Atos

TEMPPO Designer (IDATG)

Version 14.1

Tutorial

August 2011

Publisher:
Atos IT Solutions and Services GmbH
Si SOL TAM
Gudrunstrasse 11
1101 Vienna, Austria

Document Management

History of changes

Version	Status	Date	Person resp.	Reason for Change
9.1	Ready	16.9.01	Weihs Bettina	Creation
10.0	Ready	25.10.01	Adriana Strejčková	Update, proofreading
10.2	Ready	10.10.03	Stefan Mohacsi	Update
11.0	Ready	20.07.04	Stefan Mohacsi	Update
11.2	Ready	25.04.05	Wiltrud Breuss	Revision, Update
12.0	Ready	23.01.06	Wiltrud Breuss	Revision, Update
12.1	Ready	31.07.06	Wiltrud Breuss	Revision, Update
12.2	Ready	29.09.06	Wiltrud Breuss	Revision, Update
14.1.	Ready	05.08.2011	Stefan Mohacsi	Update

Document was created using the following tools:

Microsoft Office WinWord 2003

Corel Paintshop Pro 10.0

Contents

1 Introduction

1.1 Purpose of the Document

The purpose of this tutorial is to explain the usage of the test specification and generation tool TEMPPO Designer (IDATG).

1.1.1 Using this Tutorial

This tutorial provides two examples that demonstrate the usage of IDATG. The first example 'Login' is very simple and shows the basic sequence of steps to get test cases.
The second example 'Tennis' is more extensive and shows the full power of IDATG. It also explains how to tackle typical problems that arise during the specification process. Both demo applications are delivered together with this tutorial.
Basic knowledge of the test process and its purpose is presupposed.

1.2 Validity of the Document

This tutorial is valid for IDATG V14.1-

1.3 Definitions of Terms and Abbreviations

	Exercise to practice the discussed matter
	Solution and explanation of the exercises
Test Case (TC)	A set of inputs, execution conditions, and expected results developed for a particular objective, such as to execute a particular program path or to verify compliance with a specific requirement.
Test Case Execution	The execution of a test case by a human user or capture/replay tool followed by the verification of the results.
Test Case Generator (TCG)	A tool that automatically generates test cases based on a formal specification of a system.
WinRunner	A commercially available capture/replay tool for automatic test case execution

1.4 Relationship to other Documents

- IDATG User's Guide

2 IDATG Overview

IDATG serves to create test scripts for various capture/replay tools starting from a formal (abstract) specification. This chapter explains which kinds of test cases can be generated by IDATG. It also shows the role of the tool in the test process and its place in the tool architecture.

2.1 Testing with IDATG

IDATG is used for **black-box testing**, which means that the source code of the application is not required for testing. While the original purpose of IDATG has been the support of automated GUI testing, now the tool can also be applied for different forms of API testing. The following types of test cases are generated:

- **Graph-oriented** test cases based on hierarchical sequence diagrams ("task model"). This method can be used for both GUI and API testing. By defining frequently performed test sequences as "building blocks", the maintainability and clearness of the specification can be significantly improved.
- **Data-oriented** test cases: at the moment for each record in the data set a separate test case is created. They are also based on task models.
- **Transition** test cases that cover the dynamics and semantics of the GUI. They can be used for stand-alone testing of single dialogs or for integration testing.

The amount of test cases generated by IDATG depends on the level of detail provided in the formal specification. For a specification which describes the behavior of the user interface rather coarsely, only a small number of test cases can be generated. If the GUI-behavior is specified with great detail, a large amount of test cases is generated.

2.2 Tool Architecture using IDATG

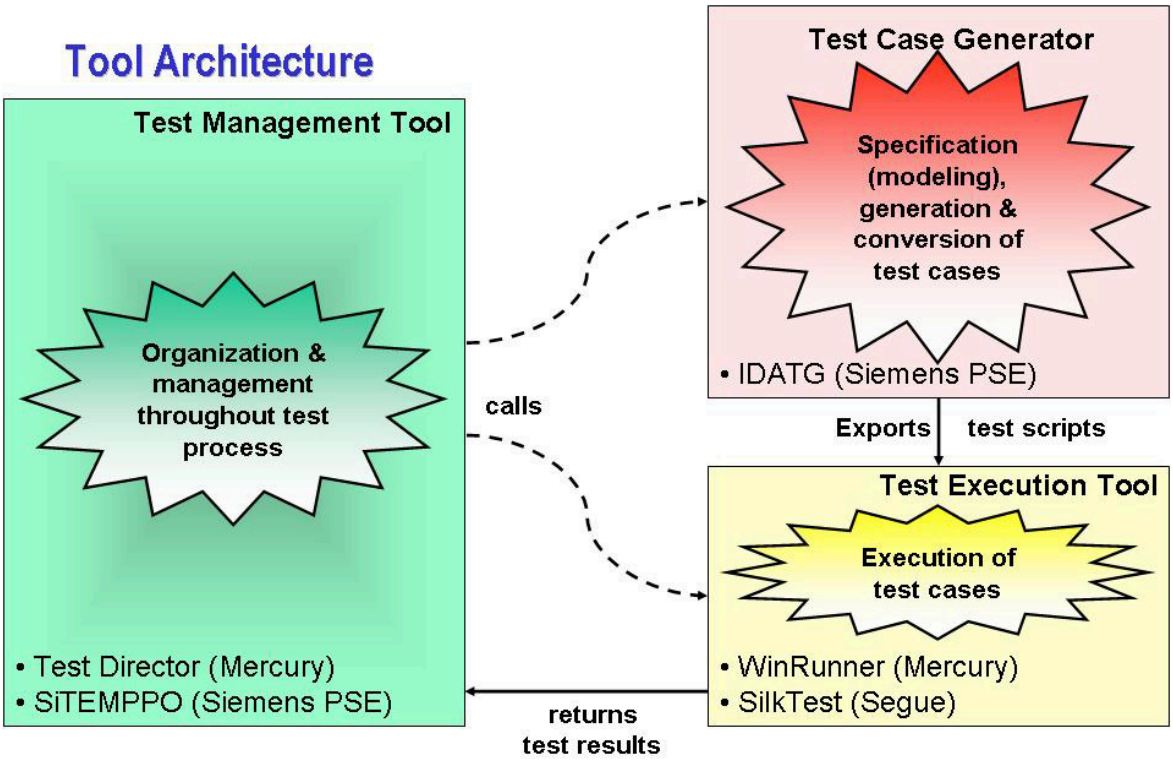


Figure 1 - Recommended Tool Architecture using IDATG

Test Management Tool

A test management tool allows the tester **to administer** system **requirements, test cases** and **test results,** organize **test suites** and manage **bug-tracking.**

Test Case Generator (IDATG)
IDATG (Integrating Design and Automated Test Case Generation) is designed for **GUI and API test specification** and the **automated generation of test cases**.

Test Execution Tool
IDATG creates test scripts in a general format that can be converted into scripts for test execution tools like QuickTest, SilkTest and WinRunner. These tools **execute the test scripts** by simulating user actions (e.g., mouse clicks) and compare the actual results with the expected ones.

2.3 Test Process

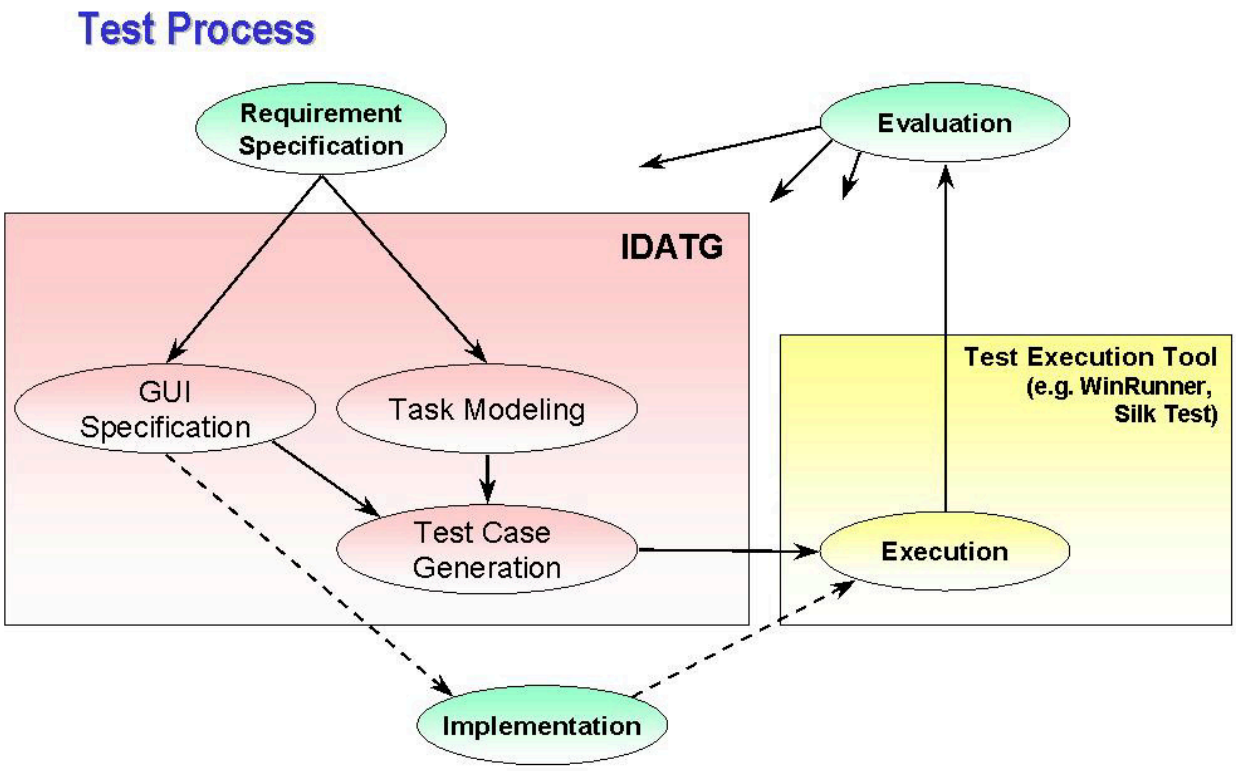


Figure 2 - Test Process using IDATG

GUI Specification
The user works with a **realistic image** of the GUI. The image is captured directly from the screen by IDATG's **GUI Spy** or it is created manually with the IDATG editors.

Task Modeling
The steps needed to perform a task are defined in the **Task Flow Editor**. Building blocks (reusable tasks, sequences of steps) greatly improve the test case design (speed, maintainability).

Test Case Generation
IDATG can generate test cases of three types: **graph-oriented** (system), **data-oriented** (system) and **state-based** (unit).

Execution
Test cases generated by IDATG are then **executed by a test execution tool** (e.g. WinRunner, Silk Test). After performing tests, the test execution tool produces a test report. Test reports can later be evaluated by testers.

2.4 Important Terms

2.4.1 Window

The term **window** is used here as an equivalent for a GUI object. Thus, dialogs, buttons, input fields, static texts, etc. are all equally seen as windows. Each window may contain an unlimited number of child windows. This view corresponds to the structure of most operating systems (e.g., MS Windows).

2.4.2 Task

A **task** is derived from a functional requirement of the specified application. These requirements are usually listed in the software requirements specification, where they can be organized in a hierarchical tree structure. This structure can be used for the corresponding tasks too. Each task is defined by a task flow consisting of steps.

2.4.3 Step

The basic term of the IDATG language is the **step**. A step can either be an atomic step containing a single test instruction or represent an entire building block task. Atomic steps can be described by the following information:

- The **test command**. For GUI testing, the test command is also called **event**, because it usually represents a user input that triggers the step (e.g. a mouse click).
- The semantic **conditions** that must be fulfilled before the step can be executed (e.g., a certain button must be enabled).
- The **actions** executed during the step (e.g. a window opens or a button becomes disabled).
- For GUI testing, the **start and destination window** of the step (in other words, the focus position before and after the step).
- For API testing, it may be necessary to define the **test driver** that will execute the test command.
- The **expected results** are defined indirectly by introducing verify steps that contain test commands that compare the actual results with the expected ones.

During your work with IDATG, you will encounter different forms of steps:

- **Task Steps**
  For graph-oriented and data-oriented testing, the sequence of steps is determined by you. You can define it in a task flow diagram.
- **Transitions**
  IDATG also supports another test method called transition testing in which you do not define a certain step sequence. You just enter individual steps and IDATG determines a proper

sequence based on the semantic information you provide for the steps (only applicable for GUI testing)

- **Test Steps**
  IDATG can generate automatically various types of test cases that - naturally - also consist of steps. Like tasks, test cases can be displayed as a sequence diagram.

2.4.4 Condition

Conditions are Boolean expressions that define when a certain step can be executed (for example, in the window 'Login', it is allowed to press ‘OK’ only after ‘user name’ and ‘password’ have been entered).

2.4.5 Action

Actions are used to describe the effects of a certain step on the GUI (for example, the Login window is closed and the main window is opened after pressing ‘OK’). These effects influence (control) the generation of test cases.

2.4.6 Parameter

Parameters belong to tasks. They work like parameters of functions.

3 Installation of IDATG and the WinRunner-AddIn

3.1 System requirements

- Hardware: Pentium PC, min. 800 MHz recommended
- Operating system: Windows 98/NT/2000/XP
- Free memory space required: 256 MB
- Additional software required: Segue's *SilkTest* **OR** Mercury's *WinRunner*

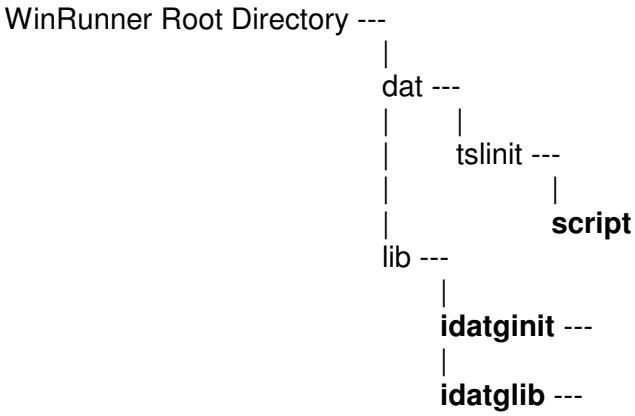
3.2 Installation instructions

1. You have to be logged in with administrator rights before you can install new software.

2. Call the self-extracting file `setup.exe` that installs the IDATG files to your hard disk and creates shortcuts in the start menu. A sample project is also included in the package.

3. If you want to run your tests with Mercury's WinRunner, you have to perform the following steps before the test scripts can be executed:

   3.1. Install WinRunner (please refer to the WinRunner documentation for instructions)
   3.2. Move the directories `idatginit` and `idatglib` that can be found in your `Idatg\WinRunnerAddIn` directory into the `lib` directory of WinRunner.
   3.3. In the file *script* in the *dat\tslinit* directory of WinRunner you have to insert the line `call "idatginit" ( );`



3.3 License Installation

Before you are able to use IDATG, you need a valid license for your computer. The license is PC-specific and usually restricted to a certain version of IDATG and a limited period.

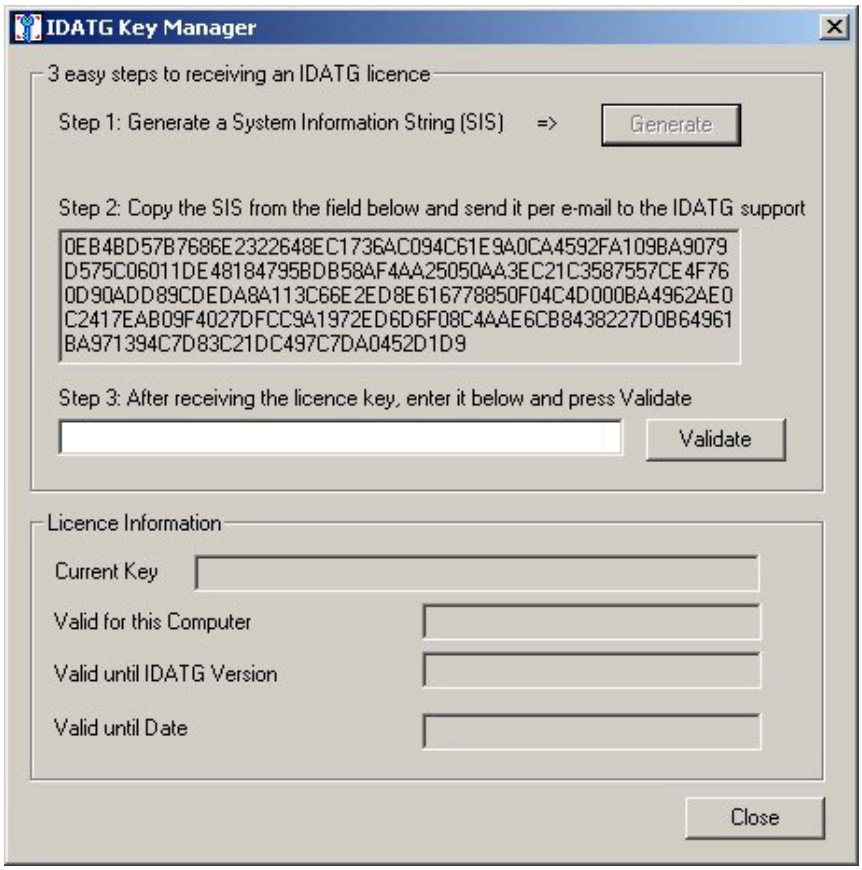


Figure 3 - Key Manager

To receive a license, just follow 3 simple steps:

- Step 1 - Start the Key Manager and press '**Generate**'. A string will be generated that contains encoded information identifying your PC. This string is called System Information String (SIS).
- Step 2 - Copy the string into an e-mail and send it to the IDATG support. (Just select the string with the mouse, press Ctrl+C to copy it and Ctrl+V to paste it into your e-mail).
- Step 3 - You will receive a reply e-mail containing your license key. Copy the key into the Key Manager and press '**Validate**'. After confirming an information message, your license will be installed.

If you wish to use IDATG on different PCs, please run the Key Manager on each of them to generate an individual SIS for each computer.

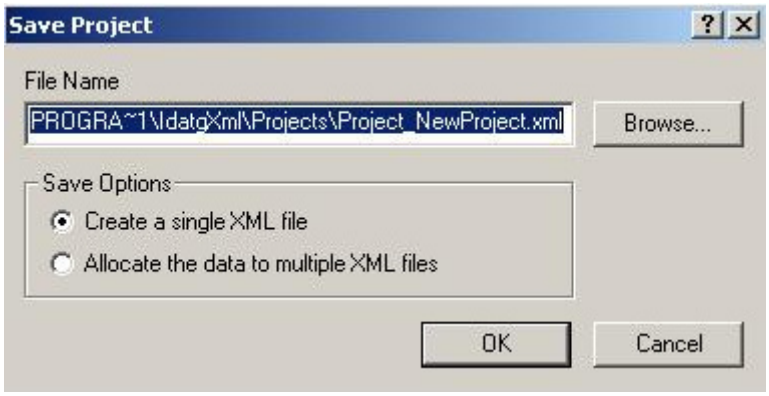
You can start the Key Manager at any time to view information about the installed license or to install a new one. If you should experience problems with your license or you need an update, do not hesitate to contact the support.

