How to get the COM concurrency model for the current thread.

dennisbabkin.com/blog

Intro

You might have seen many times the following APIs that were used to initialize the *concurrency model* for the Component Object Model, before the application starts using COM:

C++

```
CoInitialize(NULL);
```

or:
C++

```
CoInitializeEx(0, COINIT_MULTITHREADED);
```

or even a newer version:
C++

```
RoInitialize(RO_INIT_MULTITHREADED);
```

All these functions are required to be called once per thread to initialize COM and to specify the *concurrency model* for interactions between the COM objects in a thread. By specifying the *concurrency model* developers tell to the COM whether its objects will be used in a

single-threaded, or in a multi-threaded environment.

But how do you retrieve the current thread concurrency model if you need to know it later, or if you didn't initialize the process?

Internals

Surprisingly, Microsoft decided not to expose an API to retrieve the current COM thread concurrency, or the current "thread apartment model". Maybe they thought that there's no need for the programmers to know this, or they simply didn't care to document it. We will probably never know.

What is evident though is that the *COM apartment* is stored as an internal object, called `tagSOleTlsData`, pointer to which is placed into `TEB::ReservedForOle` for the thread, that is actually somewhat documented:

> To access the value of the ReservedForOle member, call CoGetContextToken.

The `TEB::ReservedForOle` is set to the pointer of the initialized `tagSOleTlsData` object when you call one of the `CoInitialize*` class of functions. Obviously, if you try to use any other COM functions for the thread where `TEB::ReservedForOle` is NULL, it will return an error `CO_E_NOTINITIALIZED`.

The `tagSOleTlsData` object can be gleaned from the Microsoft public symbols to be the following in the current version of Windows 10:

C++

```
struct tagSOleTlsData {
        void * pvThreadBase;
        CSmAllocator * pSmAllocator;
        ULONG dwApartmentID;
        ULONG dwFlags;
        LONG TlsMapIndex;
        void * * ppTlsSlot;
        ULONG cComInits;
        ULONG cOleInits;
        ULONG cCalls;
        ServerCall * pServerCall;
        ThreadCallObjectCache * pCallObjectCache;
        tagContextStackNode * pContextStack;
        CObjServer * pObjServer;
        ULONG dwTIDCaller;
        void * pCurrentCtxForNefariousReaders;
        CObjectContext * pCurrentContext;
        CObjectContext * pEmptyCtx;
        ULONGLONG ContextId;
        CComApartment * pNativeApt;
        IUnknown * pCallContext;
        CCtxCall * pCtxCall;
        CPolicySet * pPS;
        void * pvPendingCallsFront;
        void * pvPendingCallsBack;
        CAptCallCtrl * pCallCtrl;
        CSrvCallState * pTopSCS;
        IMessageFilter * pMsgFilter;
        HWND__ * hwndSTA;
        LONG cORPCNestingLevel;
        ULONG cDebugData;
        _GUID LogicalThreadId;
        void * hThread;
        void * hRevert;
        IUnknown * pAsyncRelease;
        HWND__ * hwndDdeServer;
        HWND__ * hwndDdeClient;
        ULONG cServeDdeObjects;
        void * pSTALSvrsFront;
        HWND__ * hwndClip;
        IDataObject * pDataObjClip;
        ULONG dwClipSeqNum;
        ULONG fIsClipWrapper;
        IUnknown * punkState;
        ULONG cCallCancellation;
        ULONG cAsyncSends;
        CAsyncCall * pAsyncCallList;
        CSurrogatedObjectList * pSurrogateList;
        void * pRWLockTlsEntry;
        CallEntryBuffer CallEntry;
        tagInitializeSpyNode * pFirstSpyReg;
        tagInitializeSpyNode * pFirstFreeSpyReg;
```

```
        CVerifierTlsData * pVerifierData;
        ULONG dwMaxSpy;
        UCHAR cCustomMarshallerRecursion;
        void * pDragCursors;
        IUnknown * punkError;
        ULONG cbErrorData;
        tagOutgoingCallData outgoingCallData;
        tagIncomingCallData incomingCallData;
        tagOutgoingActivationData outgoingActivationData;
        ULONG cReentrancyFromUserAPC;
        ASTAWaitContext * pASTAWaitContext;
        ULONG volatile dwCrossThreadFlags;
        ULONG dwNestedRemRelease;
        ULONG cIncomingTouchedASTACalls;
        PushLogicalThreadId * pTopPushedLogicalThreadId;
        ULONG iXslockOwnerTableHint;
        OLETLS_PREVENT_RUNDOWN_MITIGATION currentPreventRundownMitigation;
        INT fOweForcedBulkUpdateForCurrentMitigation;
        IUnknown * pClipboardBroker;
        ULONG dwActivationType;
        ULONG cTouchedAstasInActiveCall;
        ULONGLONG * pTouchedAstasInActiveCall;
        UnmarshalForQueryInterface * pTopmostUnmarshalForQueryInterface;
        CoGetStandardMarshalInProgress * pTopmostCoGetStandardMarshalInProgress;
        void tagSOleTlsData( tagSOleTlsData && );
        void tagSOleTlsData( tagSOleTlsData const & );
        tagSOleTlsData & operator=( tagSOleTlsData && );
        tagSOleTlsData & operator=( tagSOleTlsData const & );
};
```

