Helper Classes

The CTI OS Client Interface Library uses several custom data structures. This chapter describes the CTI OS Helper Classes (data structures). The following helper classes are distributed with the Client Interface Library:

- Arg. The Arg structure is the basic data type the CIL uses for any parameter included in methods or events. Objects of this type allow the CIL to be fully extensible and reusable. Arg supports many useful types including string, integer, Boolean, and Arguments array. Arg is the base class for the Arguments class. In most programming scenarios, programmers do not use Arg directly, but indirectly through the Arguments class.
- Arguments. You use the Arguments structure to maintain and send a set of key-value pairs between the CIL and CTI OS Server for events and requests. The Arguments array elements must all be of type Arg. The Arguments structure enables future growth of the CTI OS feature set, without requiring changes to the method call signature.
- CilRefArg. The CilRefArg class is a specialized subclass of Arg. It stores a reference to an object derived from CCtiOsObject (C++ only). For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CCtiOsSession, or CWaitObject.
- CCtiosException. The CTI OS uses CCtiosException class to provide detailed information when an exception occurs (C++ and Java only). When an exception is caught as CCtiosException, the programmer can query it for details such as error codes and error messages.
- CWaitObject. CWaitObject is a CIL object that derives from CtiOsObject. It is a utility class (available in all CILs except COM) that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. You create wait objects with the CreateWaitObject Session Object method and destroy them with the DestroyWaitObject Session Object method.
- Logger. The Logger class creates a Logger object and a LogManager object, if one does not already exist. Any object that needs to perform logging must instantiate the Logger class. The Logger class communicates with the singleton LogManager object, which acts as the centralized logging facility. The Logger class also defines tracing constants.
- LogWrapper. The LogWrapper class provides a default Logging mechanism. By default, the LogWrapper traces to the console. If you create the LogWrapper with a filename, then it traces to that file.

Arg Class

The Arg is a generic class used in parameters or return values in CIL methods. Information sent by CTI OS server to the CIL in an event is packed in an Arguments object where each element of the array is an object of type Arg. An Arg object's absolute data type is determined by the type of data it stores. The basic types an object can store are identified by the enumerated constants in Table 2: enumArgTypes, on page 7.

Arg class methods do conversion between types whenever possible. For example, you can do a SetValue(25) and then do a GetValueString(), which returns the string “25”. You can also do a SetValue(“25”) and then do a GetValueIntObj, which returns an Integer object containing the numeric value 25. However, if you call SetValue “abc” and try to retrieve it as an int, it fails.

The following table lists the available Arg class methods.

Table 1: Arg Class Methods

Method	Description
AddRef	Increments the reference count for the data item.
Clone	Creates an exact copy of the Arg object.
CreateInstance	Creates an Arg object.
DumpArg	Builds a string containing the value stored in the Arg.
GetType	Returns the type of the data stored in the argument (one of the values in Table 2: enumArgTypes, on page 7).
GetValueInt GetValueUInt GetValueUInt GetValueUShort GetValueShort GetValueBool GetValueString	Returns the value stored in the argument.
SetValue	Sets the data in the Arg object.

In many scenarios, programmers stick to Arguments (see the preceding section), which wraps many Arg methods and encapsulates a collection of Arg objects.

AddRef

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time (for example, if you plan to store and use Arguments received in an event handler after the event thread has returned to the calling code). When you are finished with the item, you must call the Release method or a memory leak occurs.

Syntax

C++

```
unsigned long AddRef()
```

COM

```
HRESULT AddRef()
```

VB, Java, .NET

```
Not used
```

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

C++: The current reference count after the AddRef() call.

Clone

The Clone method allocates a new Arg in memory and copies its key, value, and type to the new instance. When using the C++ or COM CILs, it is important to release the object when it is no longer needed.

Syntax

C++

```
Arg & Clone()
```

COM

```
HRESULT Clone(/*[out, retval]*/ IArg** arg);
```

VB

```
Clone() as CTIOSCLIENTLib.IArg
```

Java

```
Arg Clone()
```

.NET

```
Ojbect Clone()
```

Output Parameters

arg

Pointer to an IArg instance that is a copy of the object.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: If successful, returns a reference to a new Arg object. If unsuccessful in C++ or VB, it throws a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED. If unsuccessful in Java, it returns null but does not throw an exception.

CreateInstance

The CreateInstance method creates an object of type Arg class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.

Syntax

C++

```
static Arg& CreateInstance();
 // static creation mechanism.
static Arg& CreateInstance(Arg& arg);
// static creation mechanism.
static bool CreateInstance(Arg ** arg);
// static creation mechanism,
// alternate version
```

COM

```
Wrapped by CoCreateInstance
```

VB

```
Wrapped by New
```

Java, .NET

```
Not available
```

Parameters

arg

(output) Pointer to the newly created Arg.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Either a reference to the newly created Arg or a boolean indicating method success. If the methods not returning bool are unsuccessful, they raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Remarks

This method increments the Arg's reference count, so do not call AddRef(). However, you must call Release() after you are finished with the Arg.

DumpArg

The DumpArg method builds a string containing the value stored in the Arg. This involves doing any type conversion required to display the data as a string. For example, it automatically converts an INTEGER type to a string that can be logged for debugging. In the event that a Arg object is actually an Arguments object, the string returned is the one built by Arguments.DumpArg, and thus enabled printing of nested Arguments structures.

Syntax

C++

```
string DumpArg()
```

COM

```
HRESULT DumpArg([out,retval] BSTR* arg_string);
```

VB

```
DumpArg() as String
```

Java, .NET

```
Not available. Use the ToString method.
```

Parameters

arg_string

The pointer to the string where the contents of the Arg object are written.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: A string containing the contents of the structure.

GetArgType (.NET Only)

The GetArgType method returns the type of the contained value. This returned value is one of the following:

- ARG NOTSET
- ARG_BOOL

- ARG_SHORT
- ARG_USHORT
- ARG_INT
- ARG_UINT

For more information about valid types, see Table 2: enumArgTypes, on page 7.

Syntax

C++, COM, Java

```
Use GetType.
```

.NET

```
ArgDataType GetArgType()
```

Parameters

None.

Returns

int code for the type of value contained in this Arg.

GetType

The GetType method returns the type of the data stored by the Arg. For more information about possible types, see the following table.

Syntax

C++

```
enumArgTypes GetType()
```

COM

```
HRESULT GetType(/*[out, retval]*/ int* type);
```

VB

```
GetType () as Integer
```

Java

```
int GetType(
```

.NET

```
Use the GetArgType method.
```

Output Parameters

type

Integer that receives the enumerated constant that identifies data type stored in IArg.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Returns the enumerated value that identifies the data type stored in the Arg (for more information, see the following table).

