



DLLTESTER version 1.4



Wageningen Software Labs



Rijkswaterstaat RIZA

This page is intentionally left blank.



DLLTESTER version 1.4

A.J. Otjens

By commission of:

Ministry of Transport, Public Works and Water Management
Directorate-General of Public Works and Water Management

Institute for Inland Water Management and Waste Water Treatment/RIZA

Contacts: Michiel Blind
Flip Dirksen

RIZA working document: 2002.164X
W!SL report: 2002-030-01

Bibliography:

Otjens, A.J., 2002, GF DLLTESTER 1.4, W!SL, Wageningen, report 2002-030-01, by commission of Institute for Inland Water Management and Waste Water Treatment/RIZA, Lelystad, working document 2002.164X, The Netherlands, 24 p.

September 2002



Wageningen Software Labs



Rijkswaterstaat RIZA

This page is intentionally left blank.

Contents

SUMMARY	1
1 INTRODUCTION	3
1.1 Generic Framework	3
1.2 DLLTESTER	3
2 TECHNICAL DOCUMENTATION	4
2.1 General info	4
2.2 Architecture	5
2.3 Saving testproject settings	9
2.4 Delphi implementation	10
3 USER REFERENCE	11
3.1 General info	11
3.2 Installing the DLLTESTER program	11
3.3 Set up a DLL test	12
3.4 Execute the DLL test	12
3.5 Test function settings	14
3.6 Saving test settings	15
4 TEST REPORT	16
4.1 Introduction	16
4.2 Test results	16
5 RIZA TEST REPORT	23
5.1 Documentation	23
5.2 DLLTESTER software functioning	23
5.3 DLLTESTER software source	23
5.4 Conclusions	24

This page is intentionally left blank.

Summary

This report is the result of the project “DLLTESTER” in which a tool was built that can be used within the Generic Framework (GF). The Generic Framework is a Dutch initiative which aims at building an IT framework which allows quick and easy linking of computational models to form integrated modelling systems.

The tester for Dynamic Link Libraries (DLL) “DLLTESTER” that was built in this project supports working with the framework. The tool is beneficial in:

- 1) The development of a new computation module DLL (model DLL);
- 2) The adaptation of a legacy model to a computation module DLL which can easily migrate to the Generic Framework;
- 3) The implementation of the “wrapper”, which is the interface tool between a model DLL and the GF standard interface.

The project has produced the DLLTESTER.DLL a GF compliant component, which has been tested and accepted by RIZA. This report presents the results of the project, such as technical documentation and user references.

In general the project has been delivered on time and on budget. Due to some required additional requirements the final results were produced some time later than originally planned.

This page is intentionally left blank.

1 Introduction

1.1 Generic Framework

The Generic Framework (GF www.genericframework.org) is a Dutch initiative aiming at developing an IT framework which allows hands-on, easy and quick linkage of models to form integrated modelling systems. The framework allows time-step based linkage and feedback between models.

The current framework consists of basic components, some migrated (legacy) models and generic tools. “Migrated models” means that the models have been adapted to work within the framework and easily can be connected to other models and tools.

This report is the result of the development of a DLLTESTER tool.

The GF initiative is supported by the major players in integrated water en environmental modelling in the Netherlands.

1.2 DLLTESTER

The tester for Dynamic Link Libraries (DLL) “DLLTESTER” that was built in this project supports working with the framework. The tool is beneficial in:

- 1) The development of a new computation module DLL (model DLL);
- 2) The adaptation of a legacy model to a computation module DLL which can easily migrate to the Generic Framework;
- 3) The implementation of the “wrapper”, which is the interface tool between a model DLL and the GF standard interface.

The DLLTESTER component is a simple and easy to use program to test model wrappers and calculation modules, implemented as dynamic link libraries, for the Generic Framework (GF). This component allows you to test (parts of) model wrappers and calculation modules. Because testing and debugging in the complete GF environment can be very complex, this test facility can help you to trace errors or defects.

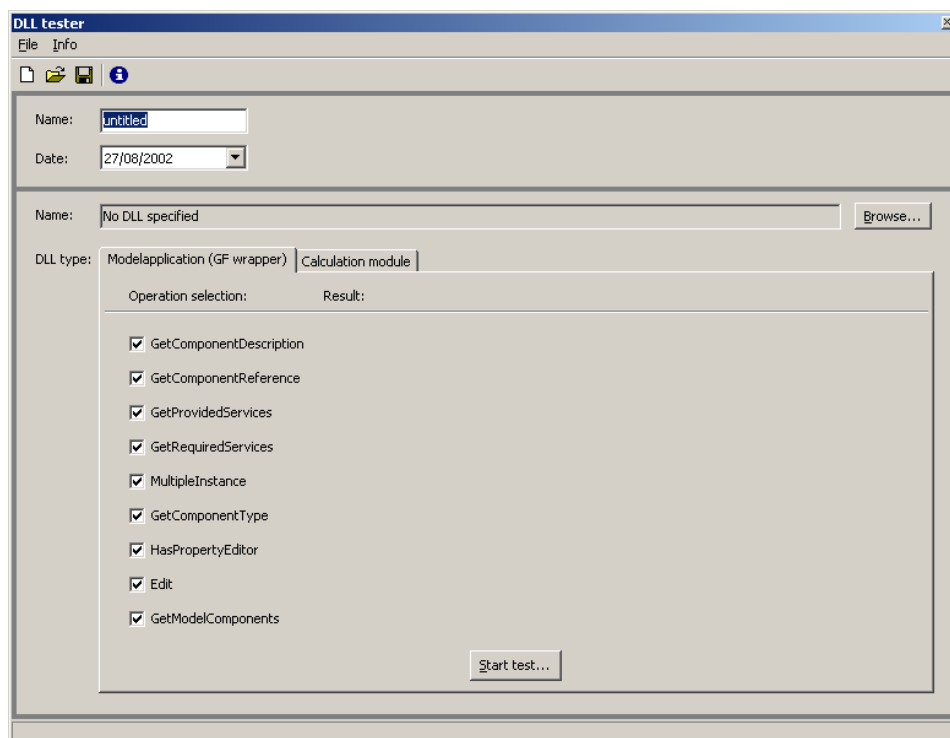
2 Technical Documentation

2.1 General info

The DLLTESTER component is a tool to check Dynamic Link Libraries (DLL's). The following tests are included:

- Test the availability of a specified DLL;
- Test of loading the DLL into memory;
- Test the availability of functions which are part a GF ModelApplication (model wrapper);
- Test the execution of functions which are part of a calculation module (DLL).