4 The Project 'Login'

The project 'Login' is a very simple example to show a basic sequence of steps to create a test case and transform it into a test script for a capture/repay tool.
The first step is to **create a new project**. After creating a new project, all information about GUI elements, tasks, and test cases will be stored in IDATG's xml-files.

4.1 Creating a Project

- Start IDATG by double-clicking on the IDATG icon in the Windows start menu or by executing the file `idatg.exe`. Press **'Cancel'** in the Open File dialog.
- Create a new project using the menu item **'Project | New'**.
- Enter the properties (name, appropriate GUI Builder and the desired output format) of the project and press **'OK'**.
- Next you get a window 'Save Project'. Enter the path where you want to store your test cases and click 'OK'.

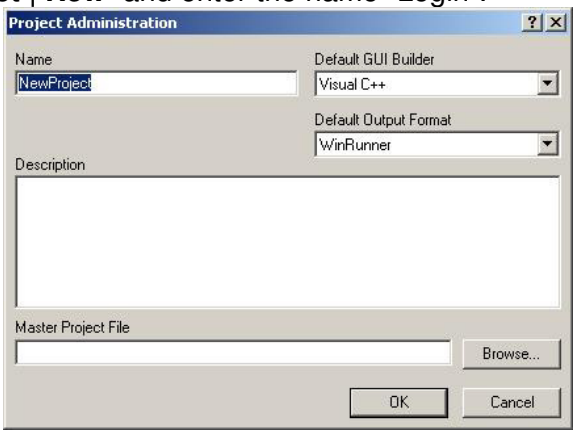


Exercise 1:

Create a new project 'Login', and set the default GUI builder to 'Visual C++' and the test-execution tool to 'WinRunner'.

Solution 1:

Install IDATG (see chapter 3) and start it.
Click the menu item **'Project | New'** and enter the name "Login".

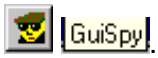


Click 'OK'
Click 'OK' in the 'Save Project' dialog.

4.2 Recording a Window with the GUI Spy

The windows of the user's application can be directly captured with IDATG's **GUI Spy**. You could also define them manually with the IDATG editors, but of course, it saves a lot of work if you simply capture them from the screen.
The GUI Spy is a very simple and efficient tool. No resource script or other source code is needed. The usage is very similar to other GUI spies like those of the Microsoft Developer Studio or common capture/replay tools.

Recording a window:

- Press the button 

.
- Start your application and place its window beside the IDATG window.
- Click left on the crosshair symbol in the GUI Spy and **keep the mouse button pressed.**
- Drag the crosshair symbol over some windows. Note that the window under the cursor is highlighted with a red frame and the information about the window is displayed in the list box of the GUI Spy. IDATG windows are ignored.

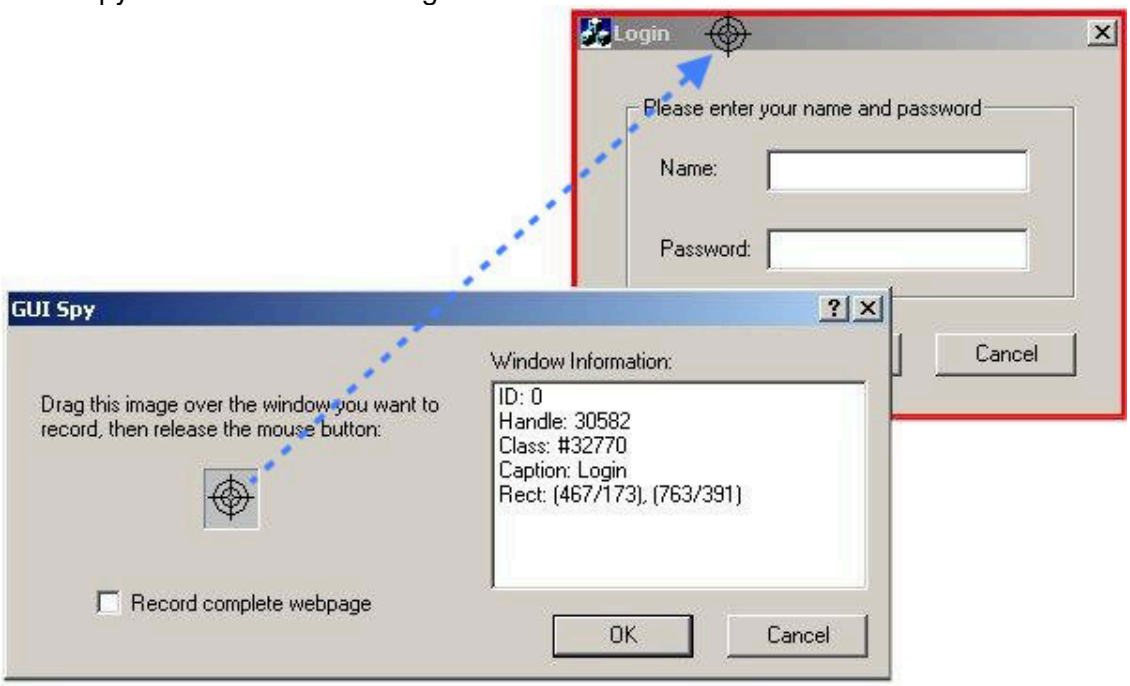


Figure 3 - GUI Spy and recorded window

- When you have reached the window you wish to record, release the mouse button. Note that exactly the window under the crosshair symbol and its child windows are recorded, but not the parent window.
- You may change your selection by dragging the crosshair symbol another time or confirm it by pressing '**OK**'. In this case, the new window is added to the IDATG project: it is shown in the **Application Editor**, and a **Window Editor** is opened for the window.

Exercise 2:

Record the main window of the example 'Login' as described above.

☑ Solution 2:

- Start the demo application ‘Login.exe’ in your tutorial directory.
- Open the GUI Spy of IDATG
- Drag the crosshair symbol over the Login window, and press ‘OK’ in the GUI Spy window.

This should be your result:

Figure 4 - Window Editor 'Login'

4.3 Application Editor - Window Hierarchy

Let us have a short look at the user interface of IDATG. The main tool for navigating through a project is the **Application Editor** that is usually located at the left side of the IDATG window. It contains two tab pages showing the hierarchy of the application’s windows and tasks. Most of the other IDATG editors will appear on the right side.

As already mentioned, the term **window** is used here as an equivalent for a GUI object. The hierarchy levels of the window tree represent the parent-child relationships between the windows. The editor always contains the project as root node, from which the entire window hierarchy is descending. Menus and menu items are also shown in the tree.

Windows that are the start state of a transition are displayed blue and their ancestors are blue as well. This color representation allows the user to see immediately which parts of the GUI have transitions specified.

Selecting a Window

Select a window by clicking on its name in the tree. Several toolbar buttons become active that allow you to edit the behavior and properties of the window. If you click the right mouse button, a menu of editing possibilities opens. The selection of the root node has no effect.

Editing the Window ID

Edit the ID of a window in the same way you would edit a file name in the Windows Explorer (Select ID, press F2 or click again). It must be unique, may only contain alphanumeric characters and '_' (underscore) and may be up to 50 characters long. You can also use the **Window Properties Editor**  to edit the window ID. Use IDs that are meaningful to you!

Choosing a (new) Start Window

The start window is the one that appears first when the tested application is executed. To define or change it, simply select the desired window and press '**Set as Start Window**' in the '**Edit**' menu. In the **Application Editor,** its name is displayed **bold**. This information is necessary for the generation of GUI test cases, especially if the test execution tool is Silktest.

Note: Only a window that has no parent and is visible and enabled can be a start window.

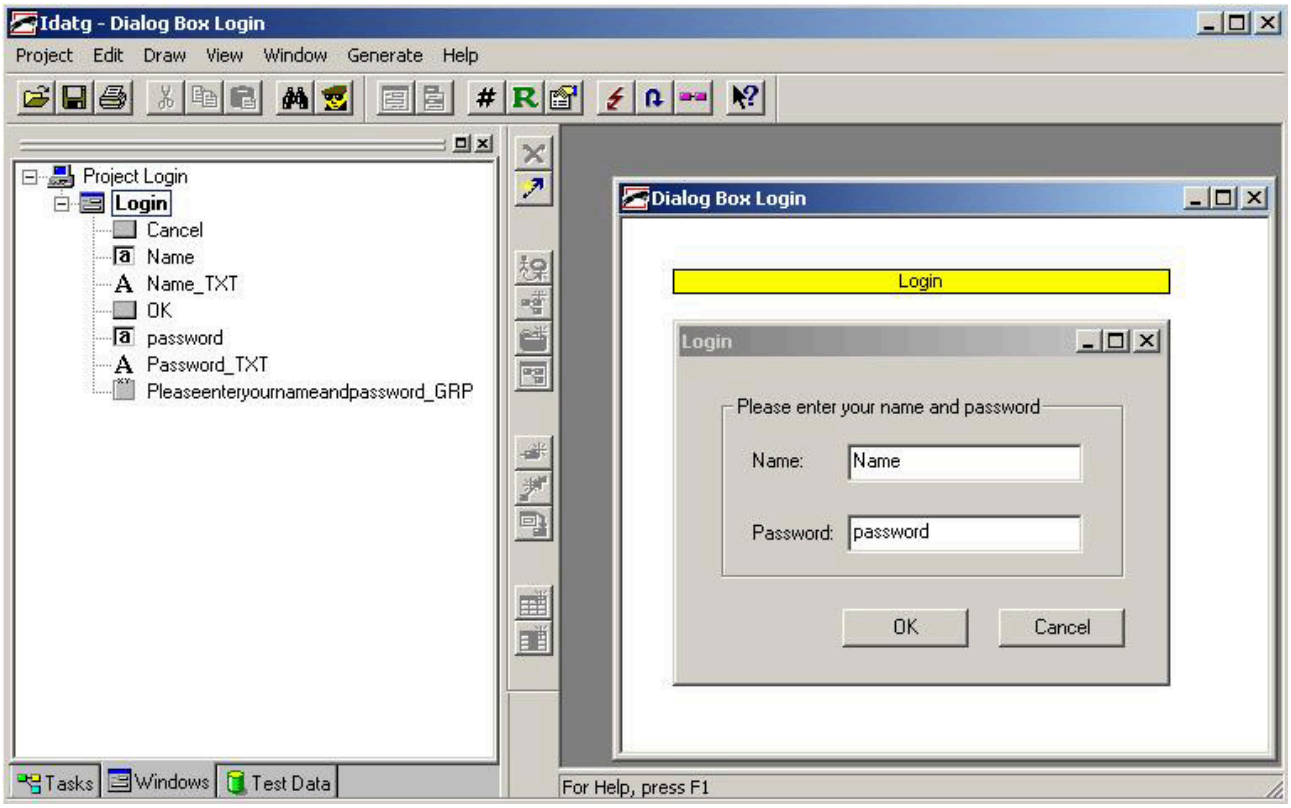
 **Exercise 3:**

In order to make the GUI description easier to understand, it is sometimes advisable to change the window IDs that have been generated by the GUI Spy. Therefore, rename the input field 'Pleaseenteryournamea' to 'Name' and 'Pleaseenteryournamea1' to 'Password'. Set 'Login' as the start window.

☑ Solution 3:

- Expand the tree of ‘Login (Dialog Box)’.
- Select “Pleaseenteryournamea” in the windows pane, press F2, then type “Name”.
- Select “Pleaseenteryournamea1” and click again, then type “Password”. Or click “Pleaseenteryournamea1” with the right mouse button and select “Edit Properties”. Then type “password” into the Window ID and click ‘OK’.
- Click right on ‘Login’ and select ‘Set as Start Window’.

The result:



4.4 Defining a Task Model

In IDATG there are two kinds of tasks: Use cases and building blocks.
The basis of the **task model** specification is the software requirements specification of the tested application. It should contain a detailed description of all tasks (functions) the software has to perform from the user's point of view.

A **task** consists of **steps.** One **step** describes one test action**:**

- **What** (user event like select, click, input) and
- **where** (window like button, input field, etc.)
- A step can also refer to a **building block task** consisting of other steps. This allows you to construct your task model as a modular system.

Getting started:

- Define the **task** hierarchy in the **Task Editor** using the software requirements specification or a test specification. **This hierarchy does not reflect any dependencies!**
- Define the **task flow** of the most important use cases with the **Task Flow Editor** (e.g., login-do something-logout) by drawing **steps**. The steps are assigned to windows or building blocks.
- Going top-down through the task hierarchy, define the task flows of all other tasks. Wherever possible, use modular building blocks as subtasks.

4.5 Application Editor - Task Hierarchy (Task Editor)

The **Task Editor** shows the task hierarchy. The root node of the tree represents the project. It is recommended to enter **use cases** according to the project's software requirements.

Tasks that are self-sufficient use cases are marked with the icon , whereas building blocks look like this . A Test Case is the smallest unit that IDATG will generate and convert separately. It should be a stand-alone test case that does not depend on any other test cases.
Tasks whose flows have already been defined are displayed blue.

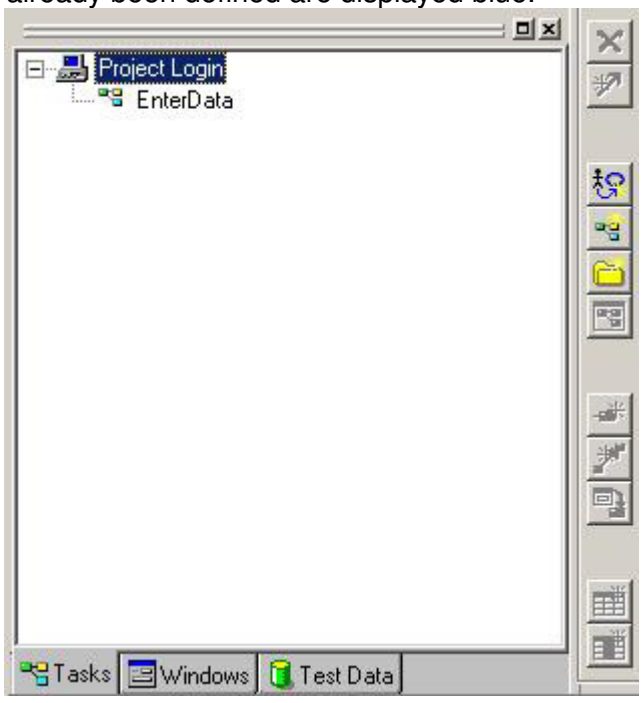


Figure 5 - Application Editor - Task Hierarchy

Adding a new Task

- Select the parent of the new task (the project or a folder)
- Press '**New Task**' in the toolbar or the menu if you want to create a use case. For a building block press 'New Building Block' or select 'New Building Block' from the context menu. The new task will be inserted as child.

Editing the Task Name

Edit the name of the task in the same way you would edit a file name in the Windows Explorer (Select name, press F2 or click again). It must be unique, may only contain alphanumeric characters and '_' and be up to 50 characters long. You can also use the **Task Properties Editor** to edit the task name.

Selecting a Task

Simply select a task by clicking on its name in the tree.

Editing Task Properties

The properties of the selected task may be edited by choosing the menu **'Properties Editor'** or by pressing the corresponding toolbar button.

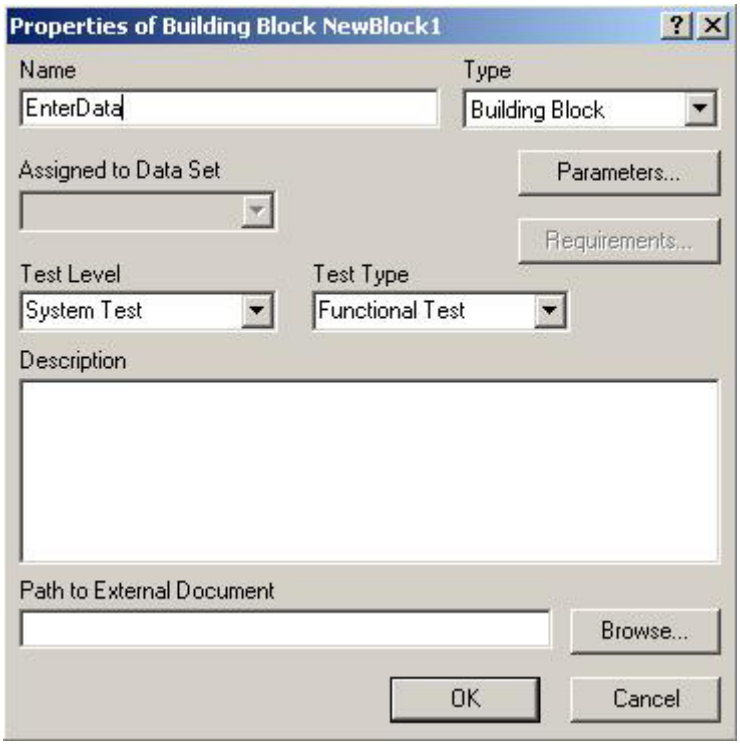


Figure 6 - Task Properties

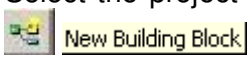
- The **Name** must be unique, may only contain alphanumeric characters and '_' and be up to 50 characters long.
- The **Test Level** refers to the test phase according to the General V-Model in which the resulting test cases should be used. Apart from the standard levels Unit Test, Integration Test, System Test and Acceptance Test, you may also enter a self-defined string.
- The **Test Type** can be used for a further categorization of the test cases. Apart from the pre-defined values like Functional Test, API Test etc. you may also enter a self-defined string.
- You can enter an arbitrary **Description** that explains the purpose of this task.

- In addition, you may enter the **Path to an External Document** in which information about this task can be found. (e.g. the requirements specification document). The '**Browse…**' button helps you to select the correct file.

Exercise 4:

Create a building block 'EnterData' and define its properties as 'System Test' and 'Functional Test'.

Solution 4:

1. Open the project 'Login'
2. Choose the tab "Tasks" of the Application Editor
3. Select the project as the Parent Folder. Define the task 'EnterData' by clicking on the button 

or selecting the menu item **'Draw | New Building Block'**.
4. Set Name to 'EnterData'
5. Choose 'System Test' for *Test Level* and 'Functional Test' for *Test Type* (see Figure 6 - Task Properties)
6. Press 'OK'
7. The result should look like Figure 5.

4.6 Task Flow Editor

In the task tree, double-click the desired task or select it and press the toolbar button  '**Edit Task Flow**'. The Task Flow Editor opens and allows you to define the sequence of steps that has to be performed to fulfill a task. In the Task Flow Editor, you can see the task flow as a directed graph.