Table 2: enumArgTypes

Argument Type	Description
ARG_NOTSET	Argument type not determined
ARG_INT	Signed integer
ARG_UINT	Unsigned integer
ARG_USHORT	2 bytes unsigned integer
ARG_SHORT	2 bytes signed integer
ARG_BOOL	1 byte integer
ARG_STRING	Character string
ARG_ARGARRAY	Variable length Arguments array

GetValue Methods

The GetValue method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it. For more information about GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in CtiOs Object

Syntax

C++

```
int GetValueInt();

unsigned int GetValueUInt();

unsigned shortGetValueUShort();

short GetValueShort();

string& GetValueString();

bool GetValueBool();

bool GetValueInt(int * value);

bool GetValueUInt(unsigned int * value);

bool GetValueUShort(unsigned short * value);

bool GetValueShort( short * psVallue);

bool GetValueBool( bool * value);

bool GetValueString(string* value);
```

COM

```
HRESULT GetValue(/*[out, retval]*/ VARIANT* value);
```

VB

```
GetValue() as Variant

GetValue (key as String, value as Variant) as Boolean
```

Java

```
Integer GetValueIntObj()

Long GetValueUIntObj()

Short GetValueShortObj()

Integer GetValueUShortObj()

Boolean GetValueBoolObj()

String GetValueString()
```

.NET

```
System.Boolean GetValueInt(out System.Int32 nValue)

System.Boolean GetValueUInt(out System.Int64 nValue)

System.Boolean GetValueShort(out System.Int16 nValue)

System.Boolean GetValueUShort(out System.Int32 nValue)

System.Boolean GetValueBool(out System.Boolean bValue)

System.Boolean GetValueString(out System.String strValue)
```

Parameters

Value

Output parameter of the specified type containing the value of the Arg.

For COM, this value is of type VARIANT * whose type is one of the types listed in the following table.

Table 3: Variant Types Supported by GetValue (COM)

Variant Type	Standard C++ Type
VT_INT	Int
VT_UINT	Unsigned int
VT_I2	Short
VT_UI2	Unsigned short
VT_BOOL	Bool
VT_BSTR	string, const string and char *

Return Values

C++

Methods taking no parameters, if successful, return the value in the object; otherwise, they raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

The methods taking a pointer to the variable receiving the result return true, if the method was able to get the value, otherwise, false.

Java

Returns null if method failed.

.NET

Returns false if method failed.

COM

If the method was able to set the variant type of the value to any of the types listed in the above table, it returns the value in the appropriate field of the variant. Otherwise it returns VT_EMPTY.

Release

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak occurs.

Syntax

C++

```
unsigned long Release()
```

COM

```
HRESULT Release()
```

VB, Java, .NET

```
Not used
```

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

C++: The current reference count after the Release() call.

SetValue

The SetValue method sets the value in the Arg object.

Syntax

C++

```
bool SetValue( int value );
bool SetValue( unsigned int value );
bool SetValue( unsigned short value );
bool SetValue( short value );
bool SetValue( bool value );
bool SetValue( char * value );
bool SetValue( string& value);
bool SetValue( const string& value);
bool SetValue( Arg & value);
```

COM

```
HRESULT SetValue(/*[in]*/ VARIANT * pVariant, /*[out,retval]*/ VARIANT_BOOL * errorcode
);
```

VB

```
SetValue(value as Variant) as Boolean
```

Java

```
boolean SetValue(Arg rArg)
  boolean SetValue(int iVal)
  boolean SetValue(short nValue)
  boolean SetValue(String sValue)
  boolean SetValueUInt(long lValue)
  boolean SetValueUShort(int iValue
```

.NET

```
System.Boolean SetValue(System.Int32 iValue)
System.Boolean SetValueUInt(System.Int64 lValue)
System.Boolean SetValueUShort(System.Int32 iValue)
System.Boolean SetValue(System.Int16 nValue)
System.Boolean SetValue(System.Boolean bValue)
System.Boolean SetValue(System.String sValue)
System.Boolean SetValue(Arg rArg)
```

Parameters

value

The value of the specified type to assign to the Arg.

For COM, this value is of type VARIANT * whose type is one of the types listed in the following table.

Table 4: Supported Variant Types

Variant Type	Standard C++ Type
VT_INT	Int
VT_UINT	Unsigned int
VT_I2	Short
VT_UI2	Unsigned short

Variant Type	Standard C++ Type
VT_BOOL	Bool
VT_BSTR	string, const string and char *
VT_DISPATCH	Pointer to an IArg interface

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 1.

Return Values

C++

If the method was able to set the value it returns true, otherwise it returns false.

COM, VB

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

Java, .NET

This method returns true if the method succeeds, otherwise false.

Arguments Class

The Arguments structure (class) provides key/value support to form a collection of values. Each value stored in an Arguments structure is associated with a key. To add an item, use the *AddItem* or *SetValue* method and pass a key and a value. The key must be a string or an enumerated value, and the value can be almost any type (i.e. all types supported by Arg). To retrieve the item, use the appropriate GetValue method with a key, and the value is returned. Keys are not case sensitive, and leading and trailing spaces are always removed from the key.

Arguments also supports access by index. The index is useful for retrieving items sequentially, but may not be as fast as retrieval by key. The *Arguments* structure's index is 1-based, to provide easier support for Visual Basic programmers. Internally, the *Arguments* structure uses a binary tree and other techniques to provide fast access to any item. *Arguments* can support a virtually unlimited number of key-value pairs, and supports nested *Arguments* structure as well.

The following table lists the Arguments class methods.

Table 5: Arguments Class Methods

Method	Description
AddItem	Adds an item to an Arguments array.
AddRef	Increments the reference count for the data item.
Clear	Deletes all elements from an Arguments array.

Method	Description
Clone	Creates a copy of an Arguments array.
CreateInstance	Creates an Arguments array.
DumpArgs	Returns Arguments object as a string
GetElement (also GetElementInt, GetElementUInt, GetElementUShort, GetElementShort, GetElementBool, GetElementString, GetElementArg, GetElementKey GetElementArgType)	Returns the value stored under a specified index.
GetValue (also GetValueInt, GetValueUShort, GetValueShort, GetValueBool, GetValueUInt, GetValueString, GetValueArray, GetValueArg)	Returns the value stored under a specified key.
IsValid	Tests if a key is present in the current Arguments array.
NumElements	Returns the number of arguments in the current Arguments array,.
Release	Decrements the reference count for the data item.
RemoveItem	Removes an item from an Arguments array.
SetElement	Sets the value of an index.
SetValue	Sets the value of a key.