Then `tagSOleTlsData` struct itself contains the `pNativeApt` member that points to an instance of a somewhat large `CComApartment` class:

C++

```
class CComApartment : public Object<ObjectOptions<ComReferenceCountingOptions<65>,
        AllocationOptions<PrivMemAllocator,0,Details::Nil>,
        ObjectNoPolymorphism,
        ObjectNoDebugging,
        ObjectNoAssociations>,
        CComApartment,
        Details::Nil>
{
public:
        void CComApartment( CComApartment const & );
        void CComApartment( APTKIND );
        HRESULT FinalConstruct();
        ULONG GetAptId();
        BOOLEAN IsOXIDEntryPresent();
        ULONGLONG GetUi64ApartmentIdentifier();
        HRESULT InitRemoting();
        HRESULT CleanupRemoting();
        HRESULT RestartServer();
        HRESULT StartServer();
        HRESULT StopServer();
        HRESULT FinishAsyncCalls();
        HRESULT StartServerExternal();
        HRESULT GetPreRegMOID( _GUID * );
        HRESULT FreePreRegMOID( _GUID const & );
        void CanRundownOIDs( ULONG , ULONGLONG * , tagRUNDOWN_RESULT * );
        HRESULT GetRemUnk( IRemUnknownN * * );
        HRESULT GetOXIDEntry( OXIDEntry * * );
        UnreferencedPtr<OXIDEntry> GetOXIDEntry() const;
        HRESULT GetOXIDEntryInitializeIfNeeded( OXIDEntry * * );
        void STAAssertNotCalledCrossThread();
        HRESULT STAPostGipRevoke( CGIPMessageParam * );
        HRESULT STAPostReleaseRifRef( IMessageParam * );
        HRESULT STAPostWinrtAsyncResponse( IMessageParam * );
        HRESULT STAPostWinrtAsyncServerContinuationMessage( IMessageParam * );
        BOOLEAN ClassicSTARemoveMessage( UINT , IMessageParam * );
        HRESULT ClassicSTAPostMessage( UINT , IMessageParam * );
        HRESULT ClassicSTASendMessage( UINT , IMessageParam * );
        ClassicSTAState & GetClassicSTAState();
        INT IsInited();
        BOOLEAN IsClassicSTA() const;
        BOOLEAN IsApplicationSTA() const;
        BOOLEAN IsAnySTA() const;
        BOOLEAN IsNA() const;
        BOOLEAN IsServerStarted() const;
        APTKIND GetApartmentKind() const;
        void AssertValid();
        void AssertNotCalledCrossApartment();
        void ASTASetMessageDispatcher( IMessageDispatcher * );
        void ASTAGetMessageDispatcher( IMessageDispatcher * * );
        ASTAIncomingCallList ASTAGetQueuedCallsNowDispatchable();
        ASTAIncomingCallList ASTAGetDispatchableCallsNowQueued();
        HRESULT ASTAPostCall( ServerCall * );
```

```
        void ASTAPostGipRevoke( CGIPMessageParam * );
        void ASTAPostReleaseRifRef( IMessageParam * );
        void ASTAPostSignal( CChannelObject * );
        void ASTAPostWinrtAsyncResponse( IMessageParam * );
        void ASTAPostWinrtAsyncServerContinuationMessage( IMessageParam * );
        void ASTAHandleQueuedCallTimeoutsAndRelease( ASTAIncomingCallList & );
        BOOLEAN ASTAHandlingPriorityEvents();
        ASTAState & GetASTAState();
        void * GetASTAWakeEvent();
        void * GetFreeUnusedDllsTimer();
        void IncrementPendingGitRegistrations();
        void DecrementPendingGitRegistrations();
        void SignalPendingGitRegistrationEvent();
        void WaitForPendingGitRegistrations();
        static BOOLEAN ASTAIsAlwaysDispatchableMethod( _GUID const & , ULONG );
        void ASTAClearPriorityEventPending();
        void ASTAClearDispatchableCallPending();
        void ASTASetDispatchableCallPending();
        void ASTAHandleMessage( IMessageParam * );
        void ASTAFlushPendingWork();
        HRESULT RegisterShutdownCallback( ComPtr<IApartmentShutdown> ,
APARTMENT_SHUTDOWN_REGISTRATION_COOKIE__ * * );
        HRESULT UnregisterShutdownCallback( APARTMENT_SHUTDOWN_REGISTRATION_COOKIE__
* );
        void NotifyApartmentShutdownCallbacks();
        void DeleteApartmentShutdownCallbacks();
private:
        void SetOXIDEntry( OXIDEntry * );
        void ClassicSTAQueueMessage( UINT , IMessageParam * );
        void ASTAWakeThreadIfNecessary();
        void ASTASetPriorityEventPending();
        static ULONG ApartmentIdForApartmentKind( APTKIND );
        static ULONGLONG ui64ApartmentIdentifierForApartmentKind( APTKIND );
        void ~CComApartment();
        INT IsStopPending();
        HRESULT WaitForAccess();
        void CheckForWaiters();
        HRESULT CallTheResolver();
        INT IsOidInPreRegList( ULONGLONG const & );
        INT IsOidInReturnList( ULONGLONG const & );
        INT IsInitializedAndNotUninitializing();
        ULONG _cRefs;
        ULONG _dwState;
        APTKIND const _AptKind;
        ULONG const _AptId;
        OXIDEntry * _pOXIDEntry;
        CRemoteUnknown * _pRemUnk;
        LONG _cWaiters;
        void * _hEventOID;
        ULONG _cPreRegOidsAvail;
        ULONGLONG _arPreRegOids[0x14];
        ULONG _cOidsReturn;
```

```
        ULONGLONG _arOidsReturn[0x14];
        ClassicSTAState * _pClassicSTAState;
        ASTAState * _pASTAState;
        void * _hEventASTAWake;
        void * _hTimerFreeUnusedDlls;
        LONG _cPendingGitRegistrations;
        void * _hEventPendingGitRegistrations;
        ComPtr<IWeakReference> _spMessageDispatcher;
        tagSOleTlsData * _pTlsSTA;
        ComPtr<ApartmentShutdownNode> _spFirstShutdownCallback;
        ComPtr<ApartmentShutdownNode> _spSentinelForNotify;
        ULONGLONG _ui64ApartmentIdentifier;
        static ULONG volatile s_ulApartmentIdentifierCounter;
        static COleStaticMutexSem s_ShutdownRegLock;
        void * __vecDelDtor( UINT );
};
```