The settings which are required in these tests can be specified in a graphical user interface which is shown below.



After specifying a DLL, the provided functions can be selected or deselected and function names can be modified (when a calculation module is tested). When the test is started, all selected functions are tested successively. The following results are possible:

1. The tested function could not be found, execution of the function is omitted;
2. The tested function is found, but execution of the function results in an error;
3. The tested function is found and successfully executed;
4. The DLL could not be found and further testing is omitted;
5. The DLL could not create the ModelApplication object (when a GF ModelApplication wrapper is tested) and further testing is omitted;

All settings can be stored for later use. A simple inifile format is used to store general information (test name and date) and specification of which functions are selected in the test and which function names are used.

The following functions can be tested:

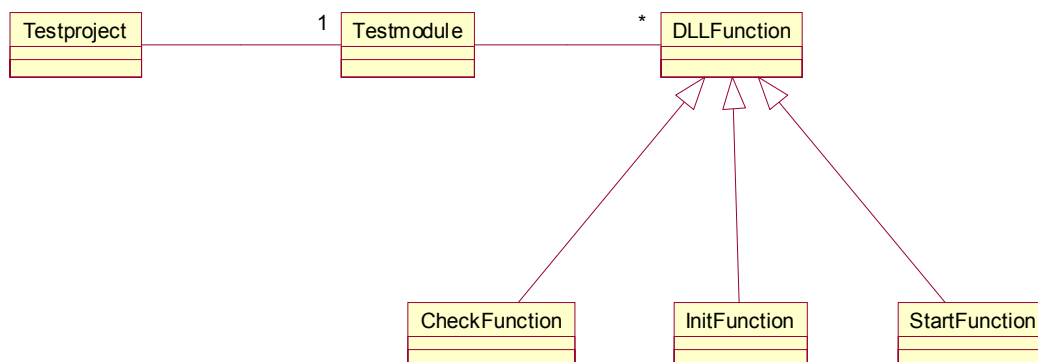
```

function Check: Boolean;
function Init1: TOperationResult;
function Start1, 2(aStartTime, aEndTime : TSrwTime; aTraceID : LongInt): TOperationResult;
function Finalize1: TOperationResult;
function GetStarttime: TSrwTime;
function GetEndtime: TSrwTime;
function GetCurrentTime: TSrwTime;
function GetNexttime: TSrwTime;
function GetComponentDescription: TComponentDescriptor;
function GetComponentReference: TComponentReference;
function GetProvidedServices: TIterator;
function GetRequiredServices: TIterator;
function MultipleInstance: Boolean;
function GetComponentType: TComponentType;
function HasPropertyEditor: Boolean;
function Edit: TOperationResult;
function GetModelComponents: TIterator;

```

2.2 Architecture

In the DLLTESTER architecture several classes are defined which implement part of the desired functionality. In the following class diagram, the classes and dependencies are presented.



The Testproject class manages all DLL test specifications and implements methods to load and save test specifications in ini-files. The Testproject interface consists of the following methods (method arguments are omitted, for details the sourcecode can be consulted):

```

interface Testproject {
    Get/Set DLLName;
        Get- and Set-method for the DLL to be tested.
    Get/Set ProjectName;
        Get- and Set-method for the name of the test (this is used as filename).
    Get/Set ProjectLocation;
        Get- and Set-method for the test location (this is used to save the test settings).
    Get/Set Date
        Get- and Set-method for the test date (this is meta info which can be stored).
    LoadProject;
        Read DLL test specifications from file.
    SaveProject;
        Save DLL test specifications to file.
}

```

¹ This function can also be tested using a Boolean result type.

² This function can also be tested without Starttime, Endtime and TraceID arguments.

```

ResetProject;
    Restore all default test specifications.
IsFunctionChecked;
    Check whether a function is marked to be tested.
GetFunctionName;
    Retrieve the name to use in a function test.
SetFunctionCheck;
    Mark a function to use in the DLL test.
SetFunctionName;
    Set the name of a function to be tested.
TestInterface;
    Start testing the marked functions.
GetResultText;
    Retrieve test results as strings.
GetFunctionArgument;
    Read the arguments for a function (used for the 'Start' function).
SetFunctionArgument;
    Set the arguments for a function (used for the 'Start' function).
GetFunctionResultType;
    Read the result type for a function (used for the 'Start', 'Init' and 'Finalize' function).
SetFunctionResultType;
    Set the resulttype for a function (used for the 'Start', 'Init' and 'Finalize' function).
};

```

The information regarding all functions to be tested are managed by a Testmodule class. This class is owned by the Testproject class: a Testproject object contains one instance of a Testmodule. The Testmodule can load and free a DLL and can test all DLL functions successivily.

```

interface TestModule {
    Reset;
        Restore all default settings .
    ResetTest;
        Clear all test results.
    GetNumberOfFunctions;
        Retrieve the number of functions which are part of the Testmodule.
    IsFunctionChecked;
        Check whether a function is marked to be tested.
    GetFunctionName;
        Retrieve the name of a function to be tested.
    GetDefaultFunctionName;
        Retrieve the default name of a function to be tested.
    SetFunctionCheck;
        Mark a function to use in a DLL test.
    SetFunctionName;
        Set a function name to use in a DLL test.
    TestFrameworkComponentInterface;
        Test all marked functions which are part of the FrameworkComponent interface.
    TestBuildingFCInterface;
        Test all marked functions which are part of the BuildingFC interface.
    GetResultText;
        Retrieve the test result of a function.
};

```

All functions which are part of the Testmodule are subclasses of DLLFunction. A DLLFunction object contains all information of a DLL function and can call a DLL function by retrieving a function address from the DLL.

```

interface DLLFunction {