Usage Notes

When writing an application using the CTI OS SDK, the following sequence of steps in the program can produce a problem:

- Programmer passes an Arguments array into a CTI OS SDK method (methodA)
- MethodA returns
- Programmer modifies the same Arguments array
- Programmer passes the modified Arguments array into another CTI OS SDK method (methodB)

When running the application, the call to methodA can behave as if it was passed the modified Arguments array. This is because many CTI OS methods simply place a pointer to the Arguments array on a queue of items to send to CTI OS server. When the same Arguments array is later modified, as in the preceding example, the pointer on the queue now points to the modified array and the modified array is sent to CTI OS server. A problem can occur depending on timing, as there are multiple threads involved: the thread pulling items off

the queue and the thread modifying the Arguments array. If the queued message is sent to CTI OS before the Arguments array is modified, the problem does not occur.

To avoid this problem, call the Clone method on the initial Arguments array and modify the copy rather than modifying the original. For example, the preceding example would change as follows:

- Programmer passes an Arguments array (initialArray) into a CTI OS SDK method (methodA)
- MethodA returns
- modifiedArray = initialArray.Clone()
- Programmer modifies modifiedArray
- Programmer passes the modifiedArray into another CTI OS SDK method (methodB)

AddItem (C++ COM VB Only)

The AddItem method expects a key-value pair. The key value can be a string or an integer. The value can be a string, an integer, or an object reference. If there is an entry with the same key, it is replaced with this entry, otherwise the new key-value pair is added to the Arguments array. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++

```
bool AddItem( std::string & key, int value );
bool AddItem( std::string& key, unsigned int value );
bool AddItem( std::string& key, unsigned short value );
bool AddItem( std::string& key, short value );
bool AddItem( std::string& key, bool value );

bool AddItem( std::string& key, char * pchar );
bool AddItem( std::string& key, std::string& value );
bool AddItem( std::string& key, Arg& value );
bool AddItem( std::string& key, const Arg& value );
bool AddItem( std::string& key, Arguments& value );
bool AddItem( std::string& key, const Arguments& value);

bool AddItem( char * key, int value );
bool AddItem( char * key, unsigned int value );
bool AddItem( char * key, unsigned short value );
bool AddItem( char * key, short value );
bool AddItem( char * key, bool value );

bool AddItem( char * key, char * value   );
bool AddItem( char * key, std::string& value );
bool AddItem( char * key, Arg& cArg );
bool AddItem( char * key, const Arg& value );
bool AddItem( char * key, Arguments& value );
bool AddItem( char * key, const Arguments& value );

bool AddItem( enum_Keywords key, int value );
bool AddItem( enum_Keywords key, unsigned int value );
bool AddItem( enum_Keywords key, unsigned short value );
bool AddItem( enum_Keywords key, short value );
bool AddItem( enum_Keywords key, bool value );

bool AddItem( enum_Keywords key, char * value );
bool AddItem( enum_Keywords key, std::string& value );
bool AddItem( enum_Keywords key, Arg& cArg );
bool AddItem( enum_Keywords key, const Arg& value );
bool AddItem( enum_Keywords key, Arguments& value );
bool AddItem( enum_Keywords key, const Arguments& value)
```

COM

```
HRESULT AddItem(/*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value, /*[out,retval]*/
VARIANT_BOOL success) As Boolean;
```

VB

```
AddItem( Key as Variant, Value as Variant)
```

Java, .NET

```
Not Applicable. Use the SetValue method.
```

Parameters

key

Key name for the item to be added.

value

Value of the item to be added.

success

An output parameter (return parameter in C++ and VB) that contains a boolean indicating success or lack thereof.

Return Value

C++: Returns True in if the entry was successfully added, otherwise False.

COM and VB: Standard return values are valid; For more information, see CIL Coding Conventions.

AddRef (C++ and COM Only)

The AddRef method increments the reference count for the data item. You must call this if you are storing a pointer to the item for some time. When you are finished with the item, you must call the Release method or a memory leak occurs.

Syntax

C++

```
unsigned long AddRef()
```

COM

```
HRESULT AddRef()
```

VB, Java, .NET

```
Not used
```

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

C++: Current reference count.

Others: None.

Clear

The Clear method deletes all the elements from Arguments object.

Syntax

C++

```
void Clear()
```

COM

```
HRESULT Clear()
```

VB

```
Clear()
```

Java, .NET

```
void Clear()
```

Parameters

None.

Return Value

None.

Clone

The Clone method creates a copy of the Arguments structure. Because in C++ this method is implemented in the base class (Arg), it returns a reference to an Arg, but this is actually a reference to an Arguments array. Therefore, it is necessary to cast the return value of this method. The following C++ code sample shows this casting:

```
Arguments & argsCopy = (Arguments&) argsOrig.Clone ();
```

To cast in VB, do the following:

```
Dim Args As CTIOSCLIENTLib.IArgumentsSet Args = Orig.Clone()
```

Syntax

C++

```
Arg & Clone()
```

COM

```
HRESULT Clone(/*[out, retval]*/ IArguments ** args);
```

VB

```
Clone() as CTIOSCLIENTLib.IArguments
```

Java

```
Arg Clone()
```

.NET

```
object Clone()
```

Parameters

args

An output parameter containing a pointer to an Arguments array that is a copy of the object.

Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: A reference to the Arg structure that is a copy of the object.

CreateInstance (C++ and COM Only)

The CreateInstance method creates an object of type Arguments class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.

Syntax

C++

```
static Arguments & CreateInstance()
static bool CreateInstance(Arguments ** args)
```

COM

```
Not exposed, called by CoCreateInstance.
```

VB

```
Not exposed, called by New.
```

Java, .NET

```
Not implemented.
```

Parameters

args

A pointer to the newly created Arguments structure.

Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Either a reference to the newly created Arguments structure or a boolean indicating method success.

Remarks

C++, COM: Internally this method increments the Arg's reference count, so do not call AddRef(). However, you must call Release() when you are finished with the Arg.

DumpArgs

The DumpArgs method builds a string showing all of the members of the Arguments structure in the form “key1 = value1; key2 = value2;...”. It is primarily used for debugging.

Note The trace mask must be set in order to run the DumpArgs method on the Arguments array of call variables.

For information on setting the trace level see, Set Trace Levels (COM and C++).

Syntax

C++

```
string DumpArgs()
```

COM

```
HRESULT DumpArgs([out,retval] BSTR* arg_string);
```

VB

```
DumpArgs() as String
```

Java, .NET

```
string DumpArgs()
```

Parameters

arg_string

The pointer to the string containing the contents of the Arguments array listing all of the key/value pairs in the format of “key1 = value1; key2 = value2;...”.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: A string containing the contents of the Arguments array listing all key/value pairs.

