GUID_NULL")
cpp_quote("#define IsEqualCATID(rcatid1, rcatid2)      IsEqualGUID(rcatid1, rcatid2)")
cpp_quote("#define IID_IEnumCATID       IID_IEnumGUID")
cpp_quote("#define IEnumCATID          IEnumGUID")
#define IEnumCATID                  IEnumGUID

cpp_quote("")
cpp_quote("////////////////////////////////////////////////////////////////////////////")
cpp_quote("//  Category IDs (link to uuid3.lib)")
cpp_quote("EXTERN_C const CATID CATID_Insertable;")
cpp_quote("EXTERN_C const CATID CATID_Control;")
cpp_quote("EXTERN_C const CATID CATID_Programmable;")
cpp_quote("EXTERN_C const CATID CATID_IsShortcut;")
cpp_quote("EXTERN_C const CATID CATID_NeverShowExt;")
cpp_quote("EXTERN_C const CATID CATID_DocObject;")
cpp_quote("EXTERN_C const CATID CATID_Printable;")
cpp_quote("EXTERN_C const CATID CATID_RequiresDataPathHost;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToMoniker;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToStorage;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToStreamInit;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToStream;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToMemory;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToFile;")
cpp_quote("EXTERN_C const CATID CATID_PersistsToPropertyBag;")

cpp_quote("")
cpp_quote("////////////////////////////////////////////////////////////////////////////")
cpp_quote("//  Interface Definitions")

//+-------------------------------------------------------------------------
//
```

```
//  Copyright (C) Microsoft Corporation, 1995 - 1996.
//
//  Contents:    IEnumGUID interface definition
//
//--------------------------------------------------------------------------
cpp_quote("#ifndef _LPENUMGUID_DEFINED")
cpp_quote("#define _LPENUMGUID_DEFINED")
[
    object,
    uuid(0002E000-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IEnumGUID : IUnknown
{
    typedef [unique] IEnumGUID *LPENUMGUID;

    HRESULT Next(
        [in] ULONG celt,
        [out, size_is(celt), length_is(*pceltFetched)] GUID *rgelt,
        [out] ULONG *pceltFetched);

    HRESULT Skip(
        [in] ULONG celt);

    HRESULT Reset();

    HRESULT Clone(
        [out] IEnumGUID **ppenum);
}
cpp_quote("#endif")


//+-------------------------------------------------------------------------
//
//  Copyright (C) Microsoft Corporation, 1995 - 1996.
//
//  Contents:    IEnumCATEGORYINFO definition
//
//--------------------------------------------------------------------------
cpp_quote("#ifndef _LPENUMCATEGORYINFO_DEFINED")
cpp_quote("#define _LPENUMCATEGORYINFO_DEFINED")
[
    object,
    uuid(0002E011-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IEnumCATEGORYINFO : IUnknown
{
    typedef [unique] IEnumCATEGORYINFO *LPENUMCATEGORYINFO;

    typedef struct tagCATEGORYINFO {
        CATID       catid;
        LCID        lcid;
        OLECHAR szDescription[128];
    } CATEGORYINFO, *LPCATEGORYINFO;

    HRESULT Next(
        [in] ULONG celt,
        [out, size_is(celt), length_is(*pceltFetched)] CATEGORYINFO* rgelt,
        [out] ULONG* pceltFetched);

    HRESULT Skip(
        [in] ULONG celt);

    HRESULT Reset();

    HRESULT Clone(
        [out] IEnumCATEGORYINFO** ppenum);
}
cpp_quote("#endif")
```

```
//+-------------------------------------------------------------------------
//
//  Copyright (C) Microsoft Corporation, 1995 - 1996.
//
//  Contents:    ICatRegister definition
//
//--------------------------------------------------------------------------
cpp_quote("#ifndef _LPCATREGISTER_DEFINED")
cpp_quote("#define _LPCATREGISTER_DEFINED")
[
    object,
    uuid(0002E012-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface ICatRegister : IUnknown
{
    typedef [unique] ICatRegister* LPCATREGISTER;

    HRESULT RegisterCategories(
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATEGORYINFO rgCategoryInfo[]);

    HRESULT UnRegisterCategories(
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT RegisterClassImplCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT UnRegisterClassImplCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT RegisterClassReqCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT UnRegisterClassReqCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);
}
cpp_quote("#endif")