And the `pCurrentContext` member points to an instance of the `CObjectContext` class, that is roughly declared as such:

C++

```
class CObjectContext : public IObjContext,
        public IMarshalEnvoy,
        public IMarshal,
        public IComThreadingInfo,
        public IContextCallback,
        public IAggregator,
        public IGetContextId
{
public:
        virtual HRESULT QueryInterface( _GUID const & , void * * );
        virtual ULONG AddRef();
        virtual ULONG Release();
        virtual HRESULT GetUnmarshalClass( _GUID const & , void * , ULONG , void * ,
ULONG , _GUID * );
        virtual HRESULT GetMarshalSizeMax( _GUID const & , void * , ULONG , void * ,
ULONG , ULONG * );
        virtual HRESULT MarshalInterface( IStream * , _GUID const & , void * , ULONG
, void * , ULONG );
        virtual HRESULT UnmarshalInterface( IStream * , _GUID const & , void * * );
        virtual HRESULT ReleaseMarshalData( IStream * );
        virtual HRESULT DisconnectObject( ULONG );
        virtual HRESULT GetEnvoyUnmarshalClass( ULONG , _GUID * );
        virtual HRESULT GetEnvoySizeMax( ULONG , ULONG * );
        virtual HRESULT MarshalEnvoy( IStream * , ULONG );
        virtual HRESULT UnmarshalEnvoy( IStream * , _GUID const & , void * * );
        virtual HRESULT SetProperty( _GUID const & , ULONG , IUnknown * );
        virtual HRESULT RemoveProperty( _GUID const & );
        virtual HRESULT GetProperty( _GUID const & , ULONG * , IUnknown * * );
        virtual HRESULT EnumContextProps( IEnumContextProps * * );
        virtual HRESULT Freeze();
        virtual HRESULT DoCallback( HRESULT (* )( void * ), void * , _GUID const & ,
UINT );
        virtual HRESULT SetContextMarshaler( IContextMarshaler * );
        virtual HRESULT GetContextMarshaler( IContextMarshaler * * );
        virtual HRESULT SetContextFlags( ULONG );
        virtual HRESULT ClearContextFlags( ULONG );
        virtual HRESULT GetContextFlags( ULONG * );
        virtual HRESULT GetCurrentApartmentType( _APTTYPE * );
        virtual HRESULT GetCurrentThreadType( _THDTYPE * );
        virtual HRESULT GetCurrentLogicalThreadId( _GUID * );
        virtual HRESULT SetCurrentLogicalThreadId( _GUID const & );
        virtual HRESULT ContextCallback( HRESULT (* )( tagComCallData * ),
tagComCallData * , _GUID const & , INT , IUnknown * );
        virtual HRESULT Aggregate( IUnknown * );
        _GUID GetContextId();
        virtual HRESULT GetContextId( _GUID * );
        HRESULT InternalQueryInterface( _GUID const & , void * * );
        ULONG InternalAddRef();
        ULONG InternalRelease();
        virtual HRESULT FreezeWithApartmentSet();
        virtual HRESULT InternalContextCallback( HRESULT (* )( void * ), void * ,
_GUID const & , INT , IUnknown * );
```

```
        HRESULT Reset( void * * );
        ULONG GetCount();
        tagContextProperty * GetNextProperty( void * * );
        CContextPropList * GetPropertyList();
        HRESULT MarshalPropertyHeader( IStream * & , _GUID const & ,
tagContextProperty * , _ULARGE_INTEGER & , _ULARGE_INTEGER & );
        HRESULT MarshalEnvoyProperties( ULONG & , tagContextProperty * & , IStream *
, ULONG );
        HRESULT MarshalProperties( ULONG & , tagContextProperty * & , IStream * ,
_GUID const & , void * , ULONG , void * , ULONG );
        HRESULT ReadStreamHdrAndProcessExtents( IStream * , tagCONTEXTHEADER & );
