# Object Tcl

## Contents



## Introduction

The Object Tcl extension is implemented in C++ in such a way as to promote the use and reuse of C++ classes from within Object Tcl scripts.

Object Tcl makes it possible to:

- Create, manipulate and destruct C++ classes from within the Object Tcl domain.
- Create, manipulate and destruct Object Tcl classes, that inherit from C++ classes, from within the C++ domain.
- Inherit Object Tcl classes from built-in C++ classes. Note that the dynamic binding of methods crosses the domain from C++ to Object Tcl and vice versa.
- Pass Object Tcl objects, of a class that inherits from an existing C++ class, into the C++ domain for manipulation and possible destruction.
- Re-implement Object Tcl classes in C++ for performance improvements and access to the more esoteric OS or third party library facilities with minimal effort.

Object Tcl provides an interpreted extension to the C++ language while still maintaining the support for object orientation.

This page describes how C++ classes may be exported to the Object Tcl system for use from Tcl.

## Restrictions

There are a few restrictions on the usage of C++ and Object Tcl. These restrictions are described below:

1. C++ makes it possible to have many methods with the same name that can be distinguished by their formal arguments. In C++ this is called overloading. Tcl is weakly typed and it is therefore impossible to disambiguate the methods based on the types of the formal arguments. For this reason method overloading is not supported.

   If a C++ class does have more than one method with a given name and it is to be exported to Object Tcl then the CDL for this class must describe only one of them.

   The most common occurrence of overloading is in class constructors. Only one constructor for a C++ class may be exported to Object Tcl.

2. C++ provides support for defining the behavior of operations upon a class. This is called operator overloading. Object Tcl does not support this. Operators on a C++ class cannot be exported into Object Tcl.

3. As Tcl is weakly typed, with all types represented as strings, then there are restrictions on the parameters and return types that may be passed back and forth between the two domains. The CDL processor is designed in a way to make it easy to build in additional type conversion support.

---

## CDL Processing

The Object Tcl system includes a CDL processor called "cdl". The CDL processor takes files in the CDL format and generates C++ files that will bind the application's C++ classes into Object Tcl.

The CDL processor takes three arguments: the first is a flag indicating whether the CDL processor is to generate a header file (-h) or source file (-s), the second is the name of the input file, including suffix, and the third is the output file, including suffix.

For each CDL file, the CDL processor must be invoked twice, once to generate the header file and once to generate the source file.

Rules can be added to makefiles to automatically take CDL files, with a suitable suffix, and generate C++ files that are then compiled into object files. The source distribution contains examples of makefiles that have such rules.

The C++ code generated by the CDL processor uses C++ static constructors to facilitate the inclusion of an application's C++ classes into the Object Tcl environment with absolutely no modification of the application code. Only a relink is necessary.

---

# Class Definition Language

CDL files contain descriptions of the C++ classes that are to be exported to Object Tcl.

## Syntax

The CDL processor is based around a Tcl interpreter itself, hence the familiarity of the CDL syntax.

In a CDL file there are two commands:

`pass -(h|s)` *`arg`*

and

`class ?-isA` *`classList`*`?` *`name desc`*

The `pass` command takes the rest of the arguments and passes them straight through to the generated C++ file. The `pass` command is generally used to place `#include` directives in the generated C++ but it is also useful for placing comments or version identifiers in the C++ files. The `-h` or `-s` flag may be used to specify that the *`arg`* is only passed to the header file or source file respectively. If the destination flag is omitted then the *`arg`* will be passed to both the header and source files.

The `class` command is used to describe a C++ class to the Object Tcl system. The *name* argument is the name of the class. The `-isA` *`classList`* optional parameter can be used to specify the list of superclasses, both direct and indirect, of this class. The list of superclasses is used for coercing the types of objects when they are passed between Tcl and C++ using the `obptr` and `obref` CDL types. The *desc* argument is a Tcl script that may use the following commands internally:

`constructor` *`args`*
: This command describes the constructor for the C++ class that will be exported to Object Tcl. The *args* argument is a Tcl script that may use the argument type commands.

`method` *`name`* `(-dynamic | -static)` *`args rtn`*
: This command describes an instance method that will be exported to Object Tcl. The *name* argument is the name of the method; the `-dynamic` or `-static` switch indicates whether the method is a C++ virtual method or not. The *args* argument is a Tcl script that may use the argument type commands, and the *rtn* argument is a script that may use one of the return type commands.

: A C++ virtual method can be exported as `-static` in which case the dynamic binding will stop at the Object Tcl/C++ interface. This may be of use in some cases as the use of the `-dynamic` option causes a performance overhead even if the method is not redefined in an Object Tcl subclass.

`classMethod` *`name args rtn`*

This command describes a class method (a C++ static member function) that will be exported to Object Tcl. The *name* argument is the name of the method. The *args* argument is a Tcl script that may use the argument type commands, and the *rtn* argument is a script that may use one of the return type commands.

## Argument Types

The *arg* parameter of the `constructor`, `method` and `classMethod` commands is a Tcl script that can make use of the following commands:

- `int`
- `float`
- `double`
- `str`
- `obptr` *`className`*
- `obref` *`className`*

The *className* argument to the `obptr` and `obref` commands specifies the actual class expected by the C++ method.

The CDL processor is designed to make it easy to support new type conversions between the C++ and the Object Tcl domains.

## Return Types

The *rtn* parameter of the `method` and `classMethod` commands is a Tcl script that can make use of the following commands:

- `int`
- `float`
- `double`
- `str`
- `obptr` *`className`*
- `obref ?-new?` *`className`*

The *className* argument to the `obptr` and `obref` commands specifies the actual class returned by the C++ method.

The `-new` flag for the `obref` return type may be used to indicate that a copy of the returned object should be made on the heap.

As with argument types, the CDL processor is designed to make it easy to add new type conversions between C++ and the Object Tcl domains.

# C++ Objects

Prior to beta 1.1 of Object Tcl, it was not possible to create a C++ object from within the C++ domain and then pass it into Object Tcl for manipulation.

Beta 1.1 of Object Tcl removes this restriction by allowing the C++ domain to create instances of *`class`*`_otcl` with the constructor described in the CDL description. For example:

```
return new Shape(300,300);
```

This new Shape object could not be passed back to Object Tcl but now you can write:

```
return new Shape_otcl(300,300);
```

and pass the new Shape object into Object Tcl.

This requires the C++ domain to include header files generated by the CDL processor.

---

# Exceptions

Dynamic binding of methods makes it possible for a C++ method invocation to result in the execution of a method described in Tcl. It is quite possible for the Tcl to be incorrect and generate an exception.

It is possible to catch exceptions in three ways.

### General exception handler

By default, any exception in a Tcl method body invoked from the C++ domain will result in the error being reported to `stdout` and the application terminating with an error status.

### Object exception handler in Tcl

It is possible to implement an instance method called **otclErrorMethod** on an Object Tcl class. This method will be called on any Tcl exception that occurs when a method is invoked from C++.

The **otclErrorMethod** takes two parameters. The first parameter is the name of the method that caused the exception; the second is the error message generated by the exception.

### Object exception handler in C++

It is possible to implement an instance method called **otclErrorMethod** on a C++ class in the same manner as for Object Tcl classes described above. This method must be exported to the Object Tcl system via the CDL description and it must take two arguments, both of which are of type `char *`.

---



---

*otcl@x.co.uk*