GetElement Methods

The GetElement method is similar to GetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The Index is 1-based. The Index should never be less than one or greater than NumElements. see also NumElements method. The GetElementKey returns the key of a given index.

Syntax

C++

```
Arg& GetElement( int index );
bool GetElement( int index, Arg ** value);
int GetElementInt( int index );
bool GetElementInt( int index, int * value);
unsigned int GetElementUInt( int index );
bool GetElementUInt( int index, unsigned int * value);
unsigned short GetElementUShort( int index );
bool GetElementUShort( int index, unsigned short * value );
short GetElementShort( int index );
bool GetElementShort( int index, short * value);
bool GetElementBool( int index );
bool GetElementBool( int index, bool * value);
std::string GetElementString( int index );
bool GetElementString( int index, std::string * value);
Arguments& GetElementArg( int index );
bool GetElementArg( int index, Arguments ** key);
std::string GetElementKey( int index );
bool GetElementKey( int nIndex, std::string * key);
bool GetElementKey( int nIndex, int * key);
```

COM

```
HRESULT GetElementKey(/*[in]*/ int index, /*[out]*/ BSTR *
key);
HRESULT GetElement(/*[in]*/ int index, /*[out]*/ VARIANT *
value);
```

VB

```
GetElement (Integer index, Variant value)
GetElement (Integer index, String key)
```

Java

```
Arg GetElement(int iIndex)
Arguments GetElementArguments(int iIndex)
Integer GetElementIntObj(int iIndex)
Long GetElementUIntObj(int iIndex)
Short GetElementShortObj(int iIndex)
Integer GetElementUShortObj(int iIndex)
Boolean GetElementBoolObj(int iIndex)
String GetElementString(int iIndex)
String GetElementKey(int iIndex)
```

.NET

```
Boolean GetElement(System.Int32 iIndex, out Arg obArg)
Boolean GetElementInt(System.Int32 iIndex, out System.Int32
iValue )
Boolean GetElementUInt(System.Int32 iIndex, out System.Int64
nValue)
Boolean GetElementUShort(System.Int32 iIndex, out System.Int32
nValue)
Boolean GetElementShort(System.Int32 iIndex, out System.Int16
nValue)
Boolean GetElementBool(System.Int32 iIndex, out System.Boolean
bValue)
Boolean GetElementString(System.Int32 iIndex, out System.String
strValue)
Boolean GetElementArguments(System.Int32 iIndex, out Arguments
argArguments)
Boolean GetElementKey(System.Int32 iIndex, out System.String
strKey)
```

Parameters

value

An output parameter containing the value of the member at the specified index.

key

An output parameter containing the key of the member at the specified index.

index

An input parameter containing an index into the Arguments array.

Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Returns either the value at the index specified independently from its key, or a boolean indicating success or failure.

GetValue Methods

The GetValue method returns the value stored under a key. You can test the existence of a key using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key. For more information about GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in CtiOs Object

Syntax

C++

```
Arg& GetValue( enum_Keywords eKey );
bool GetValue( enum_Keywords key, Arg ** value );
Arg& GetValue( std::string& key);
bool GetValue( std::string& key, Arg ** value);
Arg& GetValueArg( std::string& key);
bool GetValueArg( std::string& key, Arg ** value);
int GetValueInt( enum_Keywords key); /*throws exception*/
bool GetValueInt( enum_Keywords key, int * value);
unsigned int GetValueUInt( enum_Keywords key);
bool GetValueUInt( enum_Keywords key, unsigned int * value);
unsigned short GetValueUShort( enum_Keywords key);
bool GetValueUShort( enum_Keywords key, unsigned short * value);
short GetValueShort( enum_Keywords key);
bool GetValueShort( enum_Keywords key, short * value);
bool GetValueBool( enum_Keywords key);
bool GetValueBool( enum_Keywords key, bool * value);
std::string GetValueString( enum_Keywords key);
bool GetValueString( enum_Keywords key, std::string * value);
int GetValueInt( std::string& key); /*throws exception*/
bool GetValueInt( std::string& key , int * value);
unsigned int GetValueUInt( std::string& key  );
bool GetValueUInt( std::string& key , unsigned int * value);
unsigned short GetValueUShort( std::string& key  );
bool GetValueUShort( std::string& key , unsigned short * value);
short GetValueShort( std::string& key  );
bool GetValueShort( std::string& key , short * value);
bool GetValueBool( std::string& key  );
bool GetValueBool( std::string& key , bool * value);
std::string GetValueString( std::string& key  );
bool GetValueString( std::string& key , std::string * value);
Arguments& GetValueArray( std::string& key  );
bool GetValueArray( std::string& key , Arguments ** value);
Arguments& GetValueArray( enum_Keywords key  );
bool GetValueArray( enum_Keywords key , Arguments ** value);
Arg& GetValue( char * key );
bool GetValue( char * key, Arg ** value);
Arguments& GetValueArray( char * key );
bool GetValueArray( char * key, Arguments ** value);
int GetValueInt( char * key );
bool GetValueInt( char * key, int * value);
unsigned int GetValueUInt( char * key );
bool GetValueUInt( char * key, unsigned int * value);
unsigned short GetValueUShort( char * key );
bool GetValueUShort( char * key, unsigned short * value);
short GetValueShort( char * key );
bool GetValueShort( char * key, short * value);
bool GetValueBool( char * key );
bool GetValueBool( char * key, bool * value);
std::string GetValueString( char * key );
bool GetValueString( char * key, std::string * value);
Arg& GetValueArg( char * key );
bool GetValueArg( char * key, Arg ** value);
```

COM

```
HRESULT GetValue(/*[in]*/ BSTR key, /*[out, retval]*/ VARIANT * pVvalue);
HRESULT GetValueInt(/*[in]*/ VARIANT *key, /*[out, retval]*/ int *value);
HRESULT GetValueString(/*[in]*/ VARIANT *key, /*[out, retval]*/ BSTR *value);
HRESULT GetValueArray(/*[in]*/ VARIANT *key, /*[out, retval]*/ IArguments **pArguments);
HRESULT GetValueBool(/*[in]*/ VARIANT *key, /*[out, retval]*/ VARIANT_BOOL * value);
```

VB

```
GetValue (Key as String) as Variant
GetValue(key As Variant) As Arg
GetValueArray(key As Variant) As Arguments
GetValueBool(key As Variant) As Boolean
GetValueInt(key As Variant) As Long
GetValueString(key As Variant) As String
```