```

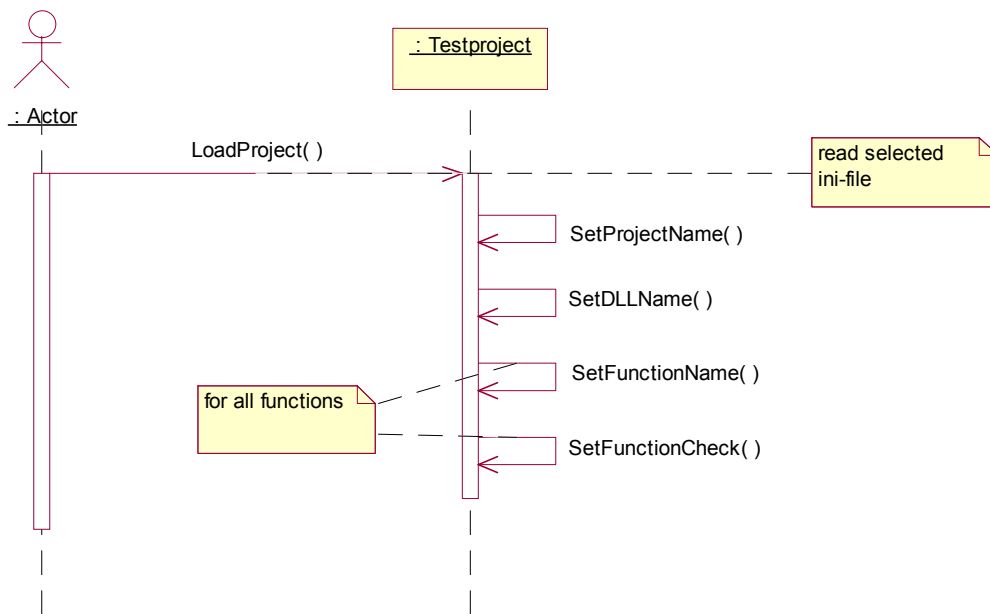
```

Select;
    Select this function in the test.
Deselect;
    Deselect this function in the test.
IsSelected;
    Check whether this function is selected for the test.
SetSucceededTest;
    Set Succeeded property when this function is tested successfully.
SetFailedTest;
    Set Failed property when this function is not tested successfully.
SetFunctionFound;
    Set FunctionFound property when this function address is found.
SetFunctionNotFound;
    Set FunctionFound property when this function address is not found.
TestOK;
    Check whether this function is tested successfully.
Get/Set FunctionName;
    Get- and Set method for the function name.
RestoreDefaults;
    Restore default function name and test result.
Get/Set DefaultName;
    Get- and Set the default function name.
Test;
    Start to find the function address and call the DLL function.
ResetTest;
    Clear all test results of this function.
GetResult;
    Retrieve the test result for this function as string.
};

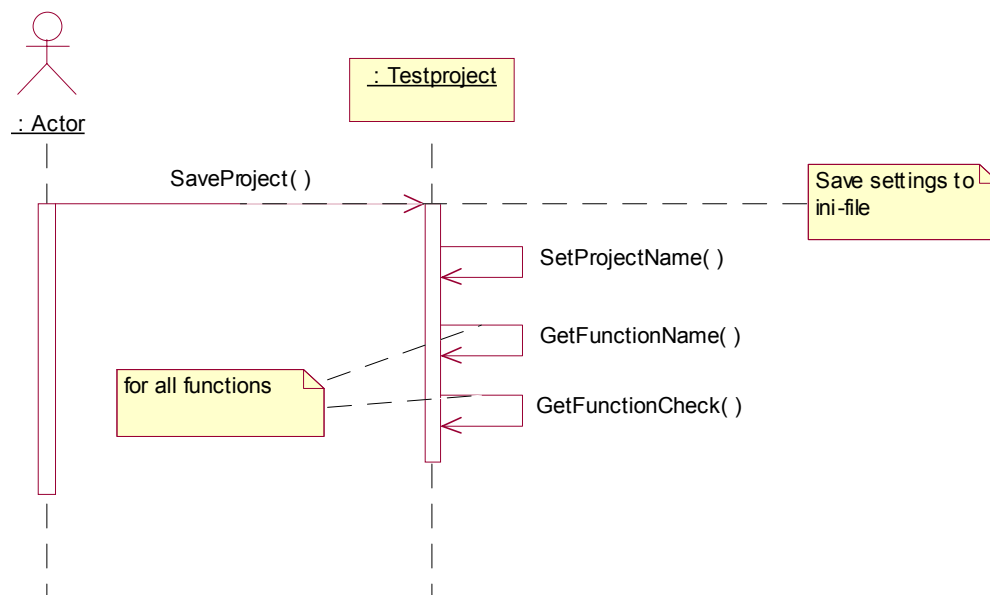
```

The co-operation between the classes which are explained above is presented in the following sequence diagrams, which show how loading, saving and execution of a test is performed.

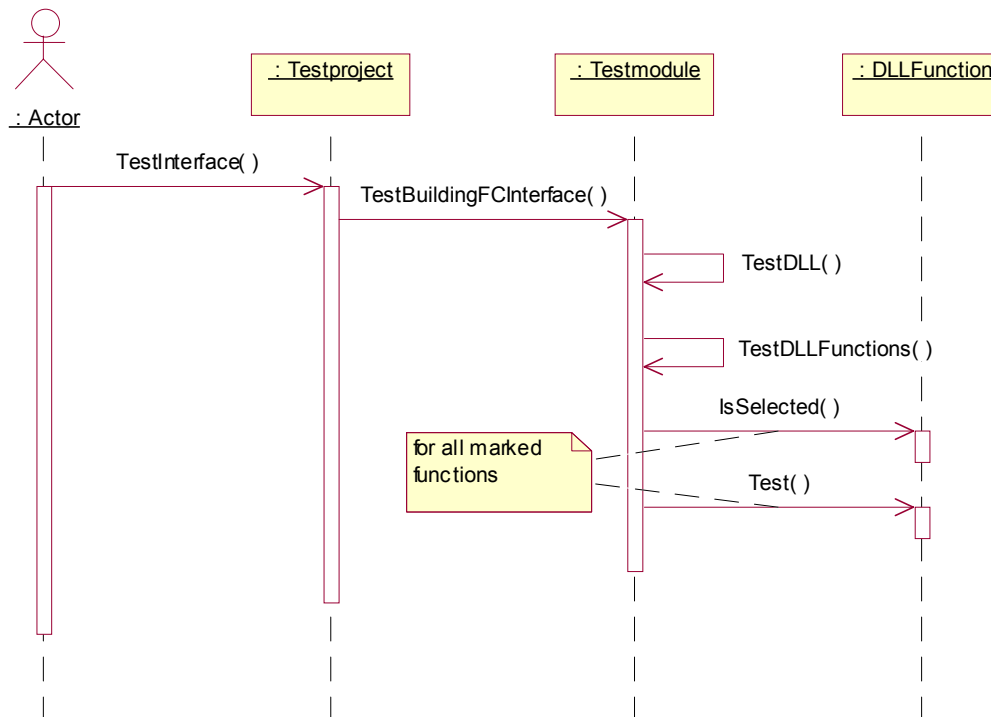
Load test project:



Save test project:



Test DLL:



2.3 Saving testproject settings

For saving test project settings, an ASCII file in simple inifile format is used. This file contains five sections with information regarding the DLL test:

- General section:
contains the project name and date.
- Frameworkcomponent section:
contains the list of functions which are part of the GF Frameworkcomponent interface.
- BuildingFC section:
contains the list of functions which are part of the GF BuildingFC interface.
- Arguments section:
contains the start-function argument settings (start- and endtime).
- Resulttype section
contains the start-, init- and finalize resulttype settings

Example:

```
[GENERAL]
Name=Test1
Date=26/08/2002
DLL=C:\Data\TestDLLSeveralFunctionWrong.dll

[FRAMEWORKFUNCTION]
GetComponentDescriptionName=GetComponentDescription
GetComponentDescriptionSelected=1
...
```

```

[BUILDINGFCFUNCTION]
CheckName=Check
CheckSelected=1
InitName=Init
InitSelected=1
...
[ARGUMENTS]
StarttimeArgument=27/03/2002
EndtimeArgument=31/03/2002

[RESULTTYPE]
Init=Boolean
Finalize=Boolean
Start=Boolean

```

2.4 Delphi implementation

The DLLTESTER component is programmed in Borland Delphi 5, based on the default GF GenericTool implementation. The program uses the GF sourcecode library SrwLib. The compiled units of this library must be available to compile the DLLTESTER program. The Delphi project file 'DLLTESTER.dpr' contains all project information. This project consists of several sourcecode units which contain user-interface or class implementations. The following units are used:

- fTestform: contains the main form of the DLL tester program;
- fAbout: contains the about form of the DLL tester program;
- fArguments: contains the form to specify arguments of the Start-function;
- fResulttype: contains the form to specify the result type of functions;
- uDLLTESTER: contains the DLLTESTER generic tool class implementation;
- uTestproject: contains the Testproject class implementation;
- uTestModule: contains the Testmodule class implementation;
- uDLLFunction: contains the DLLFunction class implementation;
- uConcreteDLLFunctions: contains the implementation of all DLLFunction subclasses;
- uDLLTESTERConstants: contains all used constants.

For testing purposes, four dynamic link libraries are developed to test the DLLTESTER program. These DLL files are also implemented in Borland Delphi 5. The accompanying Delphi projects are implemented completely in the project source file.

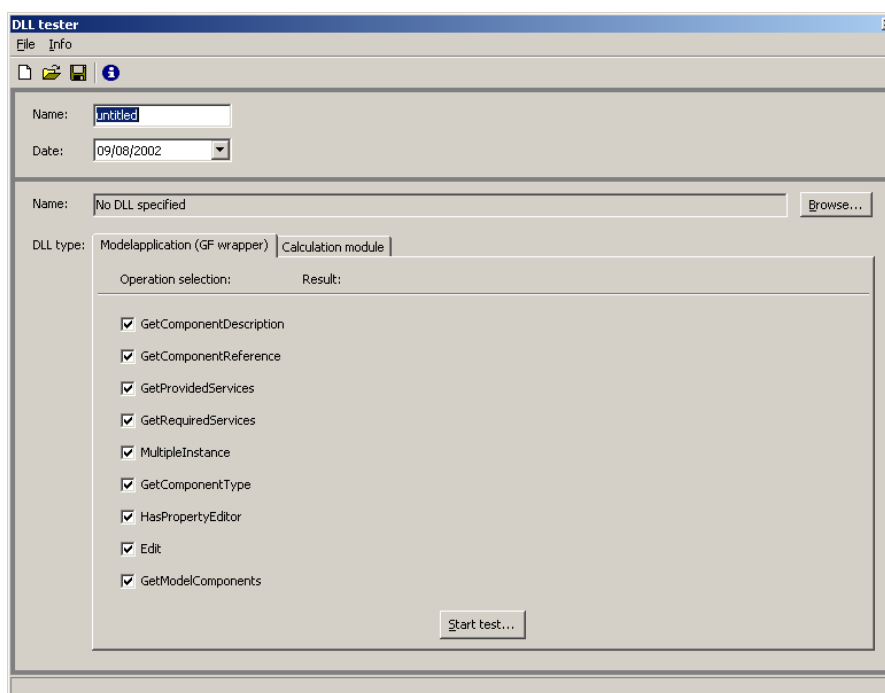
3 User Reference

3.1 General info

The DLLTESTER component is a tool to check Dynamic Link Libraries (DLL's). These DLL's can be GF model wrappers and GF calculation modules. The following tests are included:

- Test the availability of a specified DLL;
- Test of loading the DLL into memory;
- Test the availability of functions which are part a GF ModelApplication (model wrapper);
- Test the execution of functions which are part of a calculation module (DLL).

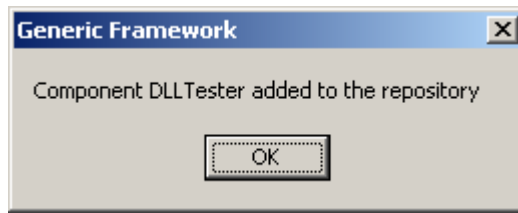
The settings which are required in these tests can be specified in a graphical user interface which is shown below.



3.2 Installing the DLLTESTER program

The DLLTESTER component is a GF component which can be used in the GF program. The DLLTESTER component is a dynamic link library (DLL)³ named 'DLLTESTER.DLL'. This file must be located on your local harddisk. To install the DLLTESTER component in the GF program, this file must be opened in the GF registration wizard. This wizard is started when the 'Edit|Register' menu-item is selected in the GF main form. A open-dialog is presented where the location of the file 'DLLTESTER.DLL' can be specified. A successful registration of the DLLTESTER ends with the message presented below.

³ In navigation-programs like Windows Explorer library files such as dynamic link libraries (extension: DLL) are often not shown, because manual removal of these files can harm the operating system or installed programs. To make sure library-files like DLLs are shown, select the 'Tools|Options|View' menu-item in Windows Explorer and select the option 'show hidden files and folders'.