//+-------------------------------------------------------------------------
//
//  Copyright (C) Microsoft Corporation, 1995 - 1996.
//
//  Contents:    ICatInformation definition
//
//--------------------------------------------------------------------------
cpp_quote("#ifndef _LPCATINFORMATION_DEFINED")
cpp_quote("#define _LPCATINFORMATION_DEFINED")
[
    object,
    uuid(0002E013-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface ICatInformation : IUnknown
{
    typedef [unique] ICatInformation* LPCATINFORMATION;

    HRESULT EnumCategories(
```

```
        [in] LCID lcid,
        [out] IEnumCATEGORYINFO** ppenumCategoryInfo);

    HRESULT GetCategoryDesc(
        [in] REFCATID rcatid,
        [in] LCID lcid,
        [out] LPWSTR* pszDesc);

    HRESULT EnumClassesOfCategories(
        [in] ULONG cImplemented,
        [in,size_is(cImplemented)] CATID rgcatidImpl[],
        [in] ULONG cRequired,
        [in,size_is(cRequired)] CATID rgcatidReq[],
        [out] IEnumCLSID** ppenumClsid);

    HRESULT IsClassOfCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cImplemented,
        [in,size_is(cImplemented)] CATID rgcatidImpl[],
        [in] ULONG cRequired,
        [in,size_is(cRequired)] CATID rgcatidReq[]);

    HRESULT EnumImplCategoriesOfClass(
        [in] REFCLSID rclsid,
        [out] IEnumCATID** ppenumCatid);

    HRESULT EnumReqCategoriesOfClass(
        [in] REFCLSID rclsid,
        [out] IEnumCATID** ppenumCatid);
}
cpp_quote("#endif")
```

2.1 Backwards Compatibility Support

To facilitate adoption of Component Categories, and to simplify client code a mapping of existing CLSID registry keys to component categories is provided by the Component Category Manager. Mappings are provided for the following keys:

- Insertable - Denotes an embeddable OLE Documents object. The suggested human readable name for this component category is “Embeddable Object”.
- Control - Denotes an OLE Control.
- Programmable - Denotes an object that supports OLE Automation.
- ~~IsShortcut - Used by the shell.~~
- ~~NeverShowExt - Used by the shell~~
- DocObject - Used by Office 95
- Printable - Used by Office 95

The implementation of ICatInformation will automatically use the appropriate "old" registry key when given the category identifiers for any of these categories. That is, for example, calling ICatInformation::EnumClass-esOfCategory using CATID_Insertable will return an object that will enumerate over all objects registered with either the “Insertable” or “CompoenentCategories\{…CATID_Insertable…}” keys.

```
// Special case CategoryIDs
DEFINE_GUID(CATID_Insertable, 40FC6ED3-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_Control, 40FC6ED4-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_Programmable, 40FC6ED5-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_IsShortcut, 40FC6ED6-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_NeverShowExt, 40FC6ED7-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_DocObject, 40FC6ED8-2438-11cf-A3DB-080036F12502);
DEFINE_GUID(CATID_Printable, 40FC6ED9-2438-11cf-A3DB-080036F12502);
```

There is no provision for a completely extensible mechanism for mapping previously defined CLSID registry keys to component categories.

2.2The ICatRegister Interface

The ICatRegister interface provides methods for registering and un-registering Component Category information.

```
interface ICatRegister : IUnknown
{
    HRESULT RegisterCategories(
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATEGORYINFO rgCategoryInfo[]);

    HRESULT UnRegisterCategories(
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT RegisterClassImplCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT UnRegisterClassImplCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT RegisterClassReqCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);