Java

```
Arg GetValue(int iKey)
Arg GetValue(String sKey)
Arguments GetValueArray(int iKey)
Arguments GetValueArray(String sKey)
Integer GetValueIntObj(int iKey)
Integer GetValueIntObj(String sKey)
Long GetValueUIntObj(int iKey)
Long GetValueUIntObj(String sKey)
Short GetValueShortObj(int iKey)
Short GetValueShortObj(String sKey)
Integer GetValueUShortObj(int iKey)
Integer GetValueUShortObj(String sKey)
Boolean GetValueBoolObj(int iKey)
Boolean GetValueBoolObj(String sKey)
String GetValueString(int iKey)
String    GetValueString(String sKey)
```

.NET

```
Boolean GetValue(System.String sKey, out Arg obArg)
Boolean GetValueInt(System.String sKey, out System.Int32 nValue)
Boolean GetValueUInt(System.String sKey, out System.Int64 nValue )
Boolean GetValueShort(System.String sKey, out System.Int16 nValue)
Boolean GetValueUShort(System.String sKey,out System.Int32 nValue)
Boolean GetValueBool(System.String sKey, out System.Boolean bValue)
Boolean GetValueString(System.String sKey, out System.String strValue)
Boolean GetValueArray(System.String sKey, out Arguments arArguments)
Boolean GetValue(Enum_CtiOs eKey, out Arg obArg)
Boolean GetValueInt(Enum_CtiOs eKey, out System.Int32 nValue)
Boolean GetValueShort(Enum_CtiOs eKey, out System.Int16 nValue)
Boolean GetValueUShort(Enum_CtiOs eKey, out System.Int32 nValue)
Boolean GetValueBool(Enum_CtiOs eKey, out System.Boolean bValue)
Boolean GetValueString(Enum_CtiOs eKey, out System.String strValue)
Boolean GetValueArray(Enum_CtiOs eKey, out Arguments arArguments)
```

Parameters

An enumerated keyword (for more information, see CTI OS Keywords and Enumerated Types) or a string specifying the keyword of the value to retrieve.

Return Values

In C++, the two-parameter version returns a boolean indicating success or failure. The one-parameter version returns the value and throws an exception upon failure.

COM returns an HRESULT. For more information, see CIL Coding Conventions.

Java methods return a null object if the method fails.

Remarks

Visual Basic's Integer type is a 16-bit integer. However, the GetValueInt method returns a 32-bit integer. Thus, in Visual Basic the return type for GetValueInt is actually a Visual Basic type Long. Visual Basic Programmers can use the GetValueInt method and receive the return value as an Integer, and Visual Basic

will perform an implicit cast. However, if the value retrieved is a 32-bit integer, an overflow error occurs. To resolve this error, use a 32-bit integer (Long).

Those methods that do not return a bool indicating success or failure throw a CtiosException if the method fails. The most common reasons for failure are NULL key or element with specified key not found.

IsValid

The IsValid method returns True if the specified key exists in the current Arguments array, otherwise it returns False.

Syntax

C++

```
bool IsValid( std::string& key );
bool IsValid( char * key );
bool IsValid( Arg& arg );
bool IsValid( enum_Keywords key );
```

COM

```
HRESULT IsValid( /*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bIsvalid);
```

VB

```
IsValid (key as string) as Boolean
```

Java, .NET

```
boolean IsValid(int key)
boolean IsValid(String key)
boolean   IsValid(Arg rArg)
```

Parameters

key/arg

Either the key of the desired Arguments member or an Arg containing a string key.

C++ and COM allow you to specify the key as a string or enumerated (for more information, see CTI OS Keywords and Enumerated Types); all others expect the key as a string.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: True if key exists in the current Arguments array, otherwise False.

NumElements

The NumElements method returns the number of elements stored in the current Arguments array. This method is useful in combination with GetElement to implement a “for” loop to iterate over all values of an Arguments array without knowing the keywords (you can retrieve those at the same time using GetElementKey).

Syntax

C++

```
int NumElements();
```

COM

```
HRESULT NumElements(/*[out, retval]*/ int * num_elements);
```

VB

```
NumElements as Integer
```

Java

```
int NumElements()
```

.NET

```
int NumElements()
```

Parameters

num_elements

Pointer to an integer value containing the number of elements in the Arguments array.

Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Number of elements in Arguments array.

Release (C++ and COM Only)

The Release method decrements the reference count for the data item. You must call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak occurs.

Syntax

C++

```
unsigned long Release()
```

COM

```
HRESULT Release()
```

VB, Java, .NET

```
Not used
```

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

C++: The current reference count after the Release() call.

RemoveItem

The RemoveItem method removes a value and its associated key from an Arguments array. Subsequent attempts to access a value that was removed using RemoveItem fail.

Syntax

C++

```
bool RemoveItem( std::string& key );
bool RemoveItem( char * key );
bool RemoveItem( enum_Keywords key );
```

COM

```
HRESULT RemoveItem(/*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bSuccess);
```

VB

```
RemoveItem ( key as Variant) as Boolean
```

Java

```
boolean RemoveItem(int key)
boolean RemoveItem(String key)
```

Parameters

key

The key to use to locate and remove the item in the Arguments array. Leading and trailing spaces are always removed from the key.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Returns true if the entry was located and removed.

SetElement (C++ COM and VB Only)

The SetElement method is identical to SetValue (which is similar to AddItem), except that it uses an index value instead of a key.

Syntax

C++

```
bool SetElement( int index, int value );
bool SetElement( int index, unsigned int value );
bool SetElement( int index, unsigned short value );
bool SetElement( int index, short value );
bool SetElement( int index, bool value );
bool SetElement( int index, std::string& value );
bool SetElement( int index, char * pchar );
bool SetElement( int index, Arg& value );
bool SetElement( int index, Arguments& value );
```

COM

```
HRESULT SetElement(/*[in]*/ int index, /*[in]*/ VARIANT * value, /*[out,retval]*/
success);
```

VB

```
SetElement (index as Integer, value as Variant) as Boolean
```

Java

```
Not available.
```

.NET

```
Not available.
```

Parameters

index

The index at which the value is to be set. This index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. Index is 1-based. Index should never be less than 1 or greater than NumElements (see above). C++ implements several overloaded methods for different value types, while COM and VB use Variants.

value

The associated value to be set in the element at the designated index.

success

Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: A boolean indicating success or failure.