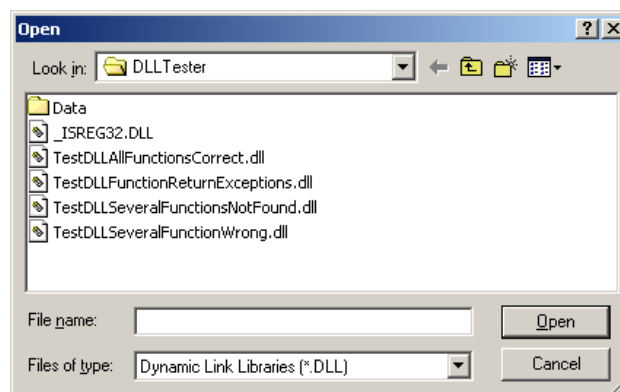


3.3 Set up a DLL test

Because the DLLTESTER program is a GF component, this program can only be used in a GF case. The following steps must be taken to use the DLLTESTER program:

- Start a new case in the GF program using the 'open case' menu item;
- Drag the DLLTESTER item from the component repository to the canvas of the case;
- Start the DLLTESTER 'Component Properties' form by double clicking the DLLTESTER instance or selecting the 'Component | Properties' menu item;
- Start the DLLTESTER program with the 'Property editor' button.

The DLLTESTER program starts with a new test project using the project name 'untitled' and the actual date. The first step (after possibly changing the project name and date) is to select a DLL to test. This selection is started by using the 'browse'-button on the main form of the DLLTESTER program. An open dialog screen enables the selection of a DLL.



After selecting a DLL, a selection can be made between the test of a GF ModelApplication and a calculation module. Within this selection, functions can be selected/deselected. When a calculation module is selected, also function names can be changed (**Note:** To call functions which are exported by a DLL, the calling conventions which are used are case sensitive. A function GetStarttime will not be found by the DLLTESTER program if the function name getstartTime is entered!)

3.4 Execute the DLL test

To start the test of the selected DLL and selected functions, the 'Start test'-button is to be used. After this button is pressed all selected functions are searched for and (if found) a call to these functions is performed. When a function is tested, three possible results can be presented:

- *This operation was not found:*

The tested function could not be found, execution of the function is omitted.

- *An error occurred while performing this operation:*

The tested function is found, but execution of the function results in an error. For the functions Check, Init, Start and Finalize this result is also shown when the return value is 'false', that is the return value can successfully be read by the DLLTESTER. This means

that a function that is not executed correctly and returns a false appears like a function that does not work at all or exports a wrong data type.

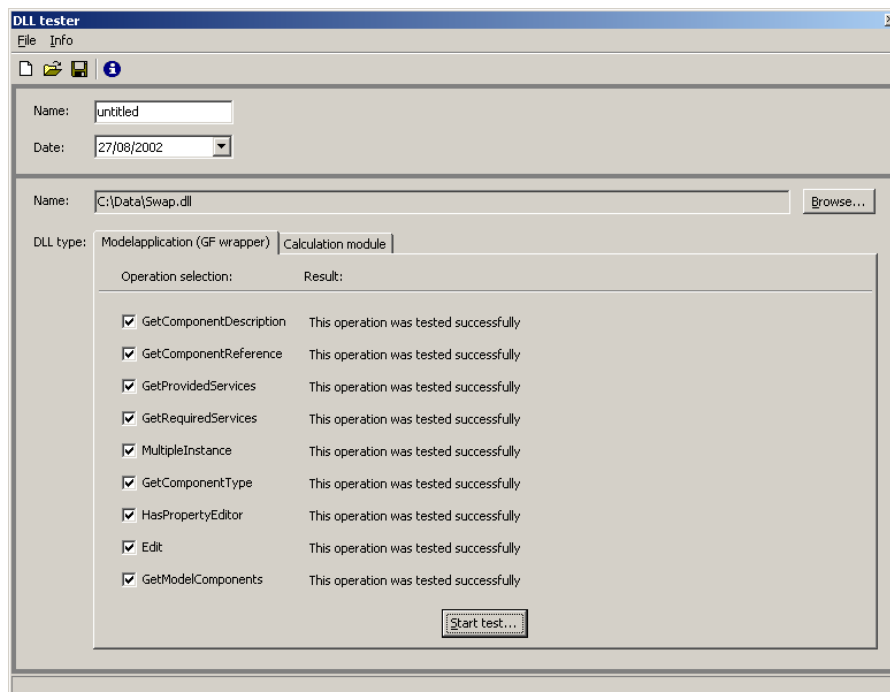
- *This operation was tested successfully:*

The tested function is found and successfully executed. This message is displayed if the functions Check, Init, Start and Finalize return 'true' or 'orSuccesfull⁴'.

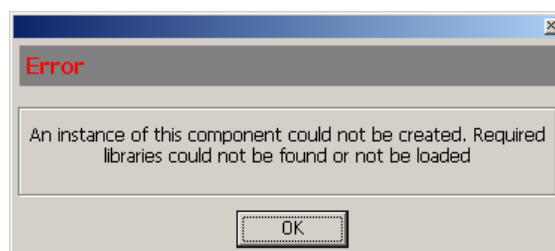
The actual function output of the following functions do not influence the message displayed by the DLLTester: GetStarttime, GetEndtime, GetCurrenttime, GetNexttime, GetComponentDescription, GetComponentReference, GetProvidedServices, GetRequiredServices, MultipleInstance, GetComponentType, HasPropertyEditor, Edit, GetModelComponents

Tip 1: Always check messages in the available error-files if the functions Check, Init, Start and Finalize return "An error occurred while performing this operation"

These results are presented besides the function specification. An example of a test result, using a DLL which provides all selected functions is shown below.



If a GF ModelApplication wrapper is tested, the DLLTESTER program attempts to load a ModelApplication object. When this object is not provided by the DLL, a GF error message is presented.



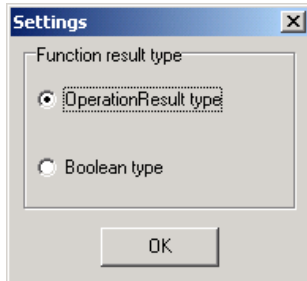
⁴ 'orSuccesfull' is a Generic Framework type identifier.

