Using Factory Functions

This topic explains how you can define and use factory functions that return specific values of a given type. This allows you to establish repositories of valid objects that can be used in automatically-generated tests and in test cases created with the Test Case Wizard.

Sections include:

About Factory Functions

A factory function is a function (global, namespace level or a static class method) that returns a given type and has a CppTest_Factory_ name prefix. There can be more than one factory function defined for a given type.

When C++test sees a declaration of such a factory function in the compilation unit for which it generates test cases, it considers this factory function to be one of the possible initialization options for a given type. If the factory function has a non-empty parameters list, C++test will initialize factory function arguments just as it would for arguments of a constructor used to create an object of a given type.

Defining Factory Functions

To create a factory function for the type Type, write the function with the following signature:

```
Type CppTest_Factory_<factory_function_name>(<factory_arguments>);
```

For example:

```
int CppTest_Factory_generateBooleanInt(bool b)
{
     return b ? 1 : 0;
}
MyClass* CppTest_Factory_createMyClass(int size, int value)
{
     MyClass* m = new MyClass;
     m->initialize(size);
     m->setValue(value);
     return m;
}
```

Since C++test needs to see the declarations of the factory functions in the context of the compilation unit for which it generates test cases, we recommend the following approach to writing factory functions:

1. Create a header file with the factory function declarations to be used in test cases. For instance, such a header can be created in the 'factory' folder under the tested project root.
2. In all tested source files for which factory functions should be used, add an include directive for the created factory functions header. We recommend having it guarded with #ifdef PARASOFT_CPPTEST.
    - You can quickly add such an include directive to the tested source file by going to the source code editor, typing `ffi`, then pressing **CTRL+SPACE**.
3. Create a source file with the factory functions implementation. Such a source file should be created in one of the folders specified in the Test Configuration's **Execution> Symbols> Use extra symbols from files found in** field, which is set to `${cpptest:cfg_dir}/safes-tubs;${project_loc}/stubs;${project_loc}/factory` by default.
    - You can quickly add a factory function section to the factory functions source file by going to the source code editor, typing `ffs` in the source code, then pressing CTRL+SPACE.
    - You can quickly add an additional factory function template to the code by going to the source code editor, typing `ff` in the source code, then pressing CTRL+SPACE.

- When using factory functions in standalone test suites—both for test cases that were written manually and those added using the Test Case Wizard—make sure you add include directives for factory function headers that contain declarations of the factory functions used.
- Source files created to contain factory function definitions need to be excluded from the regular build. See Excluding Test Cases from the Build for instructions.

Using Factory Functions

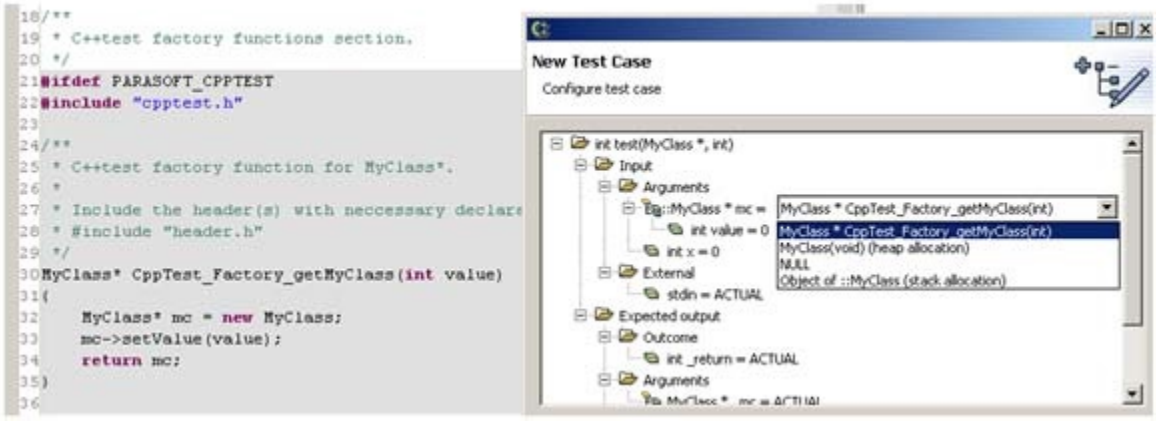
With Automatically-Generated Test Cases

To configure C++test to use factory functions in automatically-generated test cases:

1. In the Test Configuration manager's **Generation> Test Case** tab, enable the **Use factory functions** option.
2. If you want to use factory functions exclusively, also enable **Do not use other initializers for types with factory functions**.

With the Test Case Wizard

Factory functions found in the tested compilation unit can also be used when creating test case with the Test Case Wizard. In this case, factory functions will appear as additional initialization methods for a given type.



Using Data Sources with Factory Functions

Any data source that you configure for use in C++test (as described in Using Data From Data Sources to Parameterize Test Cases) can be used in factory functions. You can use the C++test Data Source API to access values from a data source.

Here is an example:

```
MyClass CppTest_Factory_getMyClassObjectFromDS (void)
{
  MyClass obj;
  if (CPPTEST_DS_HAS_COLUMN("MyClass.int")) {
    obj.initialize(CPPTEST_DS_GET_INTEGER("MyClass.int"));
  } else {
    // Data Source not available in this test case
    obj.initialize(0);
}
```

Notes

- Ensure that the given data source column exists in the context of the currently-executed test case.
- If you want to use the C++test API in a factory function, ensure that the "cpptest.h" header is included in the factory function definition file. We recommend including this header before any other header files.