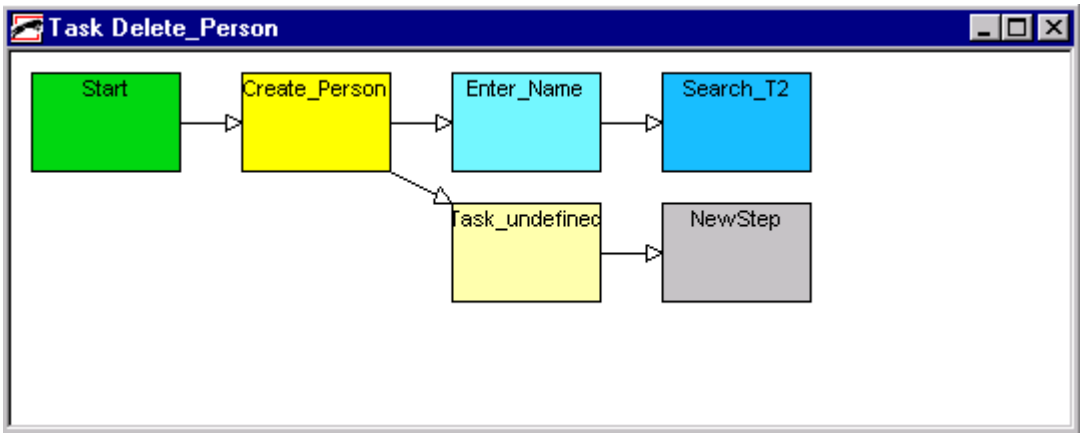


Figure 7 - Task Flow Editor with arbitrary steps

There are different kinds of steps, each represented by a different color:

Green: The **initial step** of the task flow. It is added automatically and cannot be edited or deleted.

Light Blue: **Atomic steps** that represent one single activity. For GUI testing, they are usually assigned to a window, but they can also be used for defining GUI-independent test commands or for expressing conditions and actions. A step becomes light blue, if either the event, start window, test commands, conditions or actions have been defined.

Blue: Atomic steps that are linked to a **transition** and are therefore only relevant to GUI testing. Note: If you edit such a step, in fact you edit the actual transition. This is particularly useful if the same step appears several times in your specification, since you only have to edit it once and all instances of it (i.e. all steps that are linked to the same transition) are changed automatically.

Yellow: **Composite steps** that refer to a building block. The task name is used as step name and cannot be edited.

Light Yellow: Same as above. The lighter color signifies that the **task flow** of the sub task is still **undefined**. IDATG tries to avoid such steps during test case generation.

Grey: **Undefined steps** that have not yet been assigned to a window or sub task and contain no information about event, test commands, conditions or actions. They are ignored during test case generation.

Steps are updated automatically after GUI changes like renaming a window, redirecting a transition etc. Only if a window, transition or task is deleted, all steps that refer to this object become undefined.

Selecting a Step or Connection

Steps and connections can be selected by clicking on them in the graph. The selected object is displayed with a red frame.

Selecting more than 1 Step

Multiple steps can be selected in different ways:

1. Click on an empty area and drag a rectangle over the desired steps
2. Hold the 'Ctrl' key while clicking on a step to add it to the selection. This can be useful when selecting steps from different areas of the task flow. By holding 'Ctrl' it is also possible to remove steps from the current selection.

Note: The start step cannot be added to a multi-selection.

Adding a new Step after a Step

- Select the predecessor step
- Press  'New Step', (or right-click the predecessor step and select 'Insert Step' from the context menu)
- An undefined step is created that is connected to the predecessor. If the predecessor already had a successor, a branch is created.

Adding a new Step between two Steps

- Select the connection between the two steps
- Press  'New Step'
- An undefined step is created between the two steps and connected to both of them.

Assigning an Object to a Step

This is the easiest method to define a step. When you select the object you want to use, IDATG creates a default event for this step, depending on the type of object you selected.

- Select the step
- Press the toolbar button  'Assign Object to Step', or right-click the step and select 'Assign Object to Step' from the context menu.
- All other editors are now in a special waiting state. Instead of performing their usual functions, the next selection is sent directly to the **Task Flow Editor**. You may select either a task or window in the **Application Editor** or a window or transition in a

Window or **Menu Editor**. If you select a task, it may not be a parent or super task of the current one.

- If the assignment has been successful, the color of the step changes accordingly.

Editing a Step

- Select the step and press the toolbar button  'Properties Editor', or double-click the step, or right-click the step and select 'Edit Step' from the context menu.
- The **Step Editor** appears and allows you to change the step properties. Depending on the type of step, different fields can be edited.

Viewing the Window assigned to a Step

If you want to view the window or transition that has been assigned to a step (blue), just select the step and press the toolbar button  'Edit Window' or  'Edit Menu', or right-click the step and select 'Edit Window' from the context menu.

Viewing the Sub Task assigned to a Step

If you want to view the task flow of the task that has been assigned to a subtask (yellow), just double-click on it or select it and press the toolbar button  'Edit Task Flow'.

Copying Steps

- Select one or more steps and press '**Copy**'  in the toolbar or the menu Edit >> Copy. CTRL+C works too. All connections between the selected steps are also copied (but no others). The task's start step can not be copied.
- Select a step or connection (may also lie in another task)
- Press '**Paste**' to insert the copied steps. If a step is selected, the copied steps are added as successors of this step. If a connection is selected, the copied steps are added between the start and destination step of this connection.

Moving Steps

The selected steps can be moved to another location by using '**Cut**'  and '**Paste**'. The result is equivalent to pressing '**Copy**', '**Delete**' and '**Paste**'.

Deleting Steps

The selected steps may be deleted by choosing the menu '**Delete**' or by pressing the corresponding toolbar button  . All successors of the steps will be connected to their former predecessors to avoid that they become isolated. It is not possible to delete the start step.

4.7 Step Editor

If you assign a window or a task to a step in the **Task Flow Editor,** the step parameters like Name, (default) Event or Start Window are set correctly by IDATG. For changing the entries, use the **Step Editor**.

The **Step Editor** gives you the possibility to specify the semantics of your GUI. Most importantly, you can define the trigger event for a step and define the conditions and actions associated with it. It can be used for all types of steps and is reached in the following ways:

- Double click on a transition (blue arrow) in the **Window-** or **Menu Editor**
- Double click on a task step in the **Task Flow Editor**
- Double click on a test step in the **Test Case Editor**

Depending on the type of step, some buttons or fields may not always be available. For composite task steps (yellow), a special **Sub Task Editor** is used.

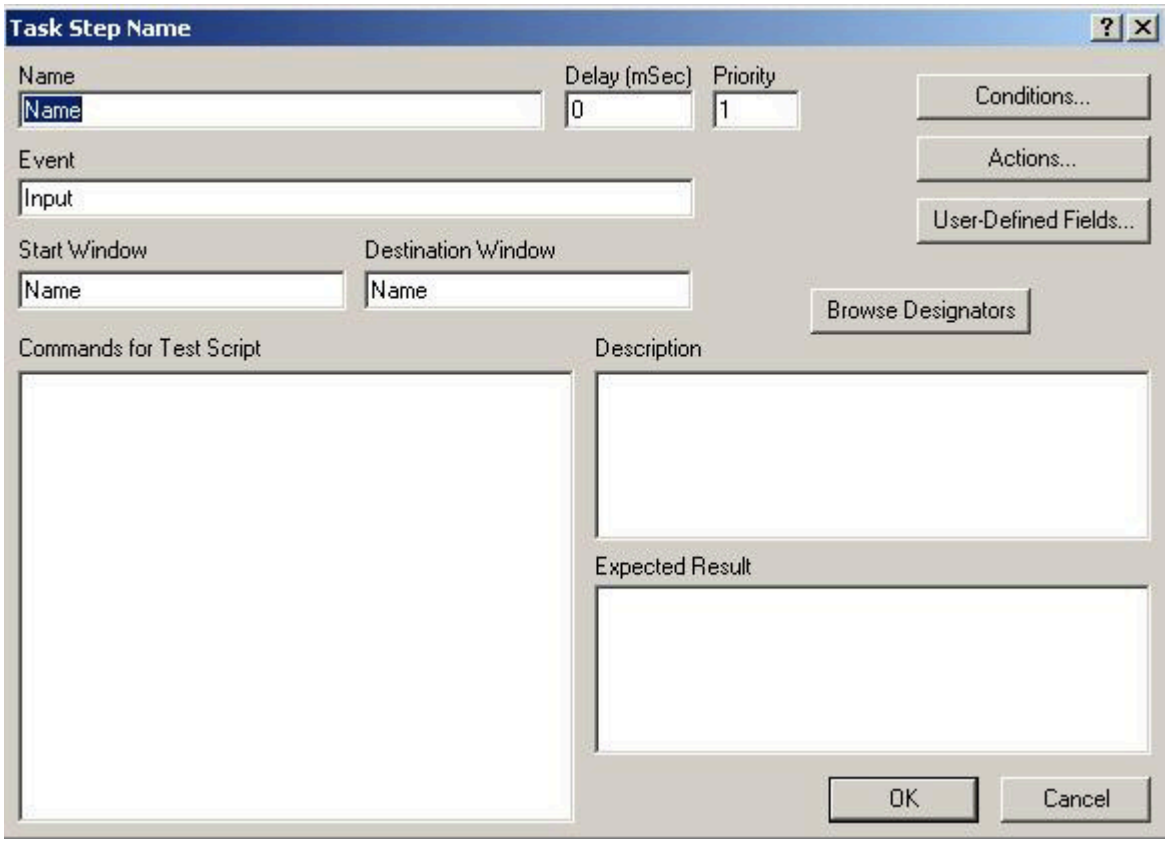


Figure 8 - Step Editor

Name

The name must be unique, may only contain alphanumeric characters and the character '_', and it can be up to 50 characters long.

Event

The event applies to the 'Start Window'. The two most important events are **Click** and **Input**:

- The function *Click* represents a mouse click on a window. The click can be executed with either the left (L) or right (R) mouse button and may be single (1) or double (2). The default is a single click on the left button.

  Examples: Click, Click(R), Click(L, 2),

- *Input* signifies text input into a field or table cell. You can either provide a specific text or call *Input* without parameters to express that an arbitrary value that matches the field's syntax is entered.

  Examples: Input, Input("MyText").

Exercise 5:

Define the three steps of the task 'EnterData': Enter a name and a password and press OK.

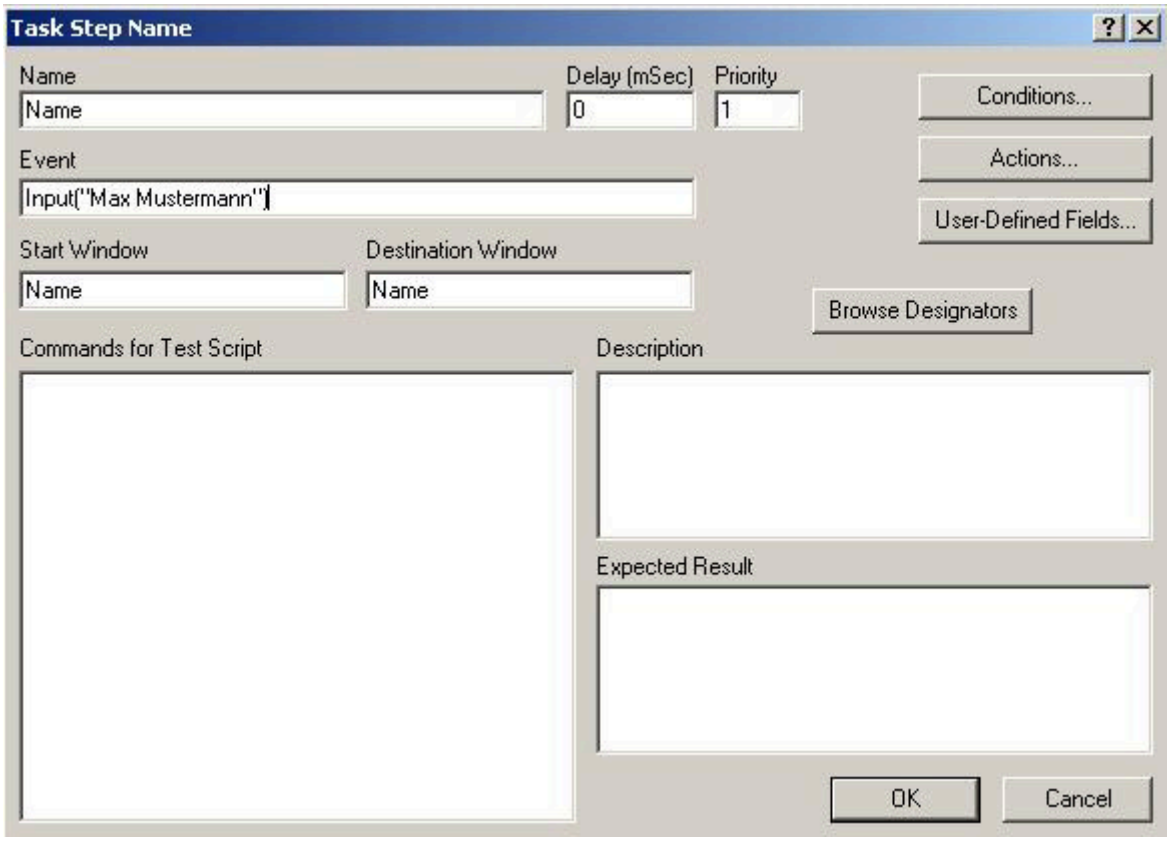
☑ Solution 5:

1. Open the **Task Flow Editor** of 'EnterData' by double-clicking it or clicking  .

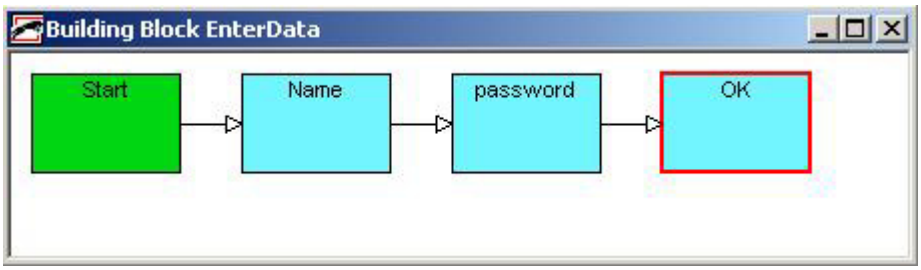
2. Create three new steps by clicking  , (or if you prefer,
   Click right on 'Start' and select 'Insert Step'
   Click right on 'New Step' and select 'Insert Step'
   Click right on the last step or on a connection between the steps and select 'Insert Step')

3. Select the first step, then click  (assign object to step), open the screenshot of the window 'Login', and click on 'Name'.

4. Click  (edit properties) to open the Step Editor, and enter a value for Name (see below)



5. Click "OK" to close the Step Editor.
6. In the same way, specify the steps for Password and OK.
7. Now your Task Flow should look like the screenshot below, and 'EnterData' in the Application Editor is displayed blue.



4.8 Task Parameters Editor

A building block can be compared to a function in a programming language. After defining the function prototype including the number and type of its parameters, the function can be called with different parameter values that affect the function's result. In the same way, the **Task Parameters Editor** can be used to define a "task prototype". Each time the building block is used as a sub task, different values may be passed to the parameters. See also section 4.10 about the **Sub Task Editor**.

To open the Task Parameters Editor, press ‘Parameters…’ in the Task Editor of a building block.

Creating a new Parameter

- Press ‘**New**’
- Enter name, type and default value. The name may only contain alphanumeric characters and ‘_’ and may be up to 30 characters long. The syntax for the default value depends on the selected type and is the same as for attributes. Please do not quote strings with "" unless the quotes are part of the value.
- Press ‘**Set**’. It is checked whether the name is unique and the default value corresponds to the selected type. If not, a message box appears and the focus moves to the invalid field.

Selecting a Parameter

Click on a parameter name in the list

Editing a Parameter

- Select a parameter in the list. Its properties are shown in the corresponding fields and can now be edited.
- Edit name, type and default value.
- Press ‘**Set**’. It is checked whether the name is unique and the default value corresponds to the selected type. If not, a message box appears and the focus moves to the invalid field.

Deleting a Parameter

- Select a parameter in the list.
- Press ‘**Delete**’. A message box appears that has to be confirmed by the user. (”Do you really want to delete this parameter?“)

Exercise 6:

Create parameters for the task ‘EnterData’ according to the picture below.

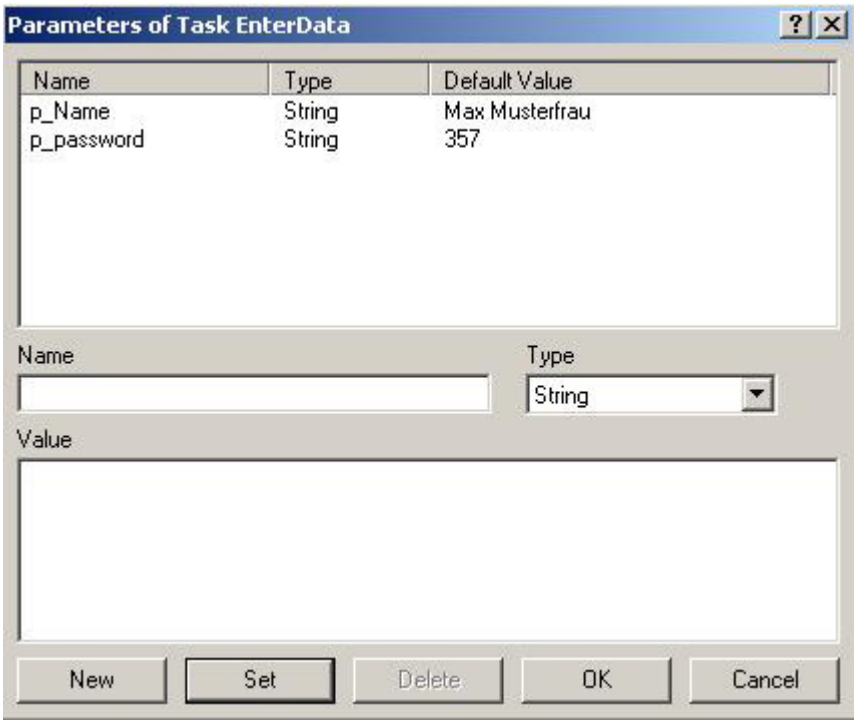


Figure 9 - Task Parameters Editor

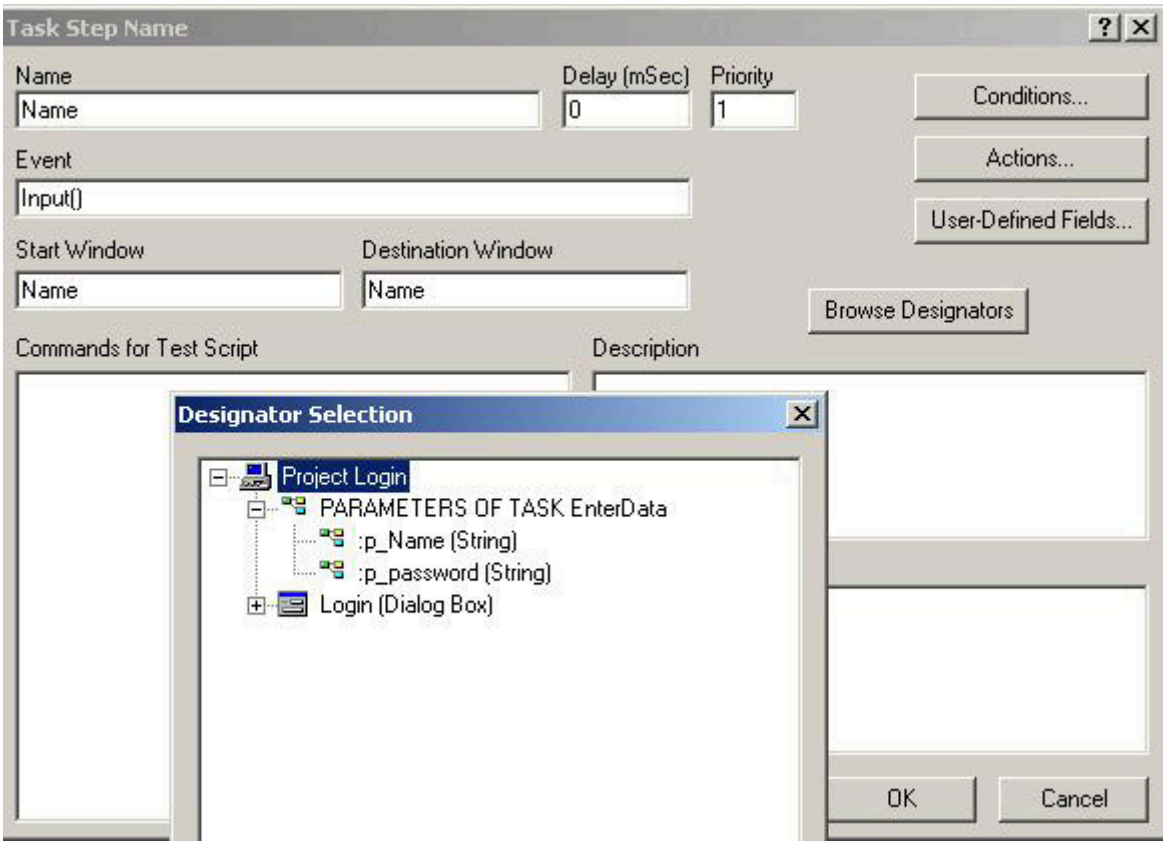
Solution 6:

1. Select the task ‘EnterData’, open its Properties Editor, and click the button ‘Parameters…’
2. Enter “p_Name” in the Name edit field, and “Max Musterfrau” as value
3. Click ‘Set’
4. Enter “p_password” in the Name edit field, and “357” as value.
5. Click ‘Set’
6. Click ‘OK’ to close the Parameters Editor, and ‘OK’ in the Task Properties Editor.