SetValue

The SetValue method sets a value for a key. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++

```
bool SetValue( std::string& key, int value );
bool SetValue( std::string& key, unsigned int value );
bool SetValue( std::string& key, unsigned short value );
bool SetValue( std::string& key, short value );
bool SetValue( std::string& key, bool value );
bool SetValue( std::string& key, std::string& value );
bool SetValue( std::string& key, char * pchar );
bool SetValue( std::string& key, Arg& value );
bool SetValue( std::string& key, Arguments& value );
bool SetValue( std::string& key, const Arguments& value);
bool SetValue( char * key, int value );
bool SetValue( char * key, unsigned int value );
bool SetValue( char * key, unsigned short value );
bool SetValue( char * key, short value );
bool SetValue( char * key, bool value );
bool SetValue( char * key, std::string& value );
bool SetValue( char * key, char * value   );
bool SetValue( char * key, Arg& value );
bool SetValue( char * key, Arguments& value );
bool SetValue( char * key, const Arguments& value );
bool SetValue( enum_Keywords key, int value );
bool SetValue( enum_Keywords key, unsigned int value );
bool SetValue( enum_Keywords key, unsigned short value );
bool SetValue( enum_Keywords key, short value );
bool SetValue( enum_Keywords key, bool value );
bool SetValue( enum_Keywords key, std::string& value );
bool SetValue( enum_Keywords key, Arg& value );
bool SetValue( enum_Keywords key, const Arg& value );
bool SetValue( enum_Keywords key, Arguments& value );
bool SetValue( enum_Keywords key, const Arguments& cArguments);
bool SetValue( enum_Keywords key, char * value );
```

COM

```
HRESULT SetValue(/*[in]*/ VARIANT* key, /*[in]*/ VARIANT*
value,/*[out, retval]*/ VARIANT_BOOL* success);
```

VB

```
SetValue (key as String, value as Variant) as Boolean
```

Java

```
boolean SetValue(Arguments rArguments)
boolean SetValue(int iKey, Arg rArg)
boolean SetValue(String sKey, Arg rArg)
boolean SetValue(int iKey, int iVal)
boolean SetValue(String sKey, int iVal)
boolean SetValue(int iKey, short nValue)
boolean SetValue(String sKey, short nValue)
boolean SetValue(int iKey, String sValue)
boolean SetValue(String sKey, String sValue)
boolean SetValueUInt(int iKey, long lValue)
boolean SetValueUInt(String sKey, long lValue)
boolean SetValueUShort(int iKey, int iValue)
boolean SetValueUShort(String sKey, int iValue)
boolean SetValue(int iKey, Arg rArg)
```

.NET

```
System.Boolean SetValueArguments(Arguments rArguments)
System.Boolean SetValue(System.String sKey, System.Int32
iValue)
System.Boolean SetValueUInt(System.String sKey, System.Int64
lValue)
System.Boolean SetValue(System.String sKey, System.Int16
nValue)
System.Boolean SetValueUShort(System.String sKey, System.Int32
iValue)
System.Boolean  SetValue(System.String sKey, System.String
sValue)
System.Boolean SetValue(System.String sKey, Arg rArg)
System.Boolean SetValue(Enum_CtiOs eKey, System.Int32 iValue)
System.Boolean SetValueUInt(Enum_CtiOs eKey, System.Int64
lValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.Int16 nValue)
System.Boolean SetValueUShort(Enum_CtiOs eKey, System.Int32
iValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.Boolean bValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.String sValue)
System.Boolean SetValue(Enum_CtiOs eKey, Arg rArg)
```

Parameters

key

The key whose value is to be set.

value

The value to use in setting the element with the designated key.

success

Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: A boolean indicating success or failure.

Remarks

The C++ methods overload several implementations for different value types and allow you to specify a key via enumerated keywords (for more information, see CTI OS Keywords and Enumerated Types) as well as string. COM and VB use String keywords and Variants as values.

CILRefArg Class (C++ Java and .NET Only)

The CILRefArg class is a subclass of the Arg class. Its main responsibility is to store a reference of a CCtiOsObject object (for more information, see CtiOs Object). This class includes object references in argument structure. The object types you can use are any of the following: CAgent, CCall, CSkillGroup, CWaitObject, or CCtiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in the following table.

Method	Description
GetType	Returns the ARG_REFERENCE.
GetUniqueObjectID	Returns the UID of the contained CtiOsObject.
GetValue	Returns the encapsulated pointer in the object.
SetValue	Encapsulates the pointer to CTI OS object into the CILRefArg object.

GetType

The GetType method returns the type of the data stored by the Arg. For a CilRefArg, this is always ARG_REFERENCE.

Syntax

C++

```
enumArgTypes GetType()
```

COM

```
HRESULT GetType(/*[out, retval]*/ int* type);
```

VB

```
GetType () as Integer
```

Java

```
int GetType()
```

.NET

```
Use the GetArgType method.
```

Output Parameters

type

Integer that receives the enumerated constant that identifies the data type stored in Arg. In this case, that data type is ARG_REFERENCE.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions.

Others: Returns the enumerated value that identifies the data type stored in the Arg (for more information, see Table 2: enumArgTypes, on page 7). For CilRefArg, this is always ARG_REFERENCE.

GetUniqueObjectID (Java and .NET Only)

The GetUniqueObjectID method returns the unique objectID of the contained CtiOsObject.

Syntax

```
String      GetUniqueObjectID()
```

Parameters

None.

Return Values

If successful, it returns the unique objectID of the contained CtiOsObject. If no object is contained in the CilRefArg, it returns null.

Remarks

To obtain a unique object ID in C++, use bool GetValueString(string* pString).

GetValue

The GetValue method returns the reference to CTI OS object encapsulated in the CILRefArg.

Syntax

C++

```
C++: CCtiOsObject * GetValue()
```

Java

```
CCtiOsObject GetValue();
```

.NET

```
System.Boolean GetValue(out CtiOsObject sValue)
```

Output Parameters

.NET:sValue

Reference to the contained CtiOsObject derived class.

Return Values

C++: Returns NULL on failure.

.NET: Returns false if the method fails.

Java: Returns a null reference if the method fails.

SetValue

Sets the reference to the CTI OS Object in the CILRefArg.

Syntax

```
bool SetValue(CCtiOsObject * pObject);
```

Input Parameters

pObject

A pointer to a CtiOsObject to encapsulate (e.g. CCall, CAgent, etc.).

Return Values

If the method was able to set the reference it returns true. Otherwise, it returns false.

CCtiOsException Class (C++ Java and .NET Only)

The CCtiosException class is normally used within the Arguments class. It provides access to additional information when exceptions are thrown, such as what parameter is in error, memory allocation failed, and so on.

The following table lists the available CCtiOsException class methods.

Method	Description
CCtiosException	Class constructor.
GetCode	Returns the error code that generated the exception.
GetStatus	Returns the error status that generated the exception.
GetString	Returns a text string containing the description of the exception.
What	Returns a text string containing the description of the exception, the code of an error and the status.

CCtiosException Constructor

The CCtiosException constructor initializes an object of type CCtiosException.

