Axis C++ Memory Management Guide

<!-- --> <!-- -->

1. Axis C++ Memory Management Guide

1.0 Version

Feedback: axis-c-dev@ws.apache.org

1.1. Introduction

Memory management is very important and if not handled correctly, will quickly consume resources and slow down your application. The basic rules are;-

For client applications,

- Any memory object that is created by the client to pass to a web service must be deleted by the client.
- Any memory object that is passed back from the web service must be deleted by the client.

For server applications,

- Any memory object that is passed to the service from the engine must be deleted by the service.
- Any memory object that is created by the service and handed back to the engine will be deleted by the engine (or generated wrappers).

Within the client or service applications, all memory object must be created using 'new' and deleted using 'delete' (The C style memory functions 'malloc' and 'free', or any of their variants must not be used).

1.2. new/delete Semantics

If you are using the wrappers produced by WSDL2Ws then a lot of the memory management is handled for you within the generated code. You still have to follow the rules for client or service, but there may be some additional steps that you will have to follow.

1.2.1. Input Parameters

The following examples rely on the application is using the stubs generated by the WSDL2Ws tool. If the user needs to use the Axis API directly, it is assumed that they know what methods to call and how these methods have been bundled by the generated code.

1.2.1.1. Simple Types

If the object is not nillable, then use the basic type. For example, if the web service requires an `xsd__byte` value, then the client/service code would be as follows;-

```
webService->asNonNillableElement( (xsd__byte) 127)
```

If the object is nillable, then use a pointer to the basic type. For example, if the web service requires a pointer to a `xsd__byte` value, then the client/service code would be as follows;-

```
xsd__byte * pNillableInput = new xsd__byte();
*(pNillableInput) = (xsd__byte) 123;
webService->asNillableElement( pNillableInput);
delete pNillableInput;
```

Notice that once the client/service code no longer requires the pNillableObject object, it is deleted (and must be deleted by the client/service code).

1.2.1.2. Arrays and Complex Types

Arrays and Complex Types are treated as nillable, even if they are not. For example, if the web service requires an array of `xsd__byte` values, then the client/service code would be as follows;-

```
// Need an xsd__byte array of 2 elements,
// each element is assigned the value 123.
int arraySize = 2;
xsd__byte ** array = new xsd__byte*[arraySize];
for ( int inputIndex = 0 ; inputIndex < arraySize ; inputIndex++ )
array[inputIndex] = new xsd__byte( 123);
// Now copy this array into the xsd__byte_Array
// that will be used to pass to the web service.
xsd__byte_Array arrayInput;
arrayInput.set( array, arraySize);
// Call the web service.
webService->asArray( &arrayInput);
// Clear up input array
for ( int deleteIndex = 0 ; deleteIndex < arraySize ; deleteIndex++ )
{
delete
array[deleteIndex];
}
```

```
delete [] array;
```

Which is exactly the same code as would be used if the array was not nillable.

1.2.2. Output Parameters

The following examples rely on the application is using the stubs generated by the WSDL2Ws tool. If the user needs to use the Axis API directly, it is assumed that they know what methods to call and how these methods have been bundled by the generated code.

1.2.2.1. Simple Types

If the returned object is not nillable, then use the basic type. For example, if the web service returns a `xsd__byte` value, then the client/service code would be as follows;-

```
xsd__byte result = webService->asNonNillableElement( (xsd__byte) 127);
```

If the object is nillable, then use a pointer to the basic type. For example, if the web service returns a pointer to a `xsd__byte` value, then the client/service code would be as follows;-

```
xsd__byte * pNillableOutput = webService->asNonNillableElement( (xsd__byte) 127);
delete pNillableOutput;
```

Notice that once the client/service code no longer requires the `pNillableOutput` object, it is deleted (and must be deleted by the client/service code).

1.2.2.2. Arrays and Complex Types

Arrays and Complex Types are treated as nillable, even if they are not. For example, if the web service returns an array of `xsd__byte` values, then the client/service code would be as follows;-

```
// Call the web service.
xsd__byte_Array * arrayOutput = webService->getArray();
// Retrieve the information within the array.
int byteArraySize = 0;
const xsd__byte ** byteArray = arrayOutput->get( byteArraySize);
// Clear up output array delete arrayOutput;
```

Which is exactly the same code as would be used if the array was not nillable. Notice that only the `arrayOutput` object (that is returned by the web service) needs to be deleted. The `byteArray` object is a pointer to the contents of the `arrayOutput` object so must not be deleted.

1.3. Dealing with SOAP Headers

1.3.1. From Stubs

IHeaderBlock is a virtual class that defines the interface to deal with SOAP headers. To create an IHeaderBlock in the client application, use the API provided with Stub classes, i.e. ;-

```
IHeaderBlock * Stub::createSOAPHeaderBlock( AxisChar * pachLocalName, AxisChar * pachUri);
```

The Stub class methods that handle header blocks keeps a list of all the created header blocks. When the destructor is called, it will clean up memory by deleting the header blocks that were created using the cerateSOAPHeaderBlock method.

Note 1: The client/service application must use the appropriate Stub method to delete a header block, i.e. ;-

```
void deleteCurrentSOAPHeaderBlock();
```

Note 2: The IHeaderBlock destructor will take care of the header block member variables (for example, BasicNodes may have children and attributes. These will be deleted when the parent is deleted.).

1.3.2. From Handlers

If header blocks are created within a 'Handler' then it is the responsibility of the 'Handler' writer to delete them. The deletion would occur in the 'clean-up' code either in the fini() method or in the destructor of the Handler, depending on the following rules;-

- If it is a Session Handler which needs to maintain its state, then the cleanup has to be done in the destructor.
- If it is a request type handler the clean up can be done in the fini() method of the Handler.

If a target handler access a header block created by the de-serializer then it is the responsibility of the Handler to delete it.

1.4. Windows Issues

For Windows platforms, everything must built with the compiler flag '/MD' regardless whether it is a DLL or an EXE. There are still problems however when passing objects over process boundaries. If an object is created in one process (say the Axis engine DLL) and then passed to another (say the client application) then when the client tries to delete that object, it cannot find it on its own process heap and throws an exception. This is because the

client process does not own the memory object. To overcome this problem, on the process boundary, the original object is cloned (the clone uses the client heap) and then the original object is freed from the engine heap. Here is an example taken from the wrapper code created by WSDL2Ws from the Arrays unit test (Arrays.cpp);-

```
xsd__int_Array * Arrays::simpleArray( xsd__int_Array* Value0)
{
xsd__int_Array * RetArray = new xsd__int_Array();
:
:
if ( AXIS_SUCCESS == m_pCall->invoke())
{
if ( AXIS_SUCCESS == m_pCall->checkMessage( "simpleArrayResponse",
"http://org.apache.axis/Arrays/"))
{
Axis_Array * RetAxisArray = m_pCall->getBasicArray( XSD_INT, "simpleType", 0);
RetArray->clone( *RetAxisArray);
Axis::AxisDelete( (void*) RetAxisArray, XSD_ARRAY);
}
}
m_pCall->unInitialize();
return RetArray;
}
```

The two lines of interest are the cloning of the memory object (RetAxisArray exists within the engine heap) into the RetArray memory object (that exists within the client heap) and then the deletion of the RetAxisArray by calling the AxisDelete method (which exists within the engine process and hence will be able to delete the object from that heap).