4.9 Replacing Static Values with Parameters

Edit each step where you want to use a parameter, replacing your static values with parameters. The syntax is *#@TaskName:Parameter#*.

- Open the Step Editor (double-click the step, press  (edit properties) or select the menu ‘Window|Properties Editor)
- Delete your argument in the event edit field (in case there is one already)
- Position the cursor in the event edit field
- Click ‘Browse Designators’ and choose your parameter (double click it)



- Make sure that the parentheses are set correctly around the argument of your event (the parameter)

4.10 Sub Task Editor

The **Sub Task Editor** is used for **task steps** that represent a building block (yellow steps). The name of such a step is always the name of the block and cannot be edited. This editor serves to define the values of the sub task's parameters.

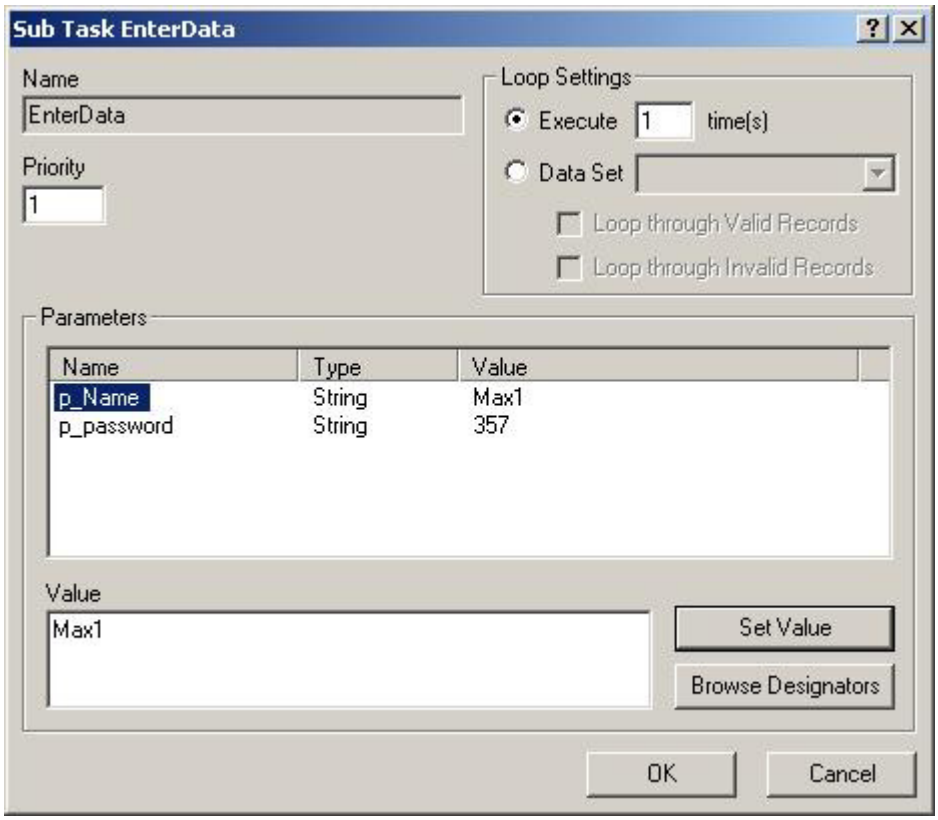


Figure 10 - Sub Task Editor

Editing Parameter Values

If you have defined parameters for the sub task using the **Task Parameters Editor,** you can enter their values here. Using sub tasks can be compared to calling a function in a programming language. You can pass different parameter values to the sub task every time you use it in a task flow.

- Select a parameter in the list
- Enter the new value. The value can be an arbitrary expression and must have the correct type. If you do not enter a value, the default value is used.
- Press '**Set**'. The value is checked for syntactic correctness.
- If more parameters are available, the next one is selected automatically.

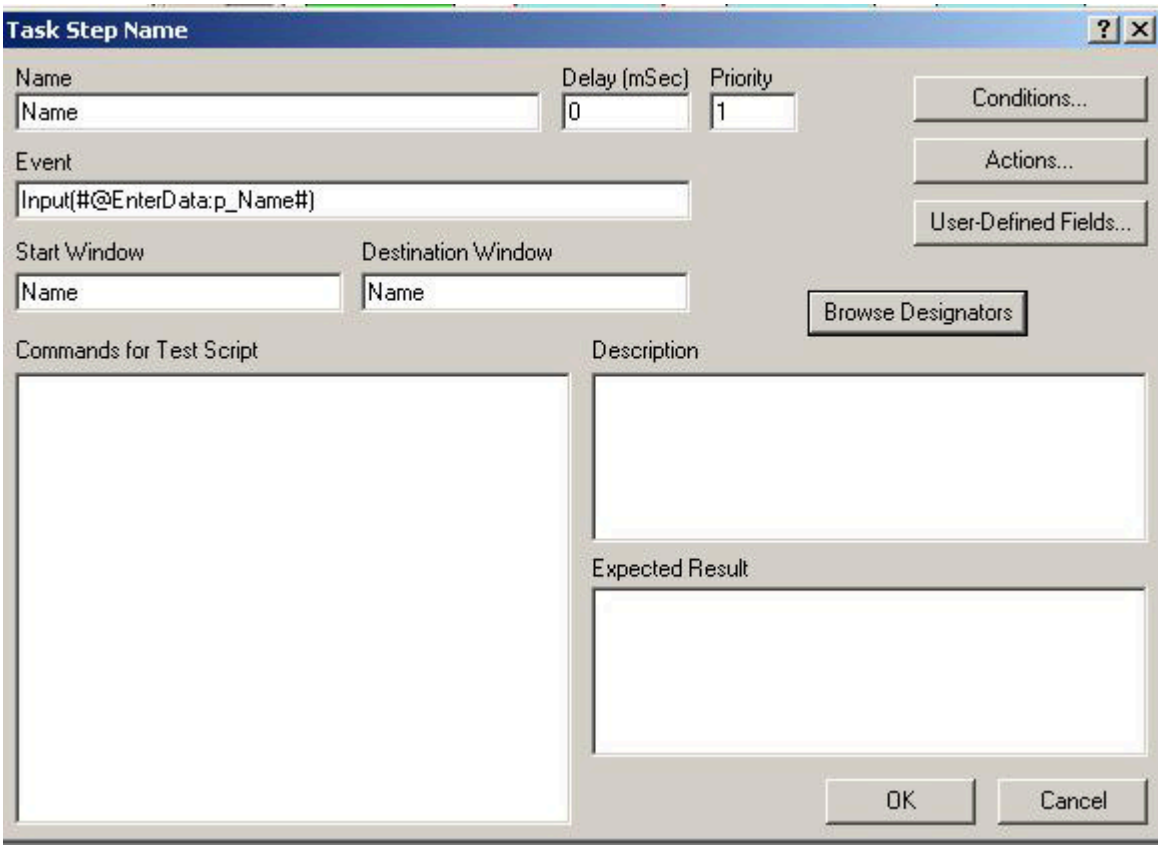
Exercise 7:

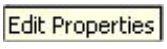
Change the task 'EnterData' so that Name and Password are set using parameters. Create a test case 'TC_EnterData' and add two parallel steps to it. Assign them both to the task 'EnterData'. Now define the parameter values using the default values for the first one and empty (invalid) strings for the second one.
During test execution, an error message box will appear in the second branch, so use the GUI Spy for capturing the box and add a step for quitting it.

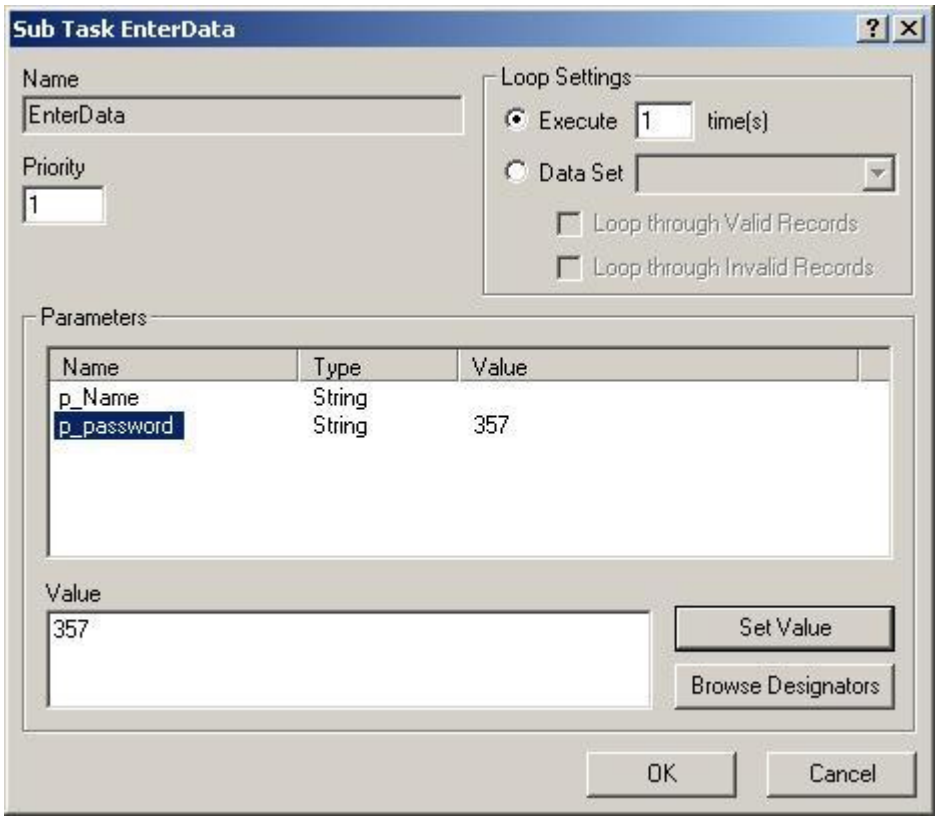
Solution 7:

1. Open the project 'Login'
2. Open the **Task Flow Editor** of 'EnterData'
3. Select the step 'Name' and press the button  Edit Properties or double-click on the Step

4. Set the Event to **Input(#@EnterData:p_Name#)** or press the button *'Browse Designators'* and select the parameter 'p_Name' (Project Login | PARAMETERS OF TASK EnterData | p_Name) and double-click it.



5. Click 'OK'
6. For the step Password, set the event to **Input(#@EnterData:p_password#)**
7. Create a use case TC_EnterData: Select 'Project Login' and press  . Rename the NewTask1 to 'TC_EnterData'.
8. Add the task 'EnterData' twice to the 'TC_EnterData'
   - open the task flow editor of 'TC_EnterData' by double-clicking it
   - add a new step (right click on task step 'Start', select 'Insert Step')
   - assign the task 'EnterData' to this step
   - add a new step (right click on task step 'Start', select 'Insert Step')
   - assign the task 'EnterData' to this step
9. Select the second 'EnterData' sub task (composite step) and press the button   or press the right mouse button and click 'Edit Step'
10. Now the **Sub Task Editor** opens



11. Set the values as shown on the screenshot value= “” (empty)

12. To capture the message box, start ‘Login.exe’ and click ‘OK’ without entering a name. Capture the window ‘Login Error’ (see chapter 4.2).
    - Right-click on the second ‘EnterData’ task, select ‘Insert Step’
    - Click  (Assign Object to Step)
    - Select the Windows tab, click on ‘Application Login|LoginError|OK1’
    - As ‘OK1’ is not a mnemonic step name, change it to ‘OK_ERROR’ (edit step).

Your result should look like this:

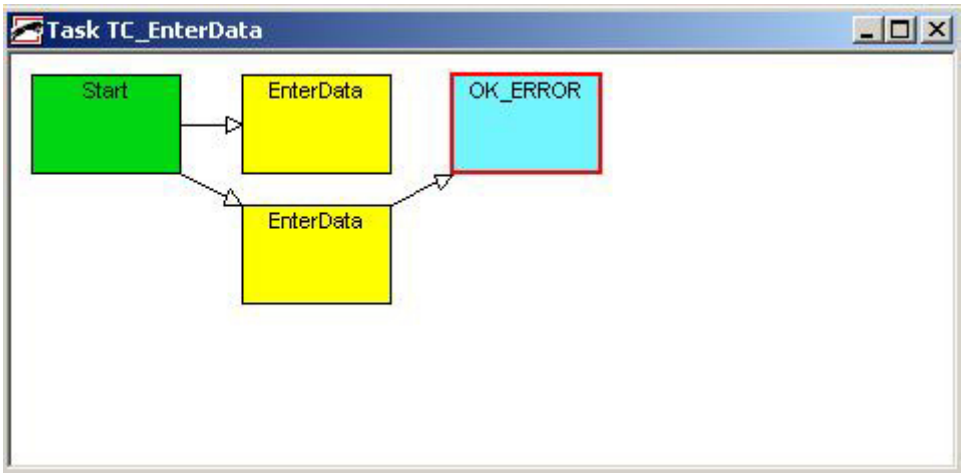


Figure 11 – Task Flow of TC_EnterData

In the first test case, where you enter correct data, you receive a message that the password is okay.

Exercise 8:

Add the window ‘Login Message’ to your GUI map, and add a step to confirm the message.

Solution 8:

- Start ‘login.exe’
- enter a name and any password, cklick ‘OK’
- record the message window ‘Login Message’ by clicking  in IDATG and dragging  over the message window. Click ’OK’ in the GUI Spy window.
- In the Task Flow of Task ‘TC_EnterData’ select the yellow subtask ‘EnterData’ which sets correct data. Click  and then  .
- Select ‘LoginMessage | OK2’ in the Windows pane of the application editor.
- double-click the blue step OK2 and change its name to ‘OK_Password’ (not necessary, but recommended)

4.11 Generating Task-based Test Cases

From the task model the **Test Case Generator** generates test cases that cover all functions defined in the task model and are suited for system testing.

You can decide whether the generated test cases should only cover all task steps or also all connections between them thus achieving an even better test coverage.

- Select the desired task and press the button  Generate Test Cases.

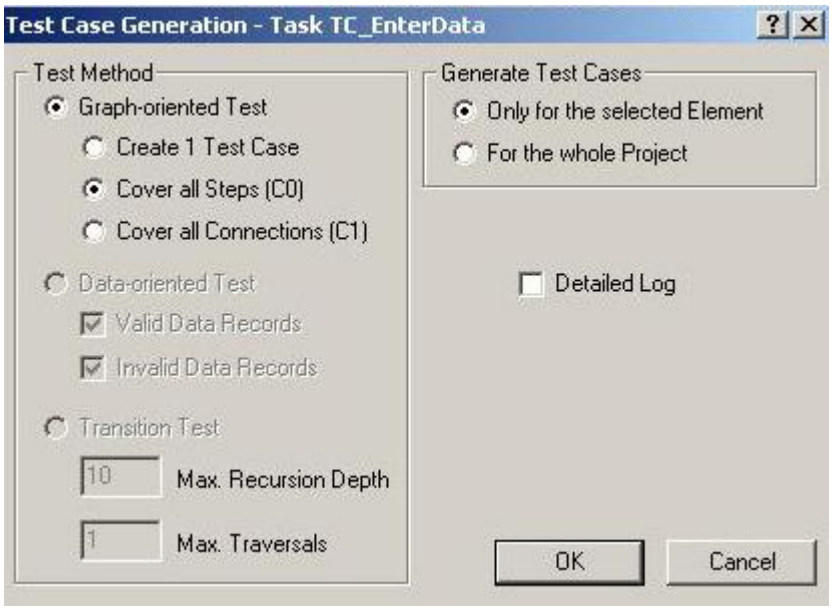


Figure 12 – Test Case Generation

- Decide if you want to generate test cases only for the selected element or for the whole application (in our example the result is the same).
- Press OK
- Confirm the success message

4.12 Converting Test Cases

- Select the desired task and call the Test Case Converter  .
- Supply the command line to call the application under test.
- Select the Output Format according to your tool from the droplist
- Press 'OK'
- Confirm the success message

For WinRunner, the following files are generated:
- A GUI Map containing a description of all GUI objects
- Test scripts containing the test cases in the tool-specific language
- A main script that calls all other scripts

With these test scripts, you can start the test with your capture-replay tool.

If you have not generated the test scripts yet, do it now:

 **Exercise 9:**

1. Generate test cases for 'TC_EnterData'.
2. Convert the test cases and run the test.

 **Solution 9:**

- Select 'TC_EnterData'
- Press  (Generate Test Cases) or right-click 'TC_EnterData' and select 'Generate Test Cases' from the context menu or choose menu 'Generate | Generate Test Cases'.
- Confirm the window 'Confirm Changes' by pressing 'OK'.
- Confirm the 'Test Case Generation' Window by pressing 'OK'
- Confirm the window 'Test Case Generation Status' by pressing 'OK'
- Press 
- Confirm the window 'Confirm Changes'
- Provide the command line for the application 'Login', *Path*\Login.exe
- Select the Output Format
- Click 'OK'
- Confirm the conversion message '2 Test Case(s) converted!'
- If available, start your GUI testing tool and execute the generated script

Your first project 'Login' is finished now.