    HRESULT UnRegisterClassReqCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cCategories,
        [in, size_is(cCategories)] CATID rgcatid[]);
};
```

2.2.1RegisterCategories

HRESULT RegisterCategories (cCategories, rgCategoryInfo)

Registers one or more component categories. Each component category consists of a CATID and a list of locale dependent description strings.

Argument	Type	Description
cCategories	ULONG	Number of component categories to register.
rgCategoryInfo	CATEGORYINFO*	Array of cCategories CATEGORYINFO structures. By providing the same catid for multiple CATEGORYINFOs, multiple locales can be registered for the same component category.
return value	S_OK	Success.
	E_INVALIDARG	One or more arguments are incorrect.
	others	Registry manipulation or GUID to string routine errors.

2.2.2UnRegisterCategory

HRESULT UnRegisterCategory(cCategories, rgcatid)

Removes the registration of one or more component categories. Each component category consists of a CATID and a list of locale dependent description strings.

Argument	Type	Description
cCategories	ULONG	Number of cCategories CATIDs to be removed.
rgcatid	REFCATID	Array of cCategories TYPID. Identifies the categories to be removed.
return value		
S_OK		Success.
E_INVALIDARG		One or more arguments are incorrect.
others		Registry manipulation or GUID to string routine errors.

This call will be successful even if one or more of the category IDs specified are not registered.

Note that this method does not remove the component category tags from individual classes; use the UnRegisterClassCategories method instead.

2.2.3RegisterClassImplCategories

HRESULT RegisterClassImplCategories(rclsid, cCategories, rgcatid)

Registers the class as implementing one or more component categories. On success, subsequent calls on the same machine to CoIsClassOfCategory(rclsid, catid), with catid ∈ rgcatid, will return S_TRUE.

On error, some, but not all, of the category identifiers may have been successfully registered to the class in the system. That is, this function is not required to restore the caller to the exact state prior to the call.

Argument	Type	Description
rclsid	REFCLSID	Class ID of the class to set category information about.
cCategories	ULONG	Number of category GUIDs to associate as category identifiers for the class.
rgcatid	CATID*	Array of cCategories CATID to associate as category identifiers for the class.
return value		
S_OK		Success.
E_INVALIDARG		One or more arguments are incorrect.
others		Registry manipulation or GUID to string routine errors.

2.2.4UnRegisterClassImplCategories

HRESULT UnRegisterClassImplCategories(rclsid, cCategories, rgcatid)

Removes one or more "implemented" category identifiers from a class. This is the inverse of the RegisterClassImplCategories method.

On error, some, but not all, of the category identifiers may have been successfully removed from the class. That is, this function is not required to restore the caller to the exact state prior to the call.

Argument	Type	Description
rclsid	REFCLSID	Class ID of the class to be manipulated.
cCategories	ULONG	Number of category GUIDs to remove.
rgcatid	CATID*	Array of cCategories CATID that are to be removed. Only the category IDs specified in this array are removed.
return value		
S_OK		Success.
E_INVALIDARG		One or more arguments are incorrect.
others		Registry manipulation or GUID to string routine errors.

This call will be successful even if one or more of the category IDs specified are not registered for the class.

2.2.5RegisterClassReqCategories

HRESULT RegisterClassReqCategories(rclsid, cCategories, rgcatid)

Registers the class as requiring one or more component categories. On success, subsequent calls on the same machine to CoIsClassOfCategory(rclsid, catid), with catid ∈ rgcatid, will return S_TRUE.

On error, some, but not all, of the category identifiers may have been successfully registered to the class in the system. That is, this function is not required to restore the caller to the exact state prior to the call.

Argument	Type	Description
rclsid	REFCLSID	Class ID of the class to set category information about.
cCategories	ULONG	Number of category GUIDs to associate as category identifiers for the class.
rgcatid	CATID*	Array of cCategories CATID to associate as category identifiers for the class.
return value		
S_OK		Success.
E_INVALIDARG		One or more arguments are incorrect.
others		Registry manipulation or GUID to string routine errors.

2.2.6UnRegisterClassReqCategories

HRESULT UnRegisterClassReqCategories(rclsid, cCategories, rgcatid)

Removes one or more "required" category identifiers from a class. This is the inverse of the RegisterClass-ReqCategories method.

On error, some, but not all, of the category identifiers may have been successfully removed from the class. That is, this function is not required to restore the caller to the exact state prior to the call.

Argument	Type	Description
rclsid	REFCLSID	Class ID of the class to be manipulated.
cCategories	ULONG	Number of category GUIDs to remove.
rgcatid	CATID*	Array of cCategories CATID that are to be removed. Only the category IDs specified in this array are removed.
return value		
S_OK		Success.
E_INVALIDARG		One or more arguments are incorrect.
others		Registry manipulation or GUID to string routine errors.

This call will be successful even if one or more of the category IDs specified are not registered for the class.

2.3The ICatInformation Interface

The ICatInformation interface allows clients to enumerate available categories, classes that belong to a particular category, and to determine if a particular class belongs to a category.