        INT AlignStream( IStream * & );
        INT PadStream( IStream * & );
        HRESULT SetStreamPos( IStream * & , ULONGLONG , _ULARGE_INTEGER * const & );
        HRESULT AdvanceStreamPos( IStream * & , ULONGLONG , _ULARGE_INTEGER * const &
);
        HRESULT GetStreamPos( IStream * & , _ULARGE_INTEGER * const & );
        INT PropOkToMarshal( tagContextProperty * & , ULONG );
        HRESULT GetPropertiesSizeMax( _GUID const & , void * , ULONG , void * , ULONG
, ULONG , INT , ULONG & );
        ULONGLONG GetId();
        CContextLife * GetLife();
        void CreateIdentity( IComObjIdentity * );
        void DestroyIdentity( IComObjIdentity * );
        INT IsUnsecure();
        INT IsFrozen();
        INT IsEnvoy();
        INT IsEmpty();
        INT IsDefault();
        INT IsStatic();
        INT IsDisconnectable();
        INT ExecutingDisconnect();
        INT IsDisconnecting();
        INT StartDisconnectIfNecessary();
        CCtxConnectionManager * GetConnectionManager();
        void NotifyServerException( _EXCEPTION_POINTERS * );
        void NotifyContextAbandonment();
        SHashChain * GetPropHashChain();
        SHashChain * GetUUIDHashChain();
        static CObjectContext * HashPropChainToContext( SHashChain * );
        static CObjectContext * HashUUIDChainToContext( SHashChain * );
        tagSPSCache * GetPSCache();
        CComApartment * GetComApartment();
        static void Initialize();
        static void Cleanup();
        static CObjectContext * CreateObjectContext( ULONG );
        static CObjectContext * CreateAndFreezeDefaultObjectContext( CComApartment *
);
        static HRESULT CreateEnvoyObjectContext( tagDATAELEMENT * , CObjectContext *
& );
        HRESULT GetEnvoyData( tagDATAELEMENT * * );
        HRESULT SetEnvoyData( tagDATAELEMENT * );
```

```
        void SetContextId( _GUID const & );
        ULONG GetHashOfId();
        static HRESULT CreatePrototypeContext( CObjectContext * , CObjectContext * *
);
        void InPropTable( INT );
        INT IsInPropTable();
        static void * operator new( ULONGLONG );
        static void operator delete( void * );
        static HRESULT CreateUniqueID( _GUID & );
        ULONG CommonRelease();
        HRESULT QIHelper( _GUID const & , void * * , INT );
        void CleanupTables();
        static HRESULT IsValidContext( CObjectContext * );
        void CObjectContext( CObjectContext const & );
private:
        void CObjectContext( ULONG , _GUID const & );
        void ~CObjectContext();
        ULONG _cRefs;
        LONG _cUserRefs;
        LONG _cInternalRefs;
        ULONG _dwFlags;
        SHashChain _propChain;
        SHashChain _uuidChain;
        tagInterfaceData * _pifData;
        ULONG _MarshalSizeMax;
        CComApartment * _pApartment;
        ULONG _dwHashOfId;
        _GUID _contextId;
        ULONGLONG _urtCtxId;
        tagSPSCache _PSCache;
        IMarshal * _pMarshalProp;
        LONG _cReleaseThreads;
        CContextPropList _properties;
        IUnknown * _pMtsContext;
        CContextLife * volatile _pContextLife;
        CCtxConnectionManager * _pConnectionMgr;
        static CPageAllocator s_CXAllocator;
        static INT s_fInitialized;
        static ULONG s_cInstances;
public:
        CObjectContext & operator=( CObjectContext const & );
private:
        void * __vecDelDtor( UINT );
};
```