4.13 Summary

In this chapter you have learned to record a GUI, define a task model and generate task-based test cases. After converting the test cases, you can run them with your capture/replay tool.

5 Project Tennis

The application 'Tennis' serves to administer tennis tournaments. Before continuing, please start `tennis.exe` and make yourself acquainted with it. Note that the program contains a number of intentional bugs.

All exercises assume that IDATG and `tennis.exe` are started and that you have accomplished all former exercises.

You will have to implement the following test cases:

Regular test cases
TC_R1: Create a new tournament
TC_R2: Create several players
TC_R3: Assign players to a tournament
….

Error test cases
TC_E1: Wrong tournament dates (end before beginning)
TC_E2: Assign a female player to a men's single tournament
…..

5.1 Recording the GUI

Start modeling your use cases by recording the GUI of your AUT (application under test). For your convenience you should check immediately whether all windows have names that are easy to remember. If necessary, change them.

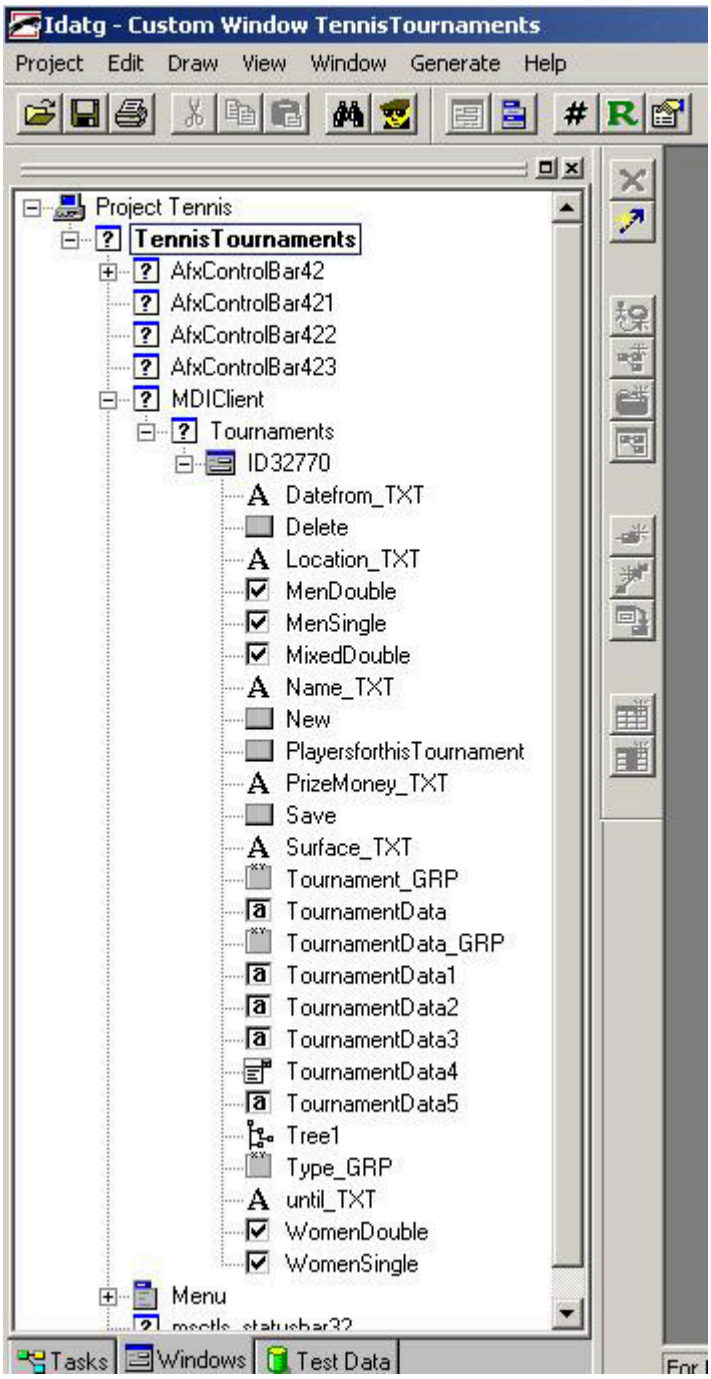
Exercise 10:

- Create a new project 'Tennis'
- Record the start window

Solution 10:

- Select menu 'Project >> New …' or press 'Ctrl+N'
- In the Project Administration window enter 'Tennis' as name, and leave Default GUI builder and Default Output Format as they are. Press 'OK'.
- Start tennis.exe
- Invoke the GUI Spy by clicking on its icon 
- Drag the crosshairs over the window 'Tennis – Tournaments' until it is highlighted by a red frame
- Click 'OK' in the 'GUI Spy'

First of all, let's define the newly recorded window as start window. Just click right on 'TennisTournaments' and select 'Set as Start Window'. Now expand the tree nodes 'Project Tennis', 'Tennis Tournaments', 'MDIClient', 'Tournaments' and 'ID 32770'. This is what you (should) see:



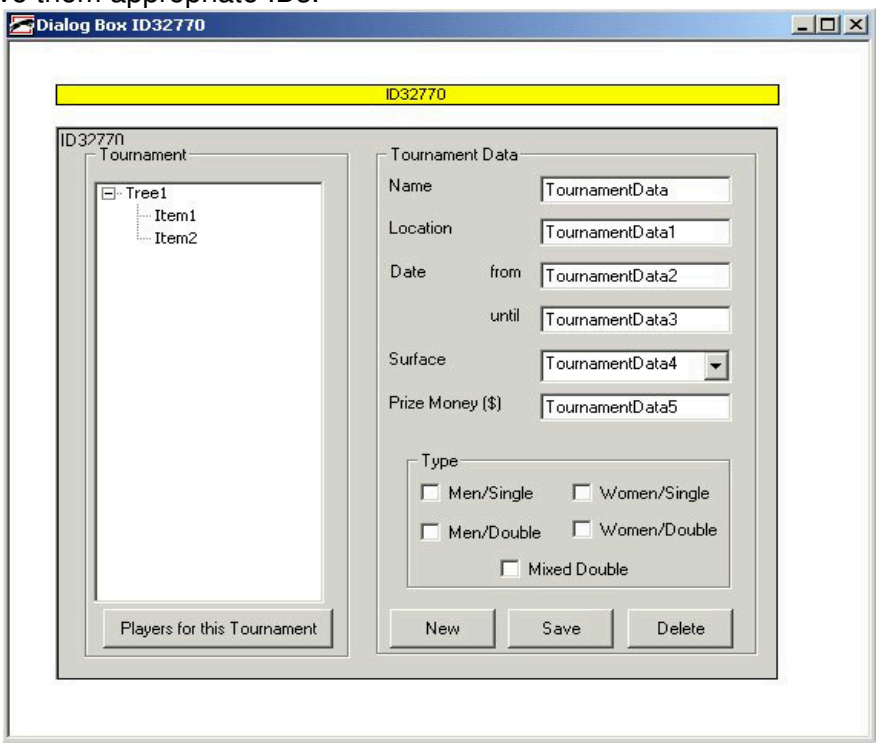
There are several windows starting with 'AfxControlBar..'. Most likely you will never need them in your tests, so delete them to increase the overview (if you really need them later on, it is still possible to add them again, either by recording or by drawing).

If a window has no caption, the GUI Spy uses the class name as window ID. For your own convenience, you should rename such windows using meaningful IDs. For example: The window 'tree1' can be renamed to 'TournamentCalendar' using the **Window Properties Editor**. Other candidates are the fields 'TournamentData..'. To see what they are, open the **Window Editor** of their parent:

- right click on 'ID 32770' and select 'Edit Window' or
- double click 'ID 32770', or
- select 'ID 32770' and choose menu 'Window >> Window Editor', or

- select ‘ID32770’ and click the icon ‘Edit Window’).

Now you can give them appropriate IDs.

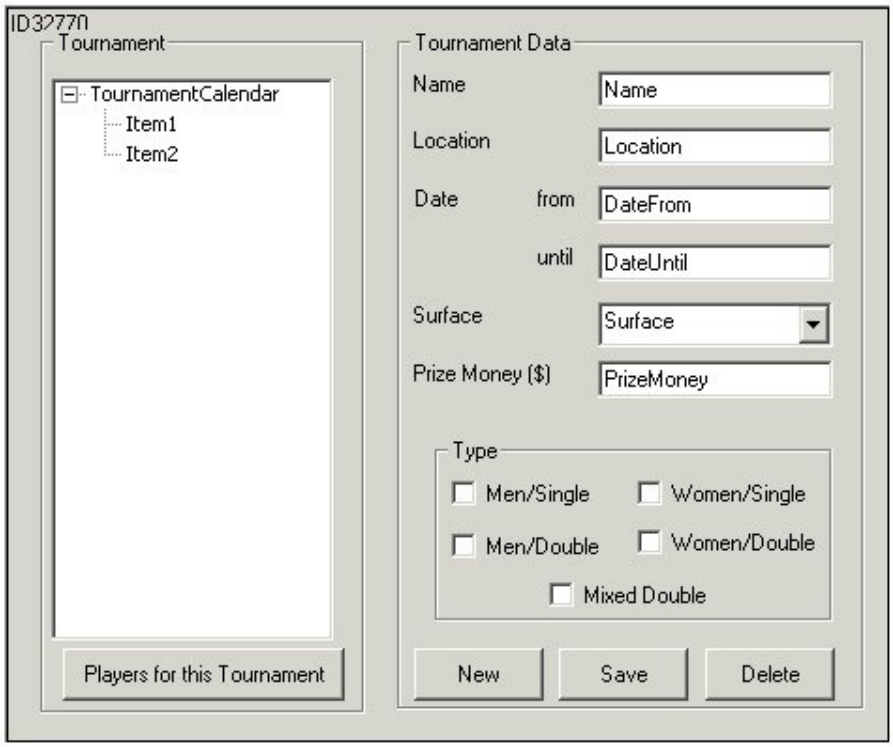


Window Editor of ‘ID 32770’ before changing window IDs

Your possibilities to edit the ID are:

- like in the Windows Explorer, select the item in the windows pane and click it again
- right click the item (in the windows pane or the windows editor) and choose ‘Edit Properties’ from the context menu
- select the item and click the toolbar button ‘Edit Properties’
- select the item and choose the menu ‘Window >> Properties Editor’

The name must be unique, may only contain alphanumeric characters and the character ‘_’, and it can be up to 50 characters long.



Window Editor of 'ID 32770' after editing the window names

Exercise 11:

Change the window IDs according to the above picture and set 'Tennis Tournaments' as start window.

Solution 11:

Choose one of the methods above and edit the window IDs.

5.2 Searching for a window

Now what about the 'msctl_statusbar32' (in the Application Editor, Windows pane)? What happens if you try to change the ID to 'Statusbar'?
An error message pops up, telling you that a window with ID Statusbar already exists. To find out where this window is, use the search function. IDATG provides a context-sensitive search function.

If the window pane is open, the menu 'Edit >> Search Object' or clicking on will open the dialog 'Search Window'. (In case you are in the task pane, it will open 'Search Task'.) When a window is selected, you are prompted to search for a window that is a child of the selected one, otherwise it is searched in the whole application.

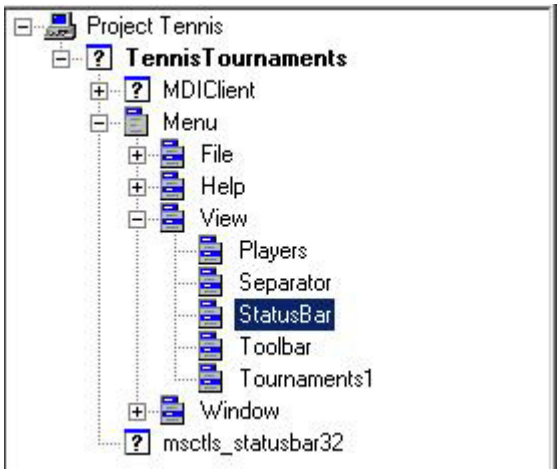
Exercise 12:

Find the window 'Statusbar'

Solution 12:

- As you have no expectation yet where this window belongs, select the entire 'Project Tennis' and open the search window by one of the above mentioned methods.

- Type 'Statusbar' into 'Window ID' and click 'OK'. Note that any combination of search criteria is possible, as well as entering only the beginning of the ID or caption (e.g. 'Stat').
- The tree 'ApplicationTennis' is expanded and the window 'StatusBar' is selected. Now we see that 'StatusBar' is actually a menu item.



Please note that the window IDs are not case sensitive.

Now you can make a qualified decision about the IDs you want to use. In our example we leave the msctls_statusbar32 as it is.

5.3 Recording Tab Pages

To continue our example:

- In the Tennis application, open the window 'Players' by selecting the menu 'View >> Players'.
- Record the window (only the child window; the main window is recorded already)
- Click 'New Player'
- Record the window 'Player Data'
- Expand all nodes of the window 'PlayerData' in the Windows pane of the Application Editor

Note: Idatg always puts newly recorded windows on the highest hierarchical level. If your replay tool is WinRunner, this is not a problem, since WinRunner knows only two hierarchical levels in the GUI-map (windows and objects).
However, if your replay tool is SilkTest, you have to drag the windows under their actual parent (TennisTournaments >> MDIClient for “Players”, TennisTournaments for “PlayerData”) because SilkTest needs the hierarchy information for identification. You can find out where a window belongs by checking the include-files of SilkTest.

The three tab pages of ‘PlayerData’ are shown as independent dialog boxes, and there is also a SysTabControl32. The relation between these items cannot be recorded, therefore it has to be edited manually.:

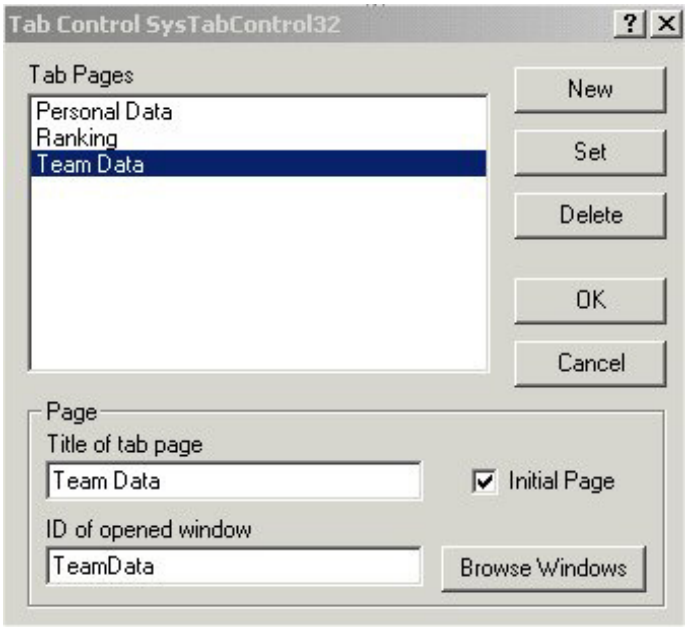
- Open the Properties Editor for *SysTabControl32*
- Click the button ‘Tab Pages…’
- Enter the caption of the first tab page (Team Data), exactly as in the application, into ‘Title of tab page’
- Set ‘ID of opened window’ by browsing for the corresponding window (TeamData)
- Check ‘Initial Page’ to specify that this is the first of the three tab pages
- Click ‘Set’ to save this information
- Click ‘OK’ twice to close the Tab Editor and Properties Editor

Exercise 13:

Enter the three tab pages of the window ‘PlayerData’

Solution 13:

- Open the Properties Editor for SysTabControl32
- Click the button ‘Tab Pages…’
- Enter the caption of the first tab page (Team Data), exactly as in the application, into ‘Title of tab page’
- Set ‘ID of opened window’ by browsing for the corresponding window (TeamData)
- Check ‘Initial Page’ to specify that this is the first of the three tab pages
- Click ‘Set’ to save this information
- Enter the caption of the next tab page (Personal Data), exactly as in the application, into ‘Title of tab page’
- Set ‘ID of opened window’ by browsing for the corresponding window (PersonalData)
- Click ‘Set’ to save this information
- Enter the caption of the last tab page (Ranking), exactly as in the application, into ‘Title of tab page’
- Set ‘ID of opened window’ by browsing for the corresponding window (Ranking)
- Click ‘Set’ to save this information
- Click ‘OK’
- Click ‘OK’ to close the Properties Editor



5.4 Radio Button Groups

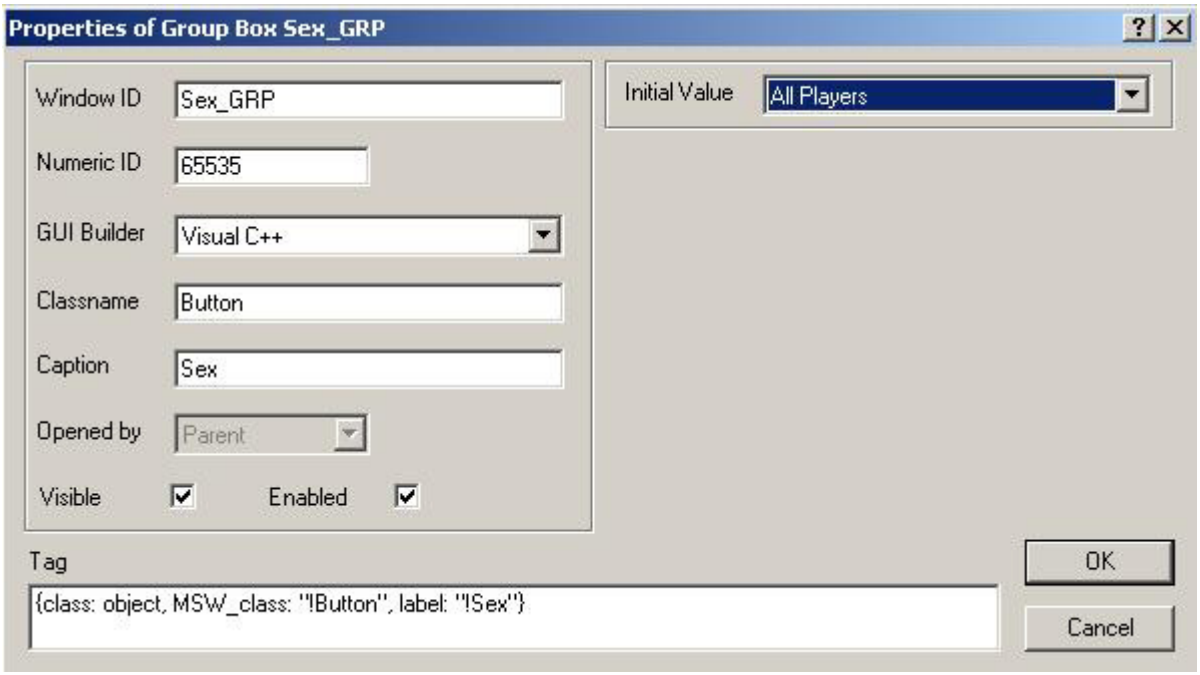
Radio Buttons never appear single, they always form a group that is usually surrounded by a group box (like in the windows 'Personal Data' and 'Players1'). Checking one radio button causes all other buttons in the group to become unchecked. Therefore the caption of the checked radio button in the group is seen as its value. The individual radio buttons do not have a value of their own. If you encounter a group of radio buttons that is not surrounded by a group box, please draw one in the Window Editor. IDATG needs this information to generate correct events.
Now look at the Group Box 'Sex' in 'Players1' (open the Properties Editor for Sex_GRP). You find a property 'Initial Value' and a drop list containing the buttons that belong to this group. Here you can set the default button according to your application.