Syntax

C++, Java, .NET

```
CCtiosException(const char *pMsg, int iCode, int iStatus);
```

C++

```
CCtiosException(const string& rstrMsg, int iCode, int iStatus);
```

Parameters

pMsg

Pointer to string that holds a description of an error.

iCode

Number that identifies an error.

iStatus

Status of an error.

rstrMsg

An STL string that holds a description of an error.

Return Values

None.

GetCode

The GetCode method returns the error code that generated the exception.

Syntax

C++, Java, .NET

```
int GetCode();
```

Parameters

None.

Return Values

Returns an integer error code that generated the exception. The errors are described in the Cilerror.h include file—for more information, see CTI OS Keywords and Enumerated Types.

GetStatus

The GetStatus method returns the error status that generated the exception.

Syntax

C++, Java, and .NET

```
int GetStatus ();
```

Parameters

None.

Return Values

Returns an integer error status that generated the exception.

GetString

The GetString method returns a text string containing the description of the exception.

Syntax

C++

```
const char* GetString();
```

Java, .NET

```
String GetString();
```

Parameters

None.

Return Values

Returns a text string containing the description of the exception.

What

The What method returns a text string containing the description of the exception, the code of an error, and the status.

Syntax

```
const char* What();
```

Parameters

None.

Return Values

Returns a text string containing the description of the exception, the code of an error, and the status.

CWaitObject Class

CWaitObject is a CIL object that derives from CtiOsObject. It is a utility class that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. Wait objects are created with the CreateWaitObject Session Object method and destroyed with the DestroyWaitObject Session Object method.

Warning **You must not use a WaitObject instance within an event handler. Events are sent to desktop applications by a single thread in the CIL. If that thread is blocked while waiting for a specific event, the thread deadlocks and the event handler does not receive any more events.**

Methods

The following table list the CWaitObject class methods.

Table 6: CWaitObject Class Methods

Method	Description
CreateWaitObject	For more information, see Session Object.
DestroyWaitObject	For more information, see Session Object.
DumpEventMask	Returns a printable string listing the events in the CWaitObject's mask.
GetMask	Sets a user provided pointer to an Arguments object that contains the list of events that the object waits for.
GetTriggerEvent	Gets the ID of the event that triggered the WaitOnMultipleEvents method to wake.
InMask	Returns true if the specified event ID is in the list of events that the object waits for.
SetMask	Set the list of events that the object waits for.
WaitOnMultipleEvents	Waits for the events in the object's event mask for the specified time period or until one of the events occurs.

CreateWaitObject

For more information, see Session Object.

DestroyWaitObject

For more information, see Session Object.

DumpEventMask

The DumpEventMask method returns a printable string listing the events in the CWaitObject's mask.

Syntax

C++, Java, .NET

```
string DumpEventMask();
```

Parameters

None.

Return Values

A printable string object listing the events in the wait mask.

GetMask

The GetMask method gets the list of events that the CWaitObject waits for.

Syntax

C++

```
bool GetMask(Arguments ** pMask);
```

Java, .NET

```
Arguments GetMask();
```

Parameters

pMask

A pointer to an Arguments object pointer. GetMask sets the value of pMask to a pointer to an Arguments object that contains the event mask.

Return Values

If the method was able to get the mask it returns true; otherwise, it returns false. For Java and .NET, the method returns null upon failure.

GetTriggerEvent

The GetTriggerEvent method returns the ID of the last event in the CWaitObject's mask that triggered the WaitOnMultipleEvents method to wake.

Syntax

C++

```
EnumCTIOS_EventID GetTriggerEvent()
```

Java

```
int GetTriggerEvent()
```

.NET

```
EventID GetTriggerEvent()
```

Parameters

None.

Return Values

The ID of the event or eUnknownEvent if no event triggered a wakeup.

InMask

The InMask method checks to see if the specified event is in the mask of events that the CWaitObject waits for.

Syntax

C++

```
bool InMask(int iEventId);
```

Java, .NET

```
boolean InMask(int iEventId);
```

Parameters

iEventId

The enumerated event ID of the event to check for.

Return Values

If the event is in the mask it returns true. Otherwise, it returns false.

SetMask

The SetMask method sets the list of events that the CWaitObject waits for.

Syntax

C++

```
bool SetMask(Arguments & args);
```

Java, .NET

```
boolean SetMask(Arguments rArgs);
```

Parameters

args

A reference to an Arguments object containing the list of events to wait for. The Arguments contains values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

The method returns true if it is able to set the. Otherwise it returns false.

WaitOnMultipleEvents

The WaitOnMultipleEvents method waits for the events in the CWaitObject's wait mask or returns if one of the events has not occurred after the specified timeout period. This is a “one of” method that returns after one of the specified events occurs.

Syntax

C++

```
int WaitOnMultipleEvents(DWORD dwMilliseconds = INFINITE);
```

Java, .NET

```
int WaitOnMultipleEvents(long iMilliseconds);
```

Parameters

Milliseconds

The maximum length of time in milliseconds to wait before timing out. The default is INFINITE if called without arguments. For Java and .NET, a value of zero causes this method to wait infinitely.

Return Values

The WaitOnMultipleEvents method returns one of the values listed in the following table.

Table 7: WaitOnMultipleEvents Return Values

Value	When Returned
EVENT_SIGNALED	If one of the events in the mask occurred.
EVENT_WAIT_TIMEDOUT	If the timeout period elapsed.
WAIT_FAILED	If unable to wait on the events in the mask.

Logger Class (.NET and Java Only)

The Logger class creates a Logger object and a LogManager object, if one does not already exist. Any object that needs to perform logging must instantiate the Logger class. The Logger class communicates with the singleton LogManager object, which acts as the centralized logging facility. The Logger class also defines tracing constants.

Methods

The following table list the methods in the Logger class.

Table 8: CWaitObject Class Methods

Method	Description
AddLogListener	Registers a listener with the LogManager.
GetTraceMask	Gets the current trace mask. Trace masks define trace levels, such as TRACE_MASK_CRITICAL, which enables tracing for critical errors. See the LogManager Javadoc for a description of trace masks that define tracing.
Logger (Constructor)	Creates a Logger object and also a LogManager object if one does not already exist. If one is already created, it just gets a reference to the existing singleton LogManager object.
RemoveLogListener	Unregisters a listener from the LogManager.
SetTraceMask	Sets the current trace mask.
Trace	Sends a trace message to the central LogManager with the specified trace mask.

Logger() Constructor

The Logger constructor creates a Logger object and also a LogManager object if one does not already exist. If a LogManager exists, the Logger gets a reference to the existing singleton LogManager object.

Syntax

```
void Logger()
```

Parameters

None.

Return Values

None.

GetTraceMask

The GetTraceMask method gets the current trace mask.

Syntax

```
int GetTraceMask()
```

Parameters

None.