3.5 Test function settings

There are three functions which provide specific function settings, which are all part of the calculation module test:

- Init function

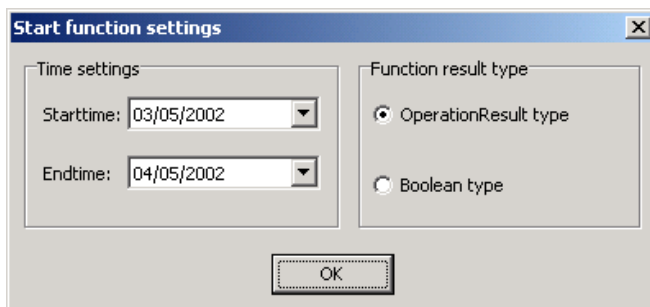
By default, the TOperationResult type is used as the Init function result type. This can be changed to a Boolean result type using the 'Settings...' button.



- Start function

The Start function uses Starttime and Endtime arguments which can be set on the 'Start function settings' form which is loaded by using the 'Settings...' button. These arguments are optional. A DLL with a Start function without arguments can be tested also. The Starttime- and Endtime-settings are ignored in that case.

By default, the TOperationResult type is used as the Start function result type. This can be changed to a Boolean result type.



- Finalize function

By default, the TOperationResult type is used as the Finalize function result type. This can be changed to a Boolean result type using the 'Settings...' button.

Warning: Reproducing results – multiple runs of the DLL-tester.

Individual DLL-routines may open streams (files) or load other DLL-routines which are freed by other DLL routines. As a result, if individual routines are tested, repetitive testing of the functions may cause problems, even leading towards a complete "crash" of the Generic Framework. Furthermore, even if all functions are tested problems may arise if e.g. the function "check" returns false due to missing files. All functions will return "success" but a re-run is likely to fail and potentially cause a system crash.

Tip 2: Build in a memory and files cleaning procedures if the return value of a DLL-function is false for individual DLL functions.

Tip 3: Check the windows task manager to see if any DLL's are still loaded.

Tip 4: External DLL functions should be well documented, to avoid testing with erroneous function calls.

3.6 Saving test settings

The general (project name and date) and function settings (function selections and function names) can be stored to file for later use. When a new name is entered for the test, this name is automatically used as filename to store. This filename can be changed by changing the project name or using the 'save as' option. The 'save as' option uses a save dialog, which is also started when the project name is 'untitled'.

4 Test report

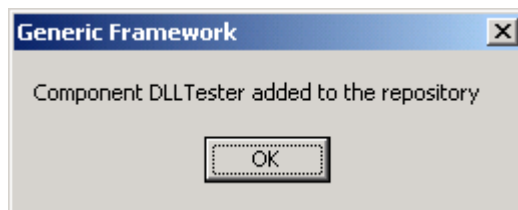
4.1 Introduction

The test of the DLLTESTER component is divided into three sections:

1. Component Registration test in Generic Framework v 0.9 using Windows 98, Windows NT4 and Windows 2000;
2. Basic run test on above mentioned operating systems;
3. Functional test using 5 scenarios:
 - a. Test with calculation module DLL which provides all functions;
 - b. Test with calculation module DLL which provides a subset of the required functions;
 - c. Test with calculation module DLL which provides all functions, but a subset of these functions return an error code;
 - d. Test with calculation module DLL which provide all functions, but a subset of these functions raise exceptions when called;
 - e. Test of the SWAP ModelApplication wrapper.

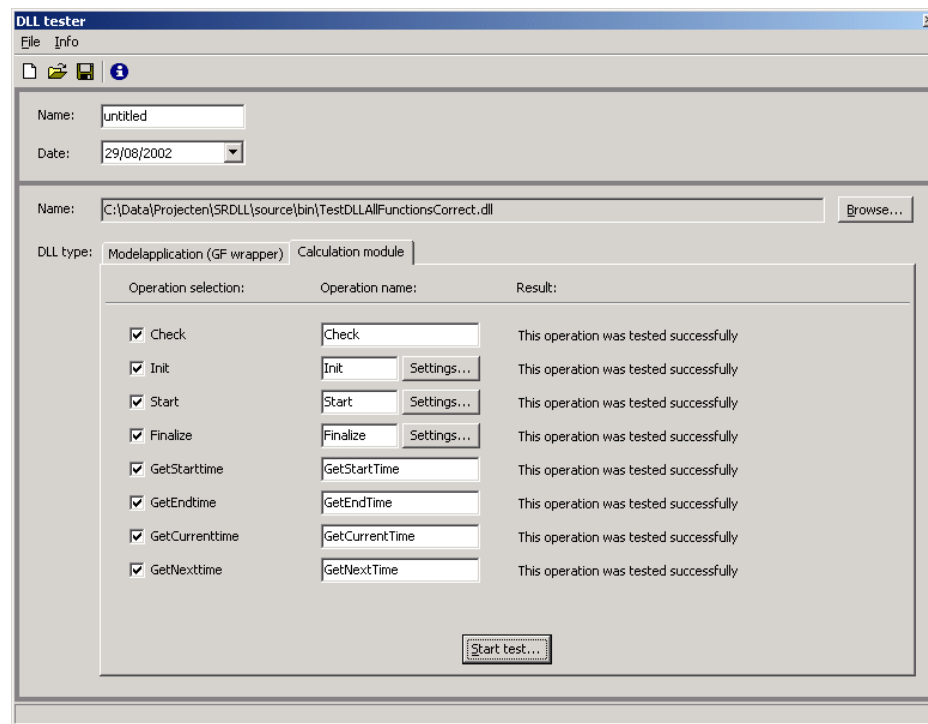
4.2 Test results

1. The DLLTESTER was registered in the GF application running on Windows 98, Windows NT4 and Windows 2000. The registration was successful on all operating systems:

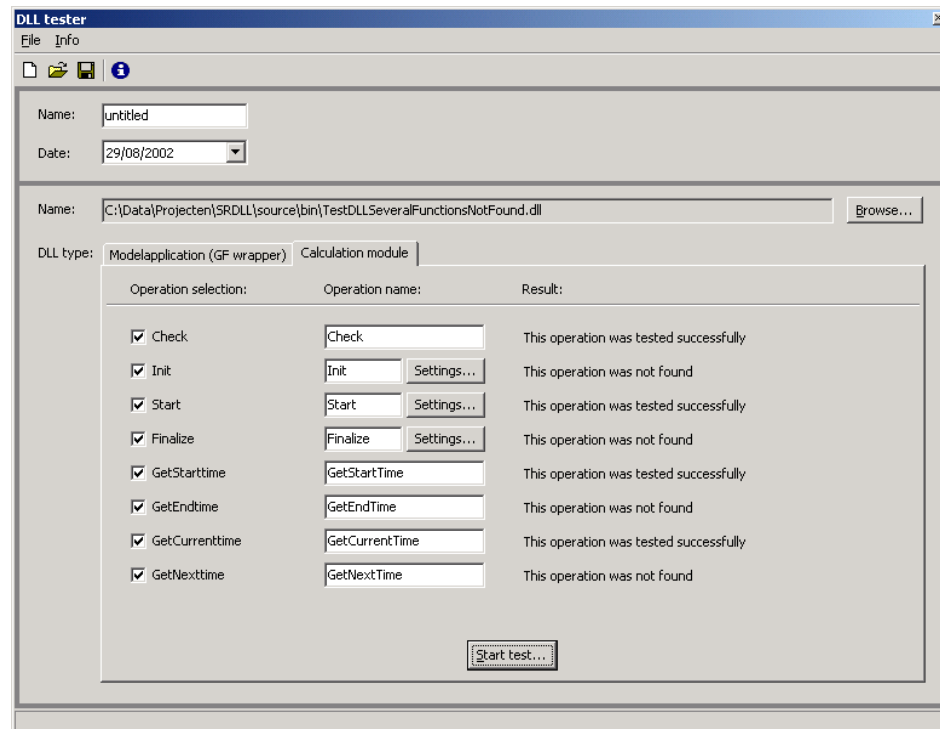


2. On above mentioned operating systems, the DLLTESTER component is used once where a new test project is defined, executed and saved. In a second run, the saved project was loaded and the test was repeated. The first and second test gave identical results.

3. Each of the defined scenarios was executed by starting the GF application, starting a new case, adding the DLLTESTER component to the case and open the component property screen of the DLLTESTER component.
 - a. Test with calculation module DLL which provided alle functions correct:
Library: TestDLLAllFunctionsCorrect.DLL
Expected Result: result message: 'this operation was tested successfully'
Found Result:



- b. Test with calculation module DLL which provided a subset of the required functions:
Library: TestDLLSeveralFunctionsNotFound.DLL
Expected Result: The functions are implemented in a way that the list of result should be alternately 'this operation was tested successfully' and 'this operation was not found'.
Found Result:

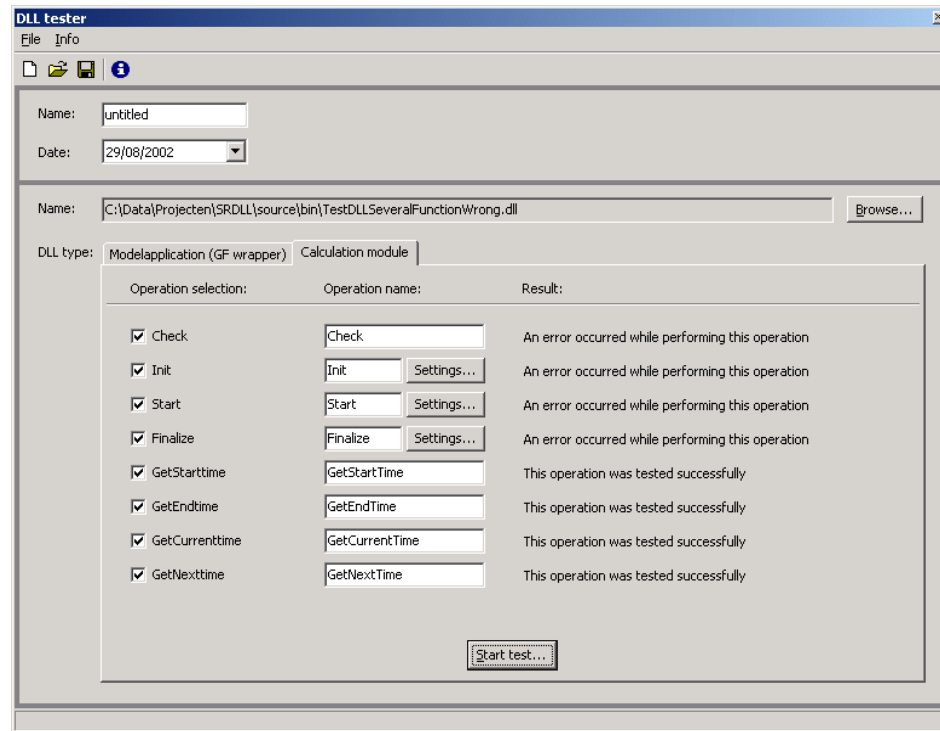


- c. Test with calculation module DLL which provided all functions, but some return error code:

Library: TestDLLSeveralFunctionWrong.DLL

Expected Result: The functions Edit, Check, Init, Start and Finalize return 'an error occurred while performing this operation' and the remaining function return 'this operation was tested successfully'.

Found Result:

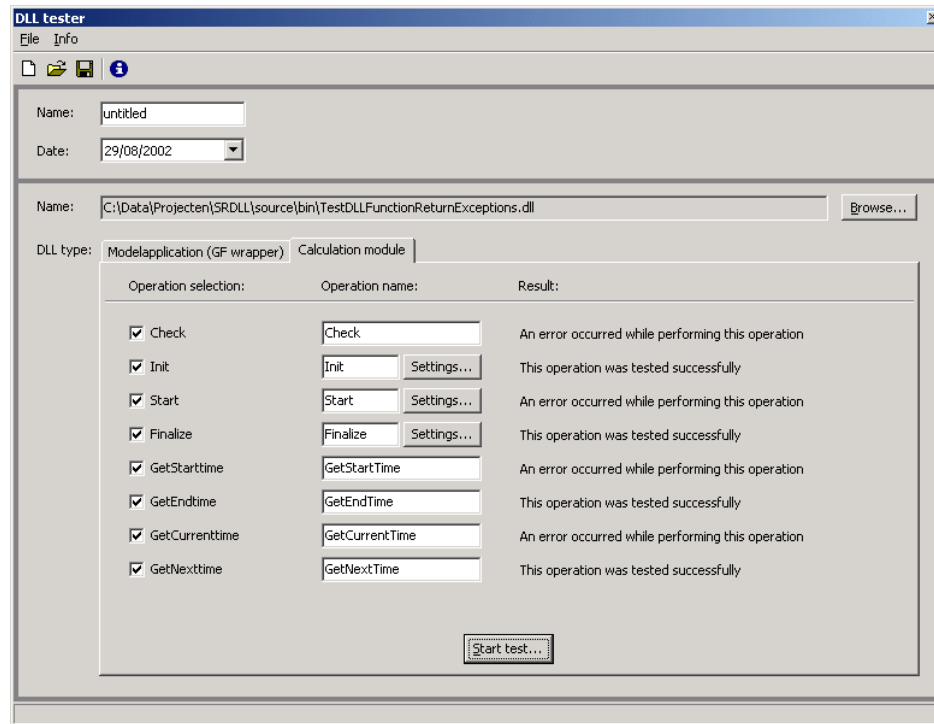


- d. Test with calculation module DLL which provided all functions, but some raise exceptions:

Library: TestDLLFunctionReturnExceptions.DLL

Expected Result: The functions are implemented in a way that the list of result should be alternately 'an error occurred while performing this operation' and 'this operation was tested successfully'.

Found Result:

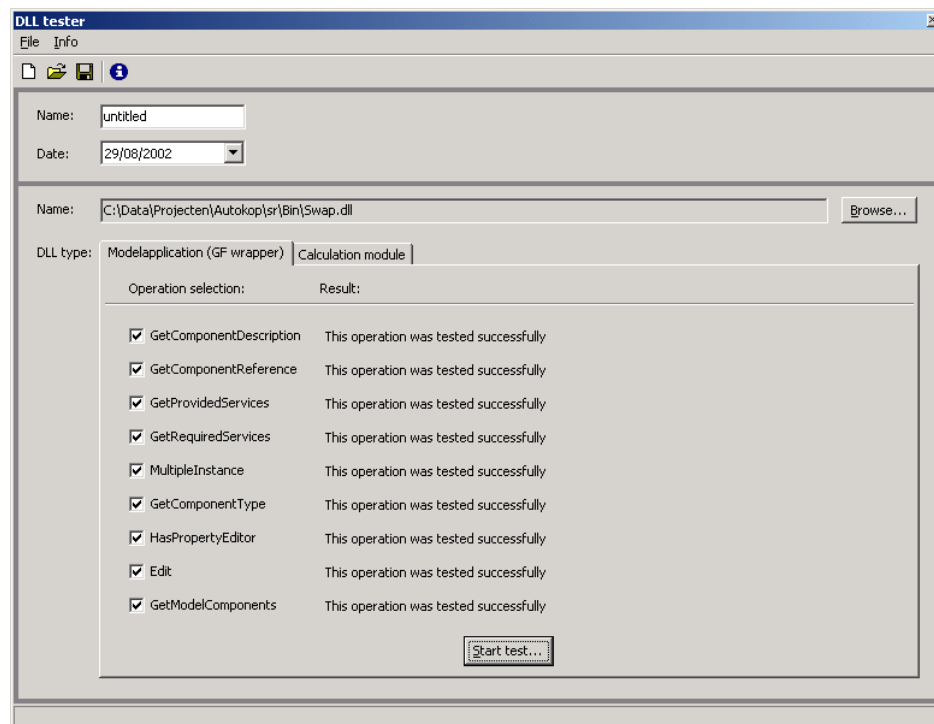


e. Test of SWAP ModelApplication wrapper:

Library: Swap.DLL

Expected Result: result message: 'this operation was tested successfully'. When calling the 'Edit' function, the SWAP component property screen is shown.

Found Result:



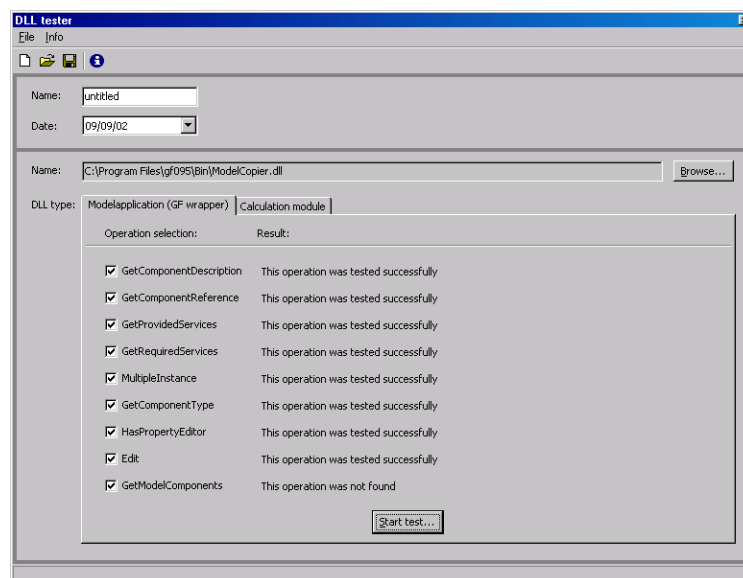
5 RIZA test report

5.1 Documentation

The documentation has been adapted based on first comments and is considered to be of good quality. As agreed RIZA has finalized the report. The most important additions are warnings about the usage of the DLLTESTER, focussing on tips for implementing DLL-functions in computational kernels.

5.2 DLLTESTER software functioning

The DLL-tester was first tested on the MODELCOPIER software. The results were satisfactory:



The DLL-tester was tested on the SWAP DLL and the results presented by W!SL were reproduced. However, testing it on the SWAP computational modules could not be executed due to lack of knowledge about the DLL functions in the two swap computational modules.

The DLL-tester was tested on RIZA's (water) distributionmodel (DM), which is a FORTRAN DLL. This test proved to be difficult, since the DM.DLL does not include safeguards for open streams and memory problems if an individual function does not work correctly and streams are not properly closed. However, the test ultimately was successful, and the results have been included in the User Reference as warnings, which are accompanied by tips to avoid the problems.

5.3 DLLTESTER software source

The source files contain simple and clear headers. Other comments are only limited available. Declarations of variables are largely self-explaining.

The source code, in combination with the reported architecture chapter in this report fit the demands of the project.

5.4 Conclusions

The products of the DLLTESTER project are of good quality and accepted by RIZA.