```
interface ICatInformation : IUnknown
{
    HRESULT EnumCategories(
        [in] LCID lcid,
        [out] IEnumCATEGORYINFO** ppenumCategoryInfo);

    HRESULT GetCategoryDesc(
        [in] REFCATID rcatid,
        [in] LCID lcid,
        [out] OLECHAR* ppszDesc);

    HRESULT EnumClassesOfCategories(
```

```
        [in] ULONG cImplemented,
        [in,size_is(cImplemented)] CATID rgcatidImpl[],
        [in] ULONG cRequired,
        [in,size_is(cRequired)] CATID rgcatidReq[],
        [out] IEnumCLSID** ppenumClsid);

    HRESULT IsClassOfCategories(
        [in] REFCLSID rclsid,
        [in] ULONG cImplemented,
        [in,size_is(cImplemented)] CATID rgcatidImpl[],
        [in] ULONG cRequired,
        [in,size_is(cRequired)] CATID rgcatidReq[]);

    HRESULT EnumImplCategoriesOfClass(
        [in] REFCLSID rclsid,
        [out] IEnumCATID** ppenumCatid);

    HRESULT EnumReqCategoriesOfClass(
        [in] REFCLSID rclsid,
        [out] IEnumCATID** ppenumCatid);
};
```

2.3.1EnumCategories

HRESULT EnumCategories(lcid, ~~rclsid,~~ IEnumCATEGORYINFO** ppenumCatInfo)

Returns an enumerator for the component categories registered on the system.

Argument	Type	Description
lcid	LCID	Identifies the requested locale for any returned szDescription of the enumerated CATEGORYINFOs. Typically the caller specifies GetUserDefaultLCID() for this parameter.
ppenumCatInfo	IEnumCATEGORYINFO**	Location to return an IEnumCATEGORYINFO interface which can be used to enumerate the CATIDs and localized description strings of the component categories which are registered with the system.
return value		
S_OK		Success. ppenumCategoryInfo contains a valid IEnumCATIDINFO enumerator.
E_OUTOFMEMORY		Insufficient memory to create an enumerator object to return.
E_INVALIDARG		ppenumCatInfo is an invalid pointer.
others		Registry manipulation or GUID-to-string routine errors.

2.3.2GetCategoryDesc

HRESULT GetCategoryDesc(rcatid, lcid, ppszDesc)

Retrieves the localized description string for a specified category ID.

Argument	Type	Description
rcatid	REFCATID	Identifies the category for which the description string is to be returned.
lcid	LCID	Specifies which the locale the resulting string should be in.
ppszDesc	PWCHAR*	A pointer to the string pointer that contains the description. Callee allocated, must be freed by the caller using CoMemTaskFree.
return value		
S_OK		The description string was allocated and returned successfully.
CAT_E_CATIDNOEXIST		The Category ID rcatid is not registered.
CAT_E_NODESCRIPTION		There is no description string for rcatid with the specified locale.

E_OUTOFMEMORY	Insufficient memory to create an enumerator object to return.
E_INVALIDARG	ppenumCatInfo is an invalid pointer.
others	Registry manipulation GUID-to-string routine errors, or task allocator errors.

2.3.3EnumClassesOfCategories

HRESULT EnumClassesOfCategories(cImplemented, rgcatidImpl, cRequired, rgcatidReq, ppenum-Clsid)

Returns an enumerator over the classes that implement one or more of rgcatidImpl. If a class requires a category not listed in rgcatidReq, it is not included in the enumeration.

Argument	Type	Description
cImplemented	ULONG	Number of category IDs in the rgcatidImpl array. May not be zero.
rgcatidImpl	CATID[]	Array of category identifiers.
cRequired	ULONG	Number of category IDs in the rgcatidReq array. May be zero.
rgcatidReq	CATID[]	Array of category identifiers.
ppenumCLSID	IEnumCLSID**	Location to return an IEnumCLSID interface which can be used to enumerate the CLSIDs of the classes which implement category rcatid.
return value		
S_OK		Success. ppenumCLSID contains a valid IEnumCLSID enumerator.
E_OUTOFMEMORY		Insufficient memory to create an enumerator object to return.
E_INVALIDARG		ppenumCLSID is an invalid pointer.
others		Registry manipulation or GUID-to-string routine errors.

2.3.4IsClassOfCategories

HRESULT IsClassOfCategories(rclsid, cImplemented, rgcatidImpl, cRequired, rgcatidReq)

Determines if a class supports the specified categories. If the class implements one or more of the categories listed in rgcatidImpl this function returns S_OK, unless the class requires a category not listed in rgcatidReq, in which case it returns E_FAIL.

Argument	Type	Description
rclsid	REFCLSID	Class ID of the class to query.
cImplemented	ULONG	Number of category IDs in the rgcatidImpl array. May not be zero.
rgcatidImpl	CATID[]	Array of category identifiers.
cRequired	ULONG	Number of category IDs in the rgcatidReq array. May be zero.
rgcatidReq	CATID[]	Array of category identifiers.
return value		
S_OK		Yes, rclsid is of category rcatid.
S_FALSE		No, rclsid is not of category rcatid.