Exercise 14:

Set the Initial Value of Sex_GRP to 'AllPlayers'.

Solution 14:

- Select "Sex_GRP" in the windows pane of the application editor
- open the properties editor
- choose "All Players" from the Droplist "Initial Value"
- press "OK"



As you have already defined your windows, you can go on to define your tasks.

5.5 Task flow

Exercise 15:

Create building blocks for the 3 regular test cases described at the beginning of chapter 5 (do not define the task flows yet). Combine these tasks into a logical sequence (new tournament, 2 new players, assign players to tournament) in the task flow of the “TC_R3_AssignPlayers”.

Solution 15:

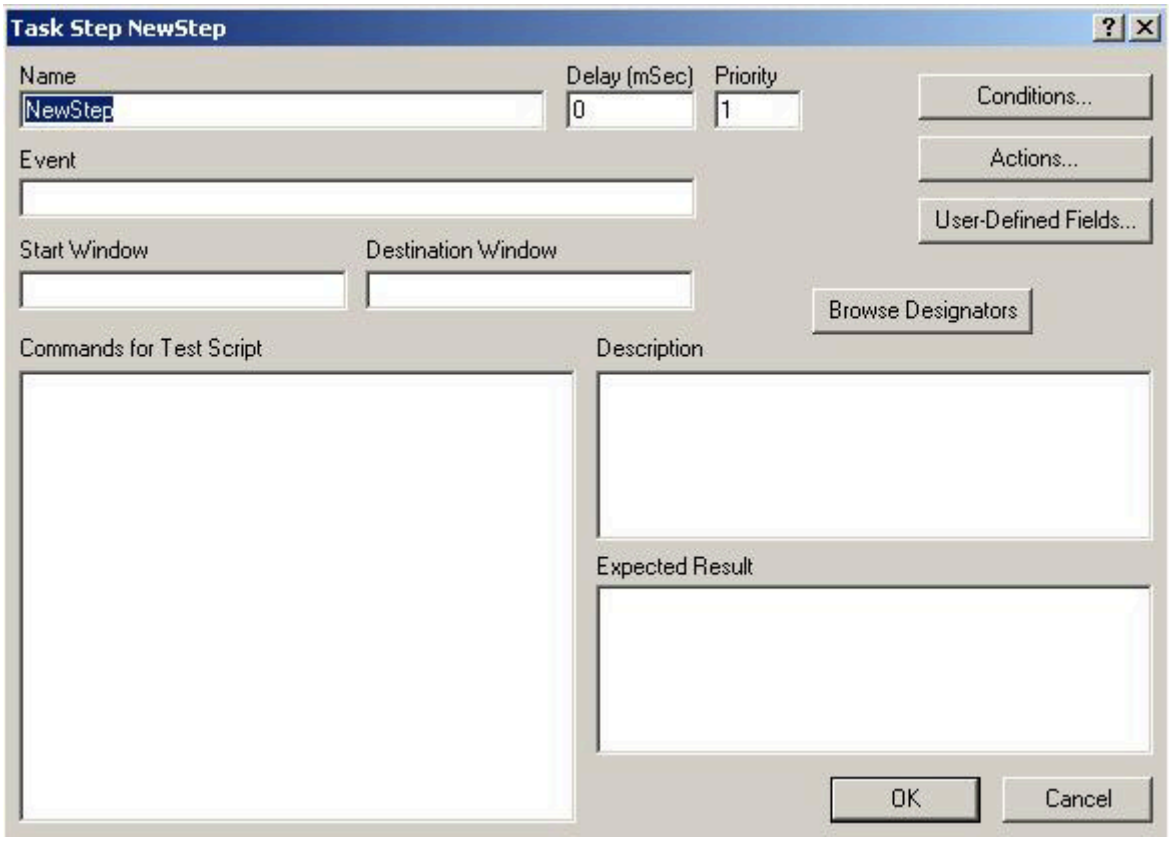
1. Click the “Tasks” tab in the Application Editor
2. Select “Project Tennis”
3. Create a new building block by clicking the toolbar button 
4. Rename the task to “NewTournament”
5. Repeat steps 1 to 4, renaming the tasks to “NewPlayers” and “AssignPlayers” respectively.
6. Create a new use case by selecting “Project Tennis” and clicking  , and rename it “TC_R3_AssignPlayers”.
7. Open the task flow of “TC_R3_AssignPlayers” by double-clicking it
8. Insert a new step and assign it to “NewTournament”
9. Add 2 instances of “NewPlayers” and 1 of “AssignPlayers” to the task flow of “TC_R3_AssignPlayers”.

In the tree the task names are written in black, and in the task flow the steps are light yellow, both indicating that the tasks steps are not defined yet.

Next you define the steps for these tasks:
Double click the task you want to define, NewTournament. The task flow editor opens and shows a start step (green), already selected (indicated by the red frame). Click right on the step and select “New Step” from the context menu. A new step is added to the task flow, connected to the step that was selected before. It is grey, indicating that it is not yet defined, and has a red frame indicating that it is selected.

5.5.1 Task Step Editing

You already learned to assign an object to a step (see Task Flow Editor and Step Editor) and to define a step this way. Of course you could enter all data manually, too, using the Step Editor. Open it by either double-clicking a step or by right clicking on a step and selecting “Edit Step” or by selecting a step and clicking  . This is what you get:



Type any name, enter an event, and enter the object (=window) on which to perform the event. Clicking the button 'Browse Designators' allows you to select the start or destination window from the windows pane.
A step is considered to be defined if it either has an event or 'Commands for Test Script' defined.

For the more convenient way close the Step Editor again (click "Cancel").

- Open the window on which you want to perform user actions in the Window Editor (here it is Application Tennis >> Tennis Tournaments >> MDIClient >> Tournaments >> ID32770, the immediate parent of all the edit fields we need to create a new tournament).
- Now right click on the step you want to define and select "Assign object to step" from the context menu (or select the step and click the toolbar button).
- Then click on the GUI element in the Window Editor, in our example "Name".

Now the step is shown in blue, which indicates that it is already defined. Open the step for editing by either right clicking on the step and selecting "Edit Step" or by selecting it and clicking . IDATG created the default event 'Input' without an input string. Enter one between brackets and quotes.

Exercise 16:

Create and define all steps necessary to define a tournament (NewTournament).

Solution 16:

- Open the window Application Tennis >> Tennis Tournaments >> MDIClient >> Tournaments >> ID 32770 by double-clicking it.
- Select 'NewTournament' in the task pane and click (Edit Task Flow).

- Click (New Step) and (Assign Object to Step).
- Click on the input field ‘Name’.
- Double-click the step ‘Name’ and enter the string (“Wimbledon”) after ‘Input’ in the event field.
- Click ‘OK’ in the Step Editor. The editor is closed, and the step is still selected.
- Click (New Step) and (Assign Object to Step).
- Click on the Input field ‘Location’.
- Double-click the step ‘Location’ and enter the string (“England”) after ‘Input’ in the event field. Click ‘OK’.
- Click (New Step) and (Assign Object to Step).
- Click on the Input field ‘DateFrom’.
- Double-click the step ‘DateFrom’ and enter the string (“2006-06-13”) after ‘Input’ in the event field. Click ‘OK’.
- Click (New Step) and (Assign Object to Step).
- Click on the Input field ‘DateUntil’.
- Double-click the step ‘DateUntil’ and enter the string (“2006-07-01”) after ‘Input’ in the event field. Click ‘OK’.
- Click (New Step) and (Assign Object to Step).
- Click on the combo box ‘Surface’.
- Double-click the step ‘Surface’ and enter the string (“Grass”) after ‘Select’ in the event field. It has to be spelled exactly as in the application. Click ‘OK’.
- The field ‘PrizeMoney’ is not obligatory, so we leave it blank.
- Click (New Step) and (Assign Object to Step).
- Click on the Check Box ‘MenSingle’.
- The default event is ‘Check(TRUE)’, so there is no need to edit the step.
- Click (New Step) and (Assign Object to Step).
- Click on the Push Button ‘Save’.
- The default event for Push Buttons is ‘Click’, so that’s okay.

A task flow like this is not suited to be used as a subtask because it assumes that the window ‘Tournaments’ is already open. We have to add another step to make sure it is opened.

Exercise 17:

Insert a step that makes sure the window ‘Tournaments’ is opened.

Solution 17:

- Select the connection between ‘Start’ and ‘Name’
- Click (New Step) and (Assign Object to Step).
- Select ‘Tournaments1’ from the windows pane of IDATG (Tennis Tournaments >> Menu >> View >> Tournaments1).

is the resulting task flow.

5.6 Conditions, Actions and Attributes

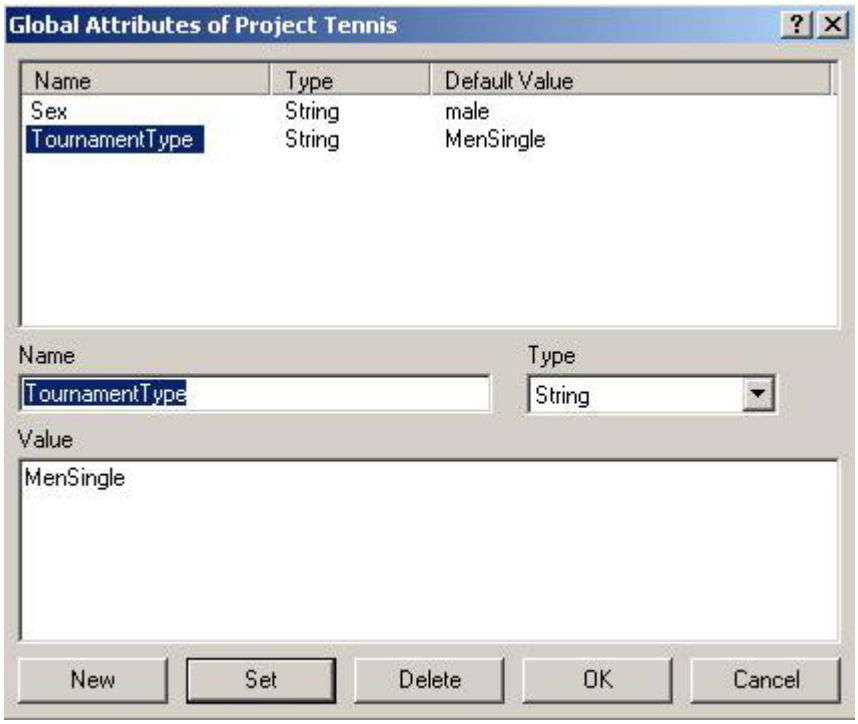
You may have wondered already about the purpose of the two buttons 'Conditions…' and 'Actions…' in the Step Editor. They are used for regulating the test case generation.

Imagine you want to make sure that only male players can be assigned to men's tournaments and only females to ladies' tournaments. One possibility would be to design a test case for 'Men', and a parallel one for 'Women'. However, it is also possible to reach this goal with a single test case:
We just have to remember the sex of the selected player in a user-defined variable ("attribute") and to select the matching tournament accordingly.

5.6.1 Creating Attributes

- Open the Attributes Editor by selecting the menu item "Window >> Attributes Editor"
- Enter 'Sex' into 'Name' and 'male' into 'Start Value'.
- Click 'Set'.
- Now enter 'TournamentType' into 'Name' and 'MenSingle' into 'Start Value'.
- Click 'Set'.

This is what you get when you select the attribute 'TournamentType':



Click "OK" to close the Attributes Editor.

5.6.2 Setting Conditions

Now we use the attribute 'Sex' in some steps. First we make the tournament type depending on the 'Sex' attribute:

- Select the step 'MenSingle' and open its Step Editor
- Click 'Conditions…'
- Click 'Browse Designators'

- Expand '# Global Attributes' and select '# :Sex (String)'
- Click the 'OK' button
- Add the condition: '#:Sex#="male"'
- Click 'Set'
- Click 'OK' twice

As a result IDATG will **generate** this test case only when the attribute 'Sex' has the value 'male'. Attributes and Conditions have no influence on the **execution** of the test cases.

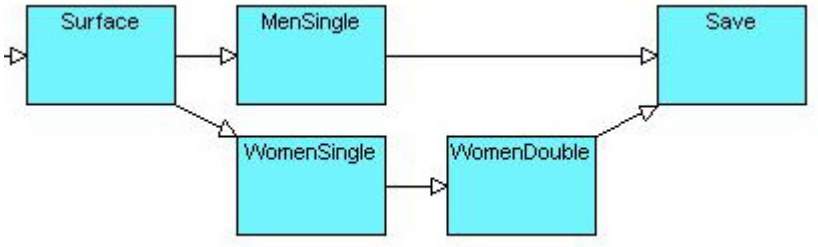
Now we create test steps for women's tournaments.

- Select the task step 'Surface' and insert a new step
- Assign the window 'WomenSingle' to this step
- Set the condition #:Sex# = "female"

Create another test step and assign 'WomenDouble' to it. You do not need to set the condition #:Sex# = "female" again, as this test step is generated only when the whole path can be generated, and the condition is already set.

Connect the step 'WomenDouble' with the step 'Save' (Use  or 'Draw >> Connect Steps' or right click and choose 'Connect Steps' from the context menu.).

The resulting task flow is:

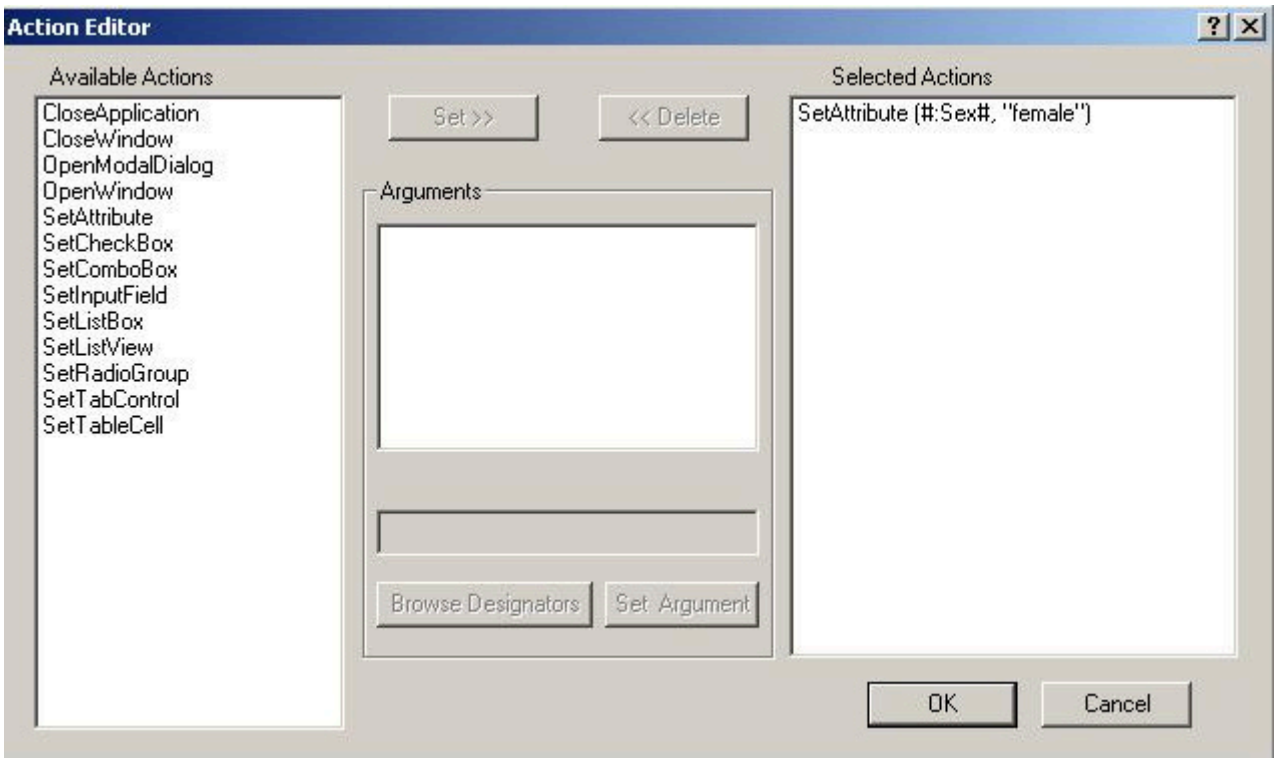


Now you can generate two different test cases, depending on the value you assign to the attribute 'Sex'.

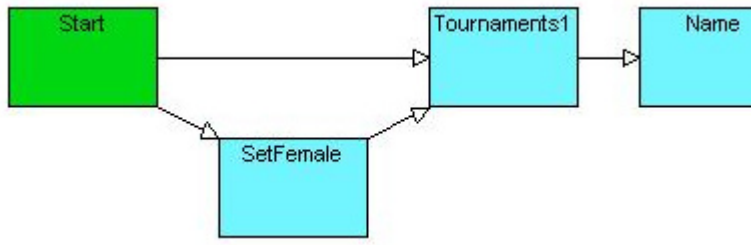
5.6.3 Setting Attribute Values with Actions

One way to set an attribute is to define an initial value, like we did above. Else you can define an action that sets the attribute.

- Select the start step of NewTournament and press 'New Step'.
- Open the Step Editor
- Click 'Actions…'
- Select 'SetAttribute' from the Available Actions in the Action Editor. You are prompted to enter the name of the attribute, or to browse for it.
- Click 'Browse Designators'
- Expand 'Global Attributes' and double click 'Sex'.
- Click 'Set Argument' in the Action Editor. The name-argument is set.
- Enter "female" for the Value-argument.
- Click 'Set Argument'. Both arguments of the SetAttribute function are defined.
- Click 'Set >>'. The action SetAttribute (#:Sex#, "female") is shown in the 'Selected Actions' pane.



- Click ‘OK’ in the Action Editor. You are back in the Task Step Editor.
- Enter ‘SetFemale’ as Name of the step.
- Click ‘OK’ in the Step Editor.
- Connect the step ‘SetFemale’ to the step ‘Tournaments1’



Now IDATG would generate 2 test cases, one for male and one for female. What about mixed tournaments? You need a third value, because ‘male AND female’ will never be true, and ‘male OR female’ is always true.