Return Values

An int containing the current trace mask.

SetTraceMask

The SetTraceMask method sets the current trace mask.

Syntax

```
void SetTraceMask(int iTraceMask)
```

Parameters

iTraceMask

The binary or combination of trace mask bits.

Note For more information about trace masks available in the .NET CIL, see Table 1.

Return Values

None.

AddLogListener

The AddLogListener method registers a listener with the LogManager.

Syntax

Java

```
void AddLogListener(ILogListener rListener)
```

.NET

```
void AddLogListener (LogEventHandler rListener)
```

Parameters

rListener

Java: Reference to the new listener.

.NET: Reference to a LogManager LogEventHandler delegate.

Return Values

None.

RemoveLogListener

The RemoveLogListener method unregisters a listener from the LogManager.

Syntax

Java

```
void RemoveLogListener(ILogListener rListener)
```

.NET

```
void RemoveLogListener (LogEventHandler rListener)
```

Parameters

rListener

Java: Reference to the listener to be removed.

.NET:Reference to a LogManager LogEventHandler delegate to be removed.

Return Values

None.

Trace

The Trace method sends a trace message to the central LogManager with the specified trace mask. If the trace mask set on the Logger contains all the bits in the trace mask that is passed into this method, the LogManager sends the trace string to all log listeners.

Syntax

```
int Trace(int iTraceMask, String sMessage)
```

Parameters

traceMask

Trace mask for this message.

traceMessage

String containing trace message.

Return Values

int 0 if traced; -1 if not traced.

LogWrapper Class (.NET and Java Only)

The LogWrapper class instantiates the default logging mechanism. By default, the LogWrapper writes trace messages to System.Console.Out. If you instantiate the LogWrapper by passing it a filename, then the LogWrapper writes trace messages to the specified file.

Methods

The following table lists the LogWrapper class methods.

Table 9: LogWrapper Class Methods

Method	Description
Dispose	Releases system resources used by the LogWrapper (.NET only).
GetMaxDaysBeforeExpire (.NET Only)	Obtains the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.
GetMaxFileSize(.NET only)	Obtains the current log file size threshold beyond which a new file is created.
GetMaxNumberOfFiles (.NET Only)	Obtains the current number of log files threshold beyond which older files are deleted.
GetTraceMask	Gets the current trace mask.
LogWrapper()Constructor	Creates a new LogWrapper object that writes tracing messages to System.Console.Out.
LogWrapper (String fileName)Constructor	Creates a new LogWrapper object that traces to the file specified in the fileName parameter.
LogWrapper (string, int, int, int)	Creates a new LogWrapper object that traces to the file specified in the fileName parameter and sets all the provided tracing properties. If the corresponding parameter value is set to 0 then the default value is used.
SetMaxDaysBeforeExpire	Changes the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.
SetMaxFileSize	Changes the current log file size threshold beyond which a new file is created.
SetMaxNumberOfFiles	Changes the current number of log files threshold beyond which older files are deleted.
SetTraceMask	Sets the current trace mask.
UpdateTraceSettings	Parses TraceConfig.cfg and imports the settings contained within.
WriteTraceLine	Prints a string to the active trace file or to System.Console.Out if no active trace file exists.

LogWrapper() Constructor

The LogWrapper constructor creates a new LogWrapper object that writes tracing messages to System.Console.Out. This constructor also creates an instance of the LogManager, if one does not already exist. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

Syntax

```
void LogWrapper()
```

Parameters

None.

Return Values

None.

LogWrapper(string filename) Constructor

This constructor creates a new LogWrapper object that traces to the file specified in the fileName parameter. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

Syntax

```
void LogWrapper (string sFileName)
```

Parameters

sFileName

Name of the trace file.

Return Values

None.

LogWrapper(string int int int) Constructor

Creates a new LogWrapper object that traces to the file specified in the fileName parameter and sets all the provided tracing properties. If the corresponding parameter value is set to 0 then the default value is used. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

Syntax

Java

```
void LogWrapper (string sFileName, long iMaxSize, int iArchives, int iFlushIntervalMs)
```

.NET

```
void LogWrapper (string sFileName, int maxSizeKB, int maxFiles, int daysUntilExpiration)
```

Parameters

.NET and Java:sFfileName

Name of the trace file.

.NET:maxSizeKB

Maximum size of a single trace file in KB (default is 2048 KB).

Java:iMaxSize

Maximum size of a single trace file.

.NET:maxFiles

Maximum number of trace files to create before older files are deleted (default is 4).

Java:iArchives

Maximum number of trace files stored.

.NET:daysUntilExpiration

Maximum age (in days) of the active trace file before it is rolled over to a new file regardless of size (default is 1 day).

Java:iExpires

Number of days before the trace file expires.

Java:iFlushIntervalMs

Number of milliseconds before data is flushed to the trace file. There is no .NET counterpart for this parameter.

Return Values

None.

Dispose (.NET Only)

The Dispose method releases system resources used by the LogWrapper.

Syntax

```
void Dispose ()
```

Parameters

None.

Return Values

None.

GetMaxDaysBeforeExpire (.NET Only)

The GetMaxDaysBeforeExpire method gets the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.

Syntax

```
int GetMaxDaysBeforeExpire ()
```

Parameters

None.

Return Values

Current age threshold.

SetMaxNumberFiles

The SetMaxNumberFiles method changes the current number of log files threshold beyond which older files are deleted. If the provided value is not greater than zero, the default value of 4 is used.

Syntax

Java

```
void SetMaxNumberFiles (int iArchives)
```

.NET

```
void SetMaxNumberFiles (int maxFiles)
```

Parameters

maxTraceFiles

New number of files threshold. If 0 is specified, the default value is used.

Return Values

None.

GetMaxNumberFiles (.NET Only)

The GetMaxNumberFiles method gets the current number of log files threshold beyond which older files are deleted.

Syntax

```
int GetMaxNumberFiles ()
```

Parameters

None.

Return Values

Current number of files threshold.

SetMaxDaysBeforeExpire

The SetMaxDaysBeforeExpire method changes the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.

Syntax

Java

```
void SetMaxDaysBeforeExpire (int iExpires)
```

.NET

```
void SetMaxDaysBeforeExpire (int maxDaysUntilExpiration)
```

Parameters

maxDaysUntilExpiration

New age threshold. If value is not greater than zero, the default value of 1 is used.

Return Values

None.

ProcessConfigFile

The ProcessConfigFile method opens the default config file (TraceConfig.cfg) in the parent directory and updates LogWrapper trace settings with data from the config file.

Syntax

```
boolean ProcessConfigFile()
```

Parameters

None.

Return Values

Returns true if operation succeeded and false if unable to open theTraceConfig.cfg file.