The undocumented `tagSOleTlsData` object is used pretty much for most calls within other COM functions, and thus obviously has to be created before any other COM object. A call to the `CoInitialize*`-class of functions creates this object in memory. It has an internal reference counter, so any subsequent calls to `CoInitialize*` / `CoUninitialize` will technically `AddRef` / `Release` that internal counter. And the `tagSOleTlsData` object will be destroyed when its reference count reaches zero.

That is why Microsoft has this to say in the documentation:

> ... each successful call to CoInitialize or CoInitializeEx, including any call that returns S_FALSE, must be balanced by a corresponding call to CoUninitialize.

To retrieve the reference to `CObjectContext` from the `TEB`, Microsoft instructs developers to use the `CoGetContextToken` function.

The question though is why is it declared as this:

C++

```
HRESULT CoGetContextToken(
        ULONG_PTR *pToken
);
```

and not as this?

C++

```
HRESULT CoGetContextToken(
        IObjContext** ppObjCtx
);
```

Which would be more logical, if you think about it.

The Resulting Function

And finally, here's the function to retrieve the current COM *apartment model*, or the COM *concurrency model* for the current thread:

C++

```
HRESULT GetCoConcurrencyModel(APTTYPE* pOutType)
{
        HRESULT hr;

        union {
                ULONG_PTR Token;
                IObjContext* pObjCtx;
        };

        APTTYPE AptType = APTTYPE_CURRENT;

        if (0 <= (hr = CoGetContextToken(&Token)))
        {
                IComThreadingInfo* pComInfo;

                if (0 <= (hr = pObjCtx->QueryInterface(IID_PPV_ARGS(&pComInfo))))
                {
                        hr = pComInfo->GetCurrentApartmentType(&AptType);

                        pComInfo->Release();
                }

                // pComInfo->Release();         //Note that we DO NOT need to call
Release as CoGetContextToken does not do AddRef internally!
        }

        if (pOutType)
                *pOutType = AptType;

        return hr;
}
```

Our `GetCoConcurrencyModel` function will return a result code that must be compared with the SUCCEEDED macro for correctness. And if it is correct, then the `pOutType` variable will receive the current thread apartment model as the `APTTYPE`.

> Note that Microsoft did not declare the APTTYPE correctly. It should have been declared as such:
> C++
>
> ```
> typedef enum _APTTYPE
> {
>         APTTYPE_CURRENT = -1,
>         APTTYPE_STA     = 0,                    //A single-threaded apartment
>         APTTYPE_MTA     = 1,                    //A multithreaded apartment
>         APTTYPE_NA      = 2,                    //A neutral apartment
>         APTTYPE_MAINSTA = 3                     //The main single-threaded
> apartment
> } APTTYPE;
> ```