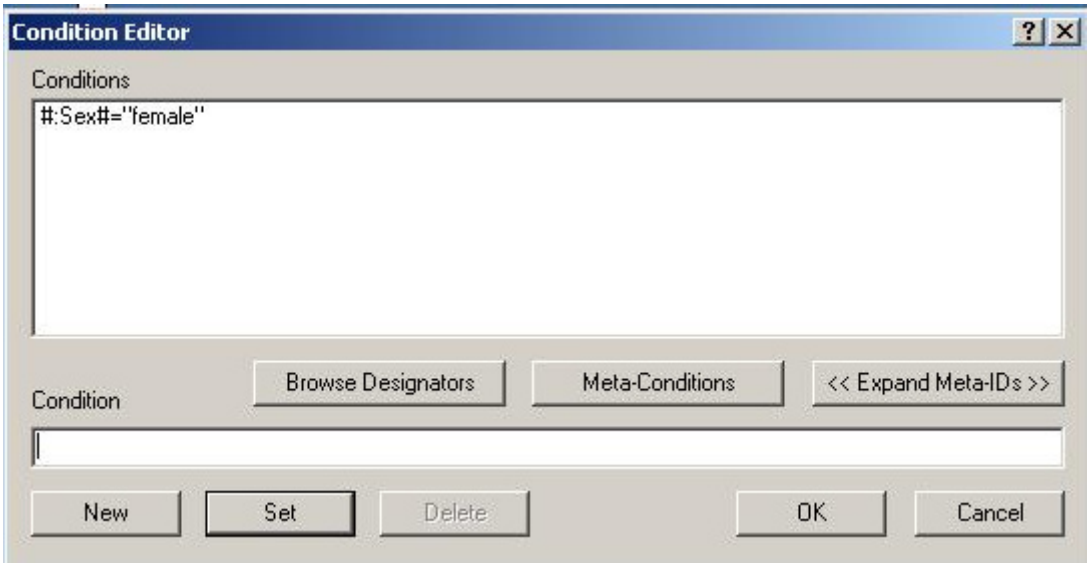
There is another place where the sex is relevant in this application: when creating and editing players.

Exercise 18:

Define the first part of the task flow of ‘NewPlayers’ that opens the new player dialog and sets the sex of the player according to the actual value of the attribute ‘Sex’.

☑ Solution 18:

- Double click ‘NewPlayers’ in the task pane to open the Task Flow Editor.
- Click (New Step) and (Assign Object to Step).
- Select ‘Tennis Tournaments’ >> ‘Menu’ >> ‘View’ >> ‘Players’ from the windows pane
- Click (New Step) and (Assign Object to Step).
- Select ‘Players1’ >> ‘ID327701’ >> ‘NewPlayer’ from the windows pane
- Open the Window Editor for ‘PlayerData’
- Select the last step of the task flow, click (New Step) and (Assign Object to Step)
- Select ‘PlayerData’ >> ‘Female’ by clicking on the radio button ‘Female’. Note that the step is assigned to the radio group Sex_GRP1 and not to the radio button itself (see 5.4).
- As ‘Female’ should be set only when the attribute ‘Sex’ has the value ‘female’, you have to define a condition. Open the Step Editor and click the button ‘Conditions…’
- Click ‘Browse Designators’
- Select ‘GLOBAL ATTRIBUTES >> Sex’ in the Designator Selection window and click ‘OK’.
- Edit the condition to make it ‘#:Sex#=”female” and click ‘Set’.



- Click ‘OK’
- Change the step name to ‘SexFemale’ and click ‘OK’

- Select the step ‘NewPlayer’ in the task flow, click (New Step) and (Assign Object to Step)
- Select ‘PlayerData’ >> ‘Male’ by clicking on the radio button ‘Male’.
- As ‘Male’ should be set only when the attribute ‘Sex’ has the value ‘male’, you have to define a condition. Open the Step Editor and click the button ‘Conditions…’
- Click ‘Browse Designators’
- Select ‘GLOBAL ATTRIBUTES >> Sex’ in the Designator Selection window and click ‘OK’.
- Edit the condition to make it ‘#:Sex#=”male” and click ‘Set’.
- Click ‘OK’
- Change the task name to ‘SexMale’ and click ‘OK’

The remaining input fields will be edited using parameters.

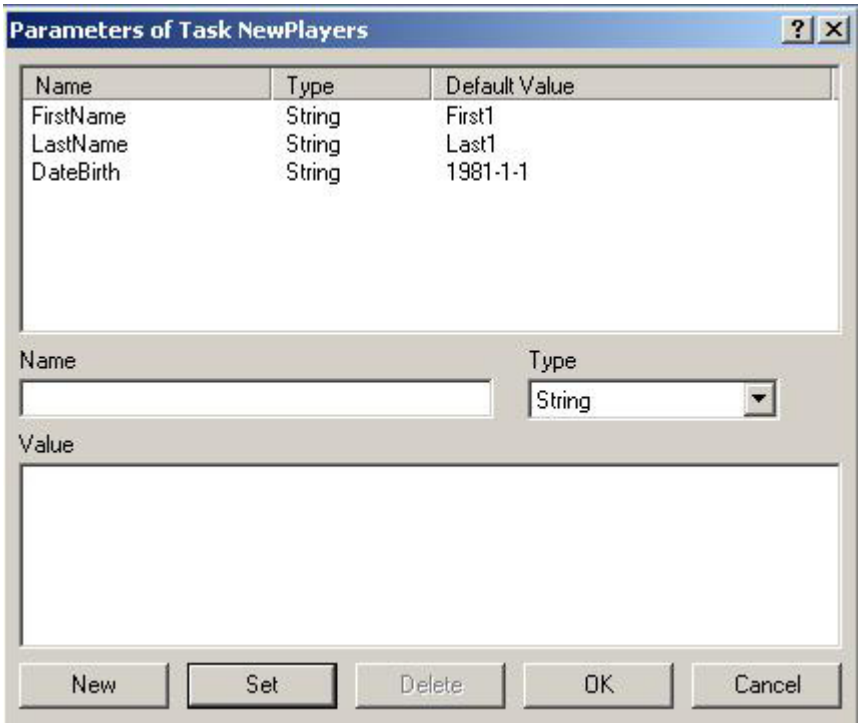
5.7 Parameters (again)

Tasks (building blocks) can have parameters that get specific values each time a task is used as a subtask in a task flow. This way one and the same task can be used for entering different data into edit fields.

5.7.1 Defining Parameters

As you already know from the Login example, there is an editor for parameters. You open it by right clicking a task in the task pane, selecting 'Edit Properties' and clicking 'Parameters…' in the Task Properties Editor. Do so for the task 'NewPlayers'. We will create parameters for all obligatory input fields.

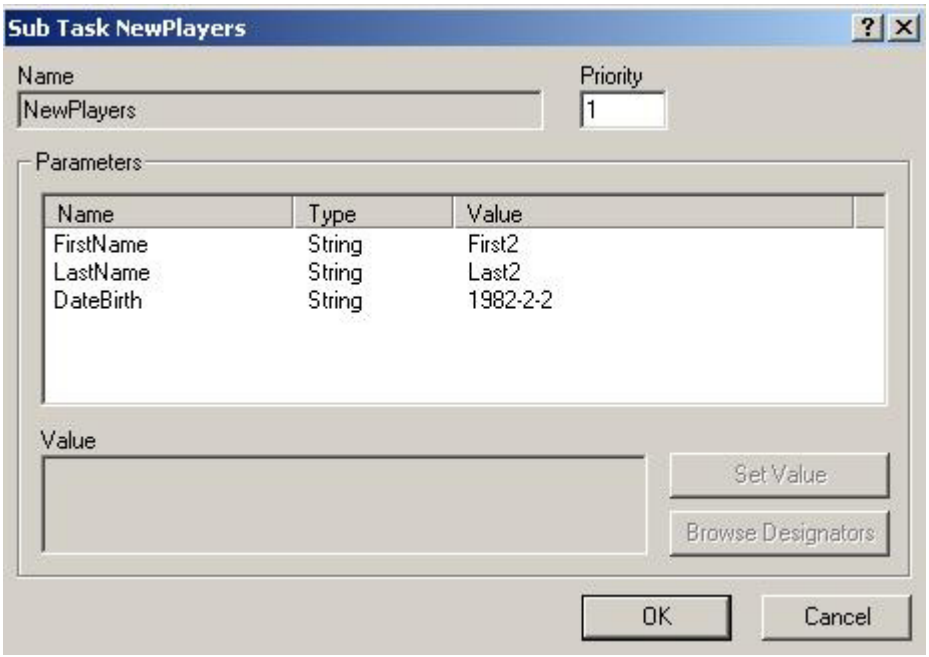
Enter 'FirstName' in the name field and 'First1' into the Value field. Click 'Set'. The parameter is displayed in the upper window. Our second parameter is called 'LastName' and gets the value 'Last1'. Next we define 'DateBirth', still of type 'String', with the value '1981-1-1'. If you want to edit a parameter, select it; now it is shown in the editable fields. For undoing changes, click 'New'. Click 'OK' twice to add your parameters to the task and to close the Task Properties Editor.



5.7.2 Using Parameters

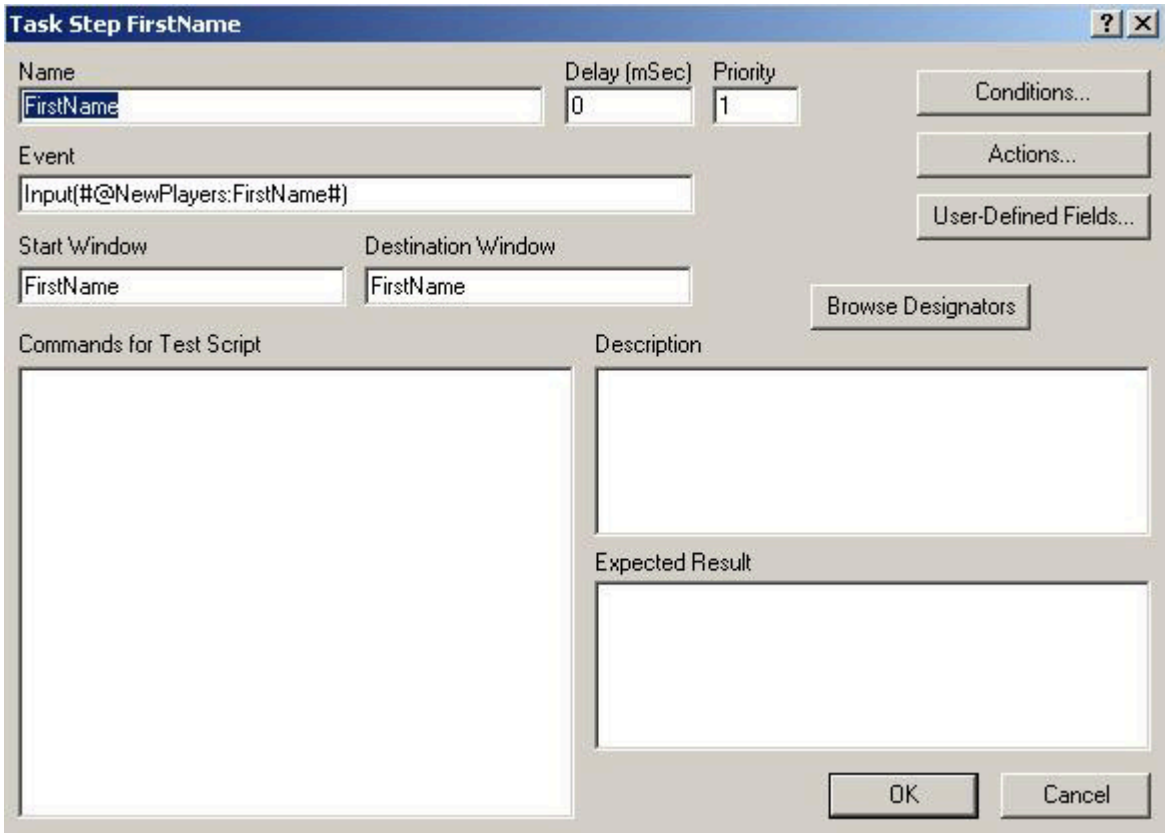
Whenever you use a task with parameters in a task flow, you can open the Sub Task Editor by right clicking on the sub task (yellow step) and selecting 'Edit Step'. All parameters defined are offered for setting their values.

Now open the Task Flow Editor for the 'TC_R3_AssingPlayers' (double click 'TC_R3_AssingPlayers' in the task pane). Right click on the first instance of 'NewPlayers' and select 'Edit Step'. The Sub Task Editor opens and the task parameters are offered for setting their values. Confirm the default values by clicking 'OK'.
Since a tennis tournament is not much fun without at least 2 players, we will now create a second player with a different name and date of birth. Right click on the second instance of 'NewPlayers' and select 'Edit Step'. This time change the values to 'First2', 'Last2' and '1982-2-2'. Edit the values and click 'Set Value'.



Click ‘OK’ to confirm the new values.

Until now the parameters are not used in any task steps. Open the task flow of ‘NewPlayers’ by double-clicking on one of its instances, insert a new step after ‘SexMale’ and assign it to ‘PlayerData >> FirstName’. Open the step for editing (right click, ‘Edit Step’), put the cursor in the Event field (it says ‘Input’ already) and click ‘Browse Designators’. The Designator Selection window shows the parameters of task NewPlayers. Double click ‘# :FirstName (String)’. Please correct the syntax by setting the brackets.

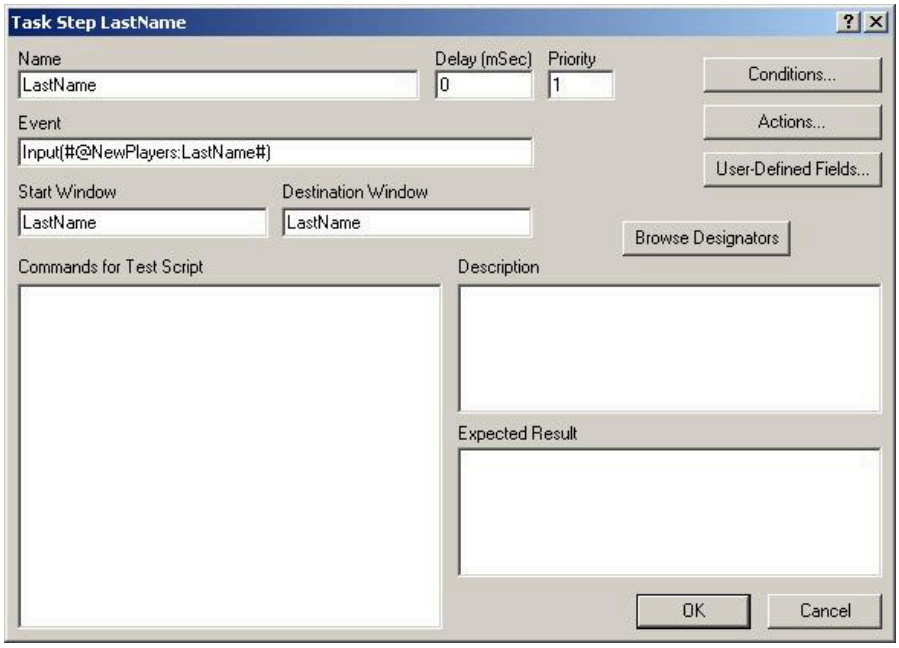


Click ‘OK’. Since entering a name is also required for female players, please create a connection between ‘SexFemale’ and ‘FirstName’.

Exercise 19:

Create a step to input ‘LastName’ using the corresponding parameter.

Solution 19:



5.7.3 Parameters in Embedded Tasks

Until now we have only entered the name and sex of the players. However, the window ‘Player Data’ also contains three tab pages for entering additional data. In order to achieve a clearer test structure, we will create separate tasks for the input into each tab page. It does not really matter where we put these tasks in the task tree because this decision does not influence the task flow. In our example, we put them into a folder we call ‘Player Input’.

- Select ‘Project Tennis’ in the task pane and click (New folder). Call it ‘Player Input’
- Right click ‘Player Input’ and select ‘New Building Block’. Name it ‘TeamData’ and click ‘OK’.
- Right click ‘Player Input’’ and select ‘New Building Block’. Name it ‘PersonalData’ and click ‘OK’.
- Right click ‘Player Input’’ and select ‘New Building Block’. Name it ‘Ranking’ and click ‘OK’.

Open the task flow of ‘NewPlayers’, create a new step at the end and assign it to the new task ‘TeamData’. Do the same for ‘PersonalData’ and ‘Ranking’.

In the first tab ‘Team Data’ the field ‘Nationality’ is obligatory. To be able to create a new player, we now define the task flow of task ‘TeamData’ using fixed values.

Double click ‘TeamData’ in your task tree. The name is still black, but a new Task Flow Editor is opened. Add a new step ( ) and assign the field ‘PlayerData >> PlayerData1 >> TeamData >>

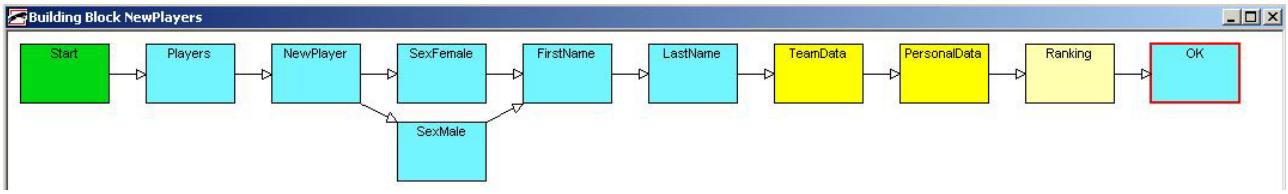
Nationality' to it. Now double-click the step and set the event to 'Input("Austria")'. Click 'OK'. Note that the task name in the tree turned blue.

Now we will define the task flow of the second tab 'Personal Data', but this time using a parameter. In this tab page we need to enter the date of birth. Open the Task Properties Editor for 'PersonalData' and click 'Parameters…'. Create a parameter 'DateBirth' of type string (you can leave the default value empty). Then click 'Set' and 'OK'. Click 'OK' in the Task Properties Editor as well.

Now open the task flow of 'NewPlayers', right click on the step 'PersonalData' and select 'Edit Step'. Remember that the task 'NewPlayers' also has a parameter 'DateBirth'. We have to pass the value of this parameter to the task 'PersonalData'. Use 'Browse Designators' to select #@ NewPlayers:DateBirth# and press 'Set Value' followed by 'OK'.

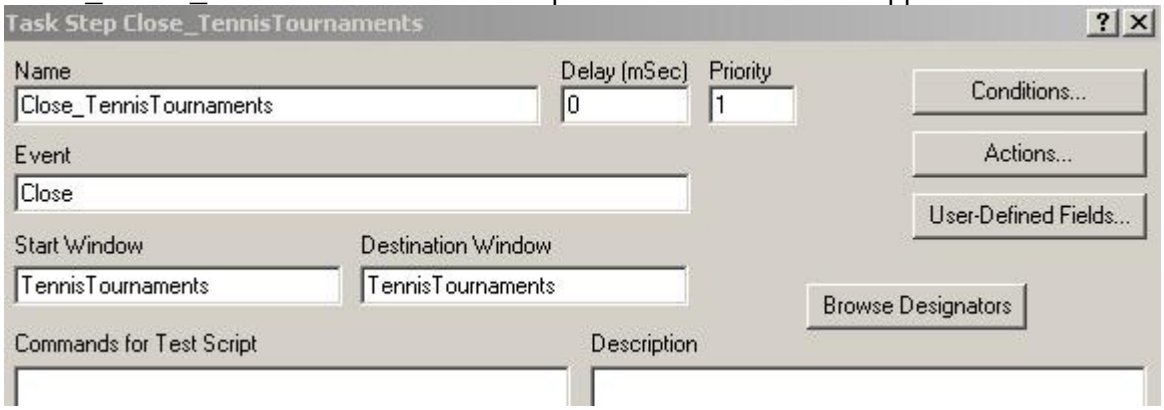
Now we have to define the task flow of 'PersonalData': First we have to add a step that changes the current tab page. This can be achieved by assigning it to the tab control 'PlayerData >> PlayerData1 >> SysTabControl32' and setting the event to 'Select ("Personal Data")'. Then we add a second step, assign it to the window 'PlayerData >> PlayerData1 >> PersonalData >> DateofBirth' and give it the event Input (#@PersonalData:DateBirth#).