others		Registry manipulation or GUID-to-string routine errors.

2.3.5EnumImplCategoriesOfClass

HRESULT EnumImplCategoriesOfClass(rclsid, IEnumCATID** ppenumCatid)

Returns an enumerator over the CATIDs that the specified class implements.

Argument	Type	Description
rclsid	REFCLSID	Class ID.
ppenumCATD	IEnumCATID**	Location to return an IEnumCATID interface which can be used to enumerate the CATIDs which are implemented by rclsid.
return value		
S_OK		Success. ppenumCATID contains a valid IEnumCATID enumerator.
E_OUTOFMEMORY		Insufficient memory to create an enumerator object to return.
E_INVALIDARG		ppenumCATID is an invalid pointer.
		others Registry manipulation or GUID-to-string routine errors.

2.3.6EnumReqCategoriesOfClass

HRESULT EnumReqCategoriesOfClass(rclsid, IEnumCATID** ppenumCatid)

Returns an enumerator over the CATIDs that the specified class requires.

Argument	Type	Description
rclsid	REFCLSID	Class ID.
ppenumCATD	IEnumCATID**	Location to return an IEnumCATID interface which can be used to enumerate the CATIDs which are required by rclsid.
return value		
S_OK		Success. ppenumCATID contains a valid IEnumCATID enumerator.
E_OUTOFMEMORY		Insufficient memory to create an enumerator object to return.
E_INVALIDARG		ppenumCATID is an invalid pointer.
		others Registry manipulation or GUID-to-string routine errors.

3Registry Entries

Classes that wish to participate in the component category scheme will have one or both of the following registry keys, depending on whether the component implements or requires one or more component categories:

HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Implemented Categories
HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Required Categories

These keys have the same structure, and are thus described together below.

The Implemented/Required Categories key has no default value, and has as sub-keys GUIDs which are used as *category identifiers* (CATID) to uniquely identify the categories for the class. The sub-keys are named with a stringized GUID (CATID) and have no values (default or otherwise).

HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Implemented Categories = (default)[8] - “”
HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Implemented Categories\{...implemented catid1...} = (default) - “”
HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Implemented Categories\{...implemented catid2...} = (default) - “”[9]

HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Required Categories = (default) - “”
HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Required Categories\{...required catid1...} = (default) - “”
HKEY_CLASSES_ROOT\CLSID\{...clsid...}\Required Categories\{...required catid2...} = (default) - “”

User interfaces may require the ability to enumerate the list of categories available on a given machine. To facilitate this the following registry key is introduced:

HKEY_CLASSES_ROOT\Component Categories = (default)- “”

[8] The Registry supports a “default” named value on a given key. We use the notation “(default)” to indicate the default named value. The default named value usually (but not always) is nameless.

[9] Note that in earlier versions of this spec, these were *named values*, but are now *sub-keys.* This facilitates future extension by providing a place where class specific information can be placed for each category.

Each category registered on the machine has a sub-key of Component Categories that provides localized human-readable names for the category. Each component category key has as a named values localized string names for the category. The name for each named value is the locale ID in hexadecimal form (not decimal~~Hex~~).

```
HKEY_CLASSES_ROOT\Component Categories\{..catid...} = (default) - “”
                                = “locale ID 1” - “human-readable string (locale 1)”
                                = “locale ID 2” - “human-readable string (locale 2)”
```


Appendix A: Assigned GUIDs

The following range of 16 GUIDs has been generated for use by the Component Categories specification.

0002E000-0000-0000-C000-000000000046000018**0**-0000-0000-c000-000000000046 through 0002E0**FF**-0000-0000-C000-000000000046000018**F**-0000-0000-c000-000000000046

Any allocations from this range should be documented in this appendix.

GUID	Symbolic Name	Description
0002E000000018**0**-...	IID_IEnumGUID	
0002E011000018**1**-...	IID_IEnumCATEGORYINFO	
0002E012000018**2**-...	IID_ICatRegister	
0002E013000018**3**-...	IID_ICatInformation	
0002E004000018**4**-...	ComponentCategoryLib	Component Category Type Library
0002E005000018**5**-...	CLSID_StdComponentCategoriesMgr	

Index