The 3rd tab 'Ranking' does not contain any obligatory fields, so we can leave it blank for now. However, we need to add a click on the 'OK' button at the end of the task 'NewPlayers'. Be careful when selecting the OK button: The button we need is directly below 'PlayerData' in the window hierarchy, but there is another 'OK' button one level below in 'PlayerData1'. You might wonder where this button comes from, since it is not visible in the Tennis application. The explanation is that sometimes the application's GUI builder generates objects that only become visible under special circumstances. IDATG records all of these objects, even if they are invisible.
The task should now look like this:



For practicing, we recommend that you finish the missing part 'Assign players to tournament' on your own. Of course, you can also invent your own test cases.

Note: Always make sure your AUT is closed (set to initial state) by the test script, so that each test case can start from a well defined starting point. Therefore: Create a building block 'Close_Tennis_Tournaments' with one step that closes the Tennis application.



5.8 Test Execution

Within IDATG all windows are identified by their ID, but the test execution tools need more 'physical' information to identify them. WinRunner uses 'Properties' to assemble unique identification descriptions. IDATG uses the information that you can see in the Window Properties Editor to create descriptions that are sufficient for the test execution tools. Within IDATG these descriptions are called 'Tags'. Where appropriate, the caption of a window is included in the tag.

Now we are finally ready to generate test cases! Just select the 'Project Tennis' item and press 'Generate Test Cases'. IDATG produces 2 test cases: one of them creates a ladies' tournament and 2 female players, the other one a men's tournament and 2 male players. You can convert these test cases into scripts (you have to specify the path to tennis.exe) and try to run them with the test execution tool of your choice.

Exercise 20:

Generate all test cases for the project 'Tennis' and convert them for your test execution tool.

Solution 20:

For help on generating and converting test cases see page 32, solution 8.

5.8.1 Resolving GUI Recognition Problems

IDATG always tries to generate perfect tags for all GUI objects, however, there are some rare cases that cannot be handled automatically and make it necessary to edit the window tags manually.

The general method for resolving such problems is as follows:

- Analyse the problematic object with the GUI Spy of your test execution tool and determine the tag that is expected.
- Modify the object's tag in the IDATG Window Properties Editor accordingly.
- Convert and execute the test cases again.

For instance, if you try to run the Tennis test cases using WinRunner, you will encounter a typical problem: One of the objects (in this case, the OK button in 'PlayerData') is not found correctly. A look at the WinRunner GUI Map explains the problem: As we already know from the last chapter there are 2 OK buttons, which causes IDATG to add the line 'location: 0' and 'location: 1' to their GUI Map descriptions. However, since the button with location=0 is currently invisible, WinRunner omits it during its count and expects the other one to have location=0.
We can solve this problem easily by opening the Window Properties Editor for 'PlayerData >> OK' and deleting the location part in the tag (alternatively, you can also set the location to 0). The tag should now read: *{class: push_button, MSW_class: "!Button", MSW_id: 1}*

Another typical case are window captions that change during the test. For instance, when you start the Tennis application, its title bar reads 'Tennis – Tournaments'. However, as soon as you open the player list it changes to 'Tennis – Players' causing the test execution tool to get confused. Such cases can be solved by using abbreviations and regular expressions in the tag. For WinRunner, the correct tag reads:
{class: window, MSW_class: "!Afx:400000:.", label: "!Tennis.*"}*

After these few manual corrections, your test cases should run flawlessly. Don't despair if there are still some problems, maybe you have only forgotten a detail in the specification. All the effort for the test specification will rapidly pay off in the test maintenance phase.

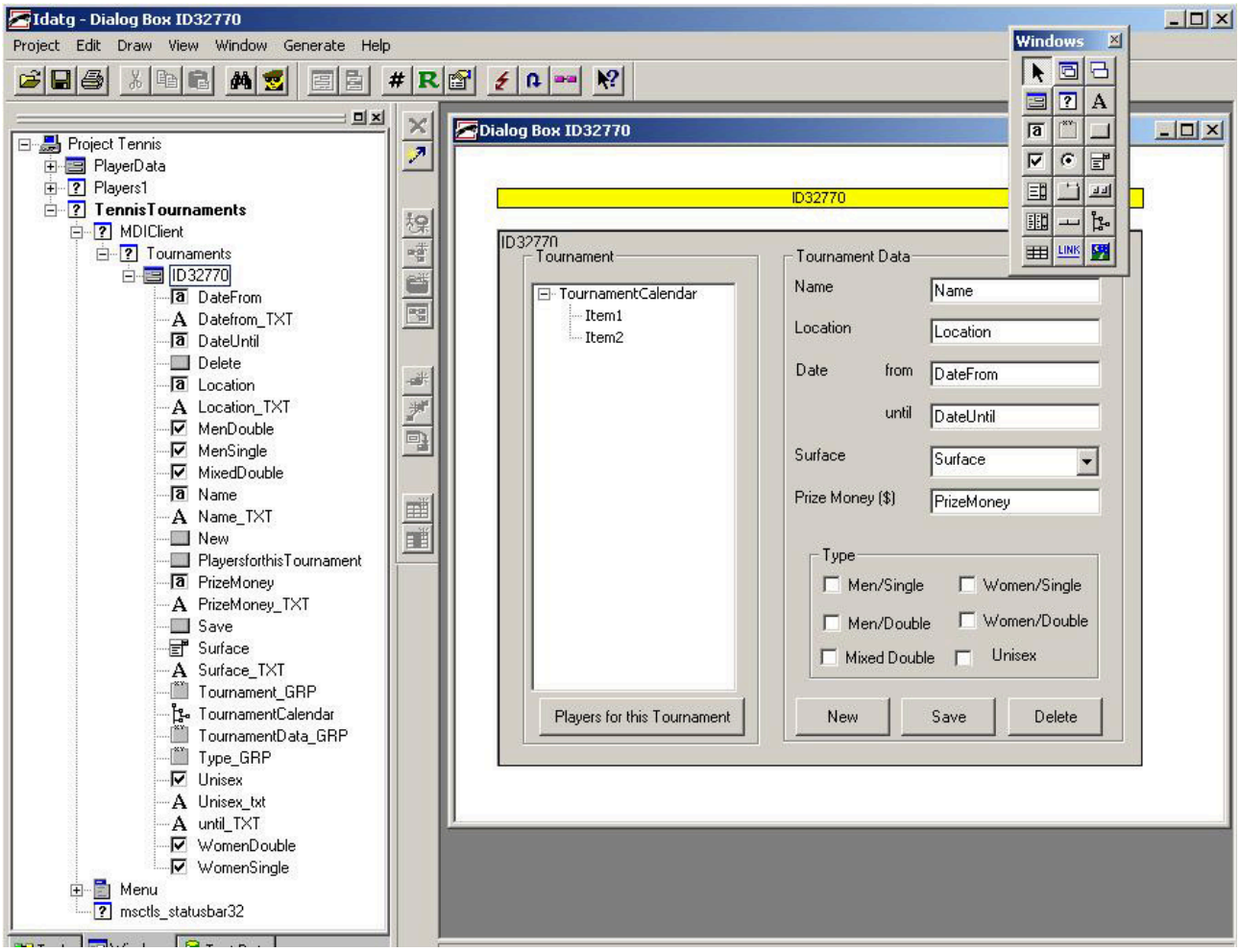
6 Test Maintenance

6.1 Adding a New Object to an Existing Window

Capture the object (window) with the GUI Spy, and drag it over its parent in the windows pane. Then open the Window Editor of this parent and move the added object to the approximate location that it has in the application. (This is only necessary for your convenience, because the identification of the object becomes easier)
If the object cannot be recorded, you can also draw it yourself. Use the 'Window Tool Bar', which is opened automatically when you open a window for editing, to add your object to the window. Then open its Properties Editor and change the WindowID (and the other properties) according to your needs. Take care to create a tag for this object that allows the test execution tool to identify it.

E.g. imagine there is a new type of tournament called 'Unisex' where the sex of the participants is irrelevant. In this case, we would have to add a check box to the 'Type' group box.

- Open the Window Editor for 'Tennis Tournaments >> MDIClient >> Tournaments >> ID32770' (double-click the name in the Windows pane).
- Select 'Check Box'  in the 'Windows' tool bar.
- Draw the check box inside the group box 'Type' (and adjust the size to the size of the other check boxes there). As soon as you release the mouse button, the check box is shown in the windows pane (with the name 'Checkbox1').
- For your convenience, arrange the check boxes inside of the group box 'Type' so that they don't overlap.
- Click right on the new check box and open its Properties Editor by selecting 'Edit Properties'.
- Edit the Window ID and the Caption, e.g. 'Unisex'. You can either change the Tag manually in here, too, or generate it later using the menu 'Generate >> Generate Window Tags'.
- Click 'OK' in the Properties Editor.



Is this your result? Congratulations! Now delete the ‘Unisex’ checkbox and static text.

6.2 Copying a Task

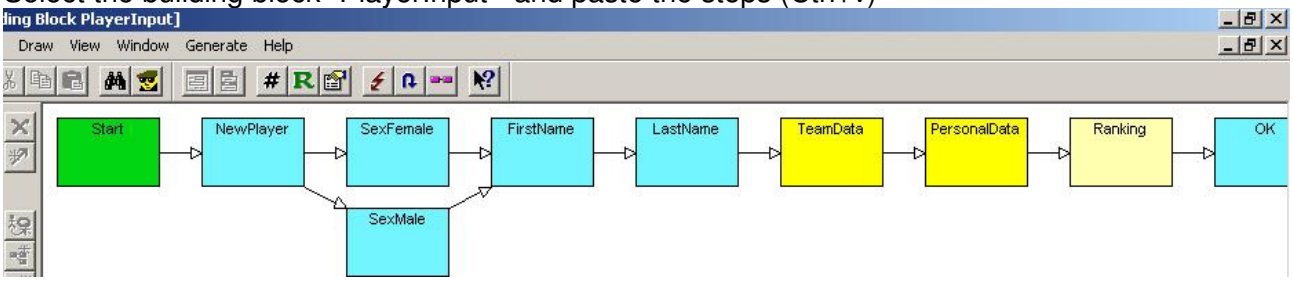
Sometimes it may be useful to copy a task and then modify it. Copying a task without any modifications does not make sense, because you can use one and the same task as often as you want as a sub task inside the task flows. Also, copying a task can often be avoided by parameterizing it. Always remember to define tasks as re-usable building blocks that facilitate the test maintenance.

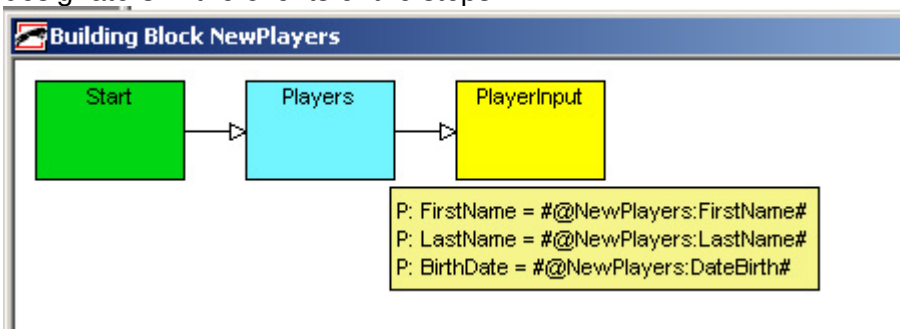
- Select a task and press ‘**Copy**’ in the toolbar or the menu
- Select another task
- Press ‘**Paste**’ to insert the copied task (and its children) as child of the selected task.

6.3 Splitting a Building Block Task

If you find out you would like to re-use a part of a building block, make it a separate building block inside the existing one (to avoid adjusting your existing test cases)
In the Tennis example the building block “NewPlayers” contains one step to navigate in the application, and steps for input into edit fields that you need to repeat to enter more than one player (an example of bad building block design).

- Create a new building block and call it “PlayerInput”.
- Open the task flow of “NewPlayers”
- Select all but the first steps

- and cut them pressing Ctrl+x
- Select the building block “PlayerInput “ and paste the steps (Ctrl+v)
- 

- insert the building block “PlayerInput” where you cut the steps out of “NewPlayers”
- Make the parameters of “NewPlayers” available to “PlayerInput”, and adjust the designators in the events of the steps
- 

This way all tasks that you already defined will remain the same, and you may use “PlayerInput” independently as well.
Note: Avoid complex hierarchy in building blocks – they become too specialized, as you cannot set parameter values independently for each building block depending on your test case.

6.4 Changing Parent-Child Relationships

Keep in mind that the parent-child relationship in the Task pane is only organizational – to make it easier to find tasks. It is not functional (i.e. has nothing to do with the actual test sequence)!

by Drag & Drop
Simply drag the desired child task with the mouse over its new parent.

by Cut & Paste

- Select a task or folder and press ‘**Cut**’ on the toolbar or the menu (“Edit >>Cut”) or (Ctrl +x). The selected element is displayed green.
- Select a folder or the project

- Press ‘**Paste**’ to move the green task (and its children) to the selected task. If you want to leave the Cut-mode without pasting the task, press ‘**Cut** ’ a second time. The task will stay where it is (it is not deleted!).

6.5 Deleting a Task

The selected task may be deleted by choosing the menu 'Edit >> Delete' or by pressing the corresponding toolbar button . The task will be deleted with all its children. If the task is used as sub task in other task flows, the corresponding steps become undefined.

6.6 Adding a new Connection

- Press the toolbar button '**Connect Steps**'
- Select the start step
- Select the destination step. Cycles in the graph are not allowed.

6.7 Deleting a Connection

A selected connection may be deleted by choosing the menu 'Edit >> Delete' or by pressing the corresponding toolbar button  . If the destination step becomes isolated (has no predecessors), it will be connected to the start step of this task.

6.8 Deleting a Window

A selected window may be deleted by choosing the menu 'Edit >> Delete' or by pressing the corresponding toolbar button  . The window will be deleted with all its children and transitions. Semantic dependencies must be removed manually. Therefore use 'Search References' (for all child windows, too) before deleting the window to find affected task steps.

6.9 Creating new Menus

Since menus are a very special type of window, you cannot draw them in the **Window Editor**. However, new menus can be created using the menu item **'Draw | New Menu'**. IDATG then opens a new **Menu Editor** showing an empty menu. Menu Items can now be added using **'Draw | New Menu Item'**.

Exercise 21:

Add an item 'Maximize' to the menu 'Window'.

Solution 21:

Expand 'TennisTournaments' and double-click 'Menu' in the 'Windows' pane of the Application Editor. The Menu Editor opens.

- Right-click 'Window' and select 'New Menu Item'
- Right-click on the item 'Item' and select 'Edit Properties'.
- Change WindowID and Caption to 'Maximize'. (for 'Maximize' use '&Maximize' as caption).
- Click 'OK' to close the Properties Editor.
- Make sure that the item is still selected and use the menu 'Generate >> Generate Window Tags' to create a proper tag for the new item.

6.10 Replacing a window

Sometimes windows undergo thorough changes (several windows are added, or window types are changed, e.g. an input field becomes a combo box). In such cases it is more convenient to use the Replace Window function instead of adding each new window separately.

Caution:

- Only replace windows with objects of the same (or a compatible) type, so that the events are still applicable.
- When replacing radio buttons, you have to replace the group box (because the group box is the window that is referred to). Make sure that the buttons in that group box have the same captions (else you have to edit your task steps).

Replacing

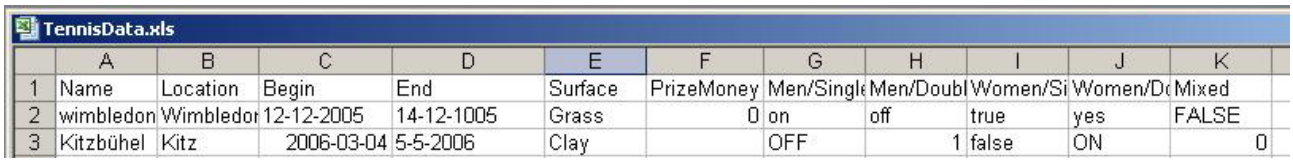
- Record the new window with all its windows (elements)
- Rename windows were appropriate (e.g. check boxes)
- Open the Window Editor for the old and the new window
- You have to replace each single element that is used somewhere in a task (replacing the unused ones is not necessary).
- Right click the window you want to replace in the old window and select “Replace Window” from the context menu.
- In the “Replace Window” dialog box click “Browse Windows” for the new window
- In “Window Selection” double click the new window (or select it and click “OK”)
- Confirm the replacement by clicking “OK” in the “Replace Window” dialog box
- After replacing all affected windows, you should delete the old windows from the window hierarchy.

7 Data-oriented tests

7.1 Importing Test Data

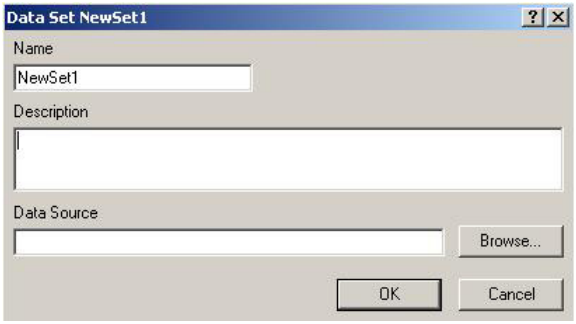
Another way of using one building block with different input data is to read the data from a file (during test creation).

Let us return to our Tennis example. Surely you would like to edit data for new tournaments in a convenient tool like Excel, where you can see them together. The data may be specified in whatever way you like, as long as you convert them into a tab-separated file or a .csv file afterwards. Here we use an Excel sheet and convert it into a .csv file.


TennisData.xls

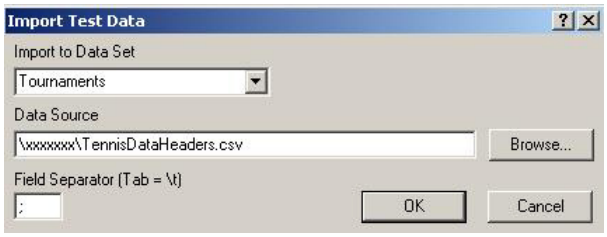
	A	B	C	D	E	F	G	H	I	J	K
1	Name	Location	Begin	End	Surface	PrizeMoney	Men/Singl	Men/Doubl	Women/Si	Women/D	Mixed
2	wimbledon	Wimbledor	12-12-2005	14-12-1005	Grass	0	on	off	true	yes	FALSE
3	Kitzbühel	Kitz	2006-03-04	5-5-2006	Clay		OFF	1	false	ON	0

To make these data available to the project we use IDATG's data import function:
Click the “Test Data” tab at the bottom of the application editor, then click right somewhere in the panel. Two options are available at this moment: “New Data Set” and “Import Test Data”. “New Data Set” opens the properties editor for data sets that allows you to specify a name and a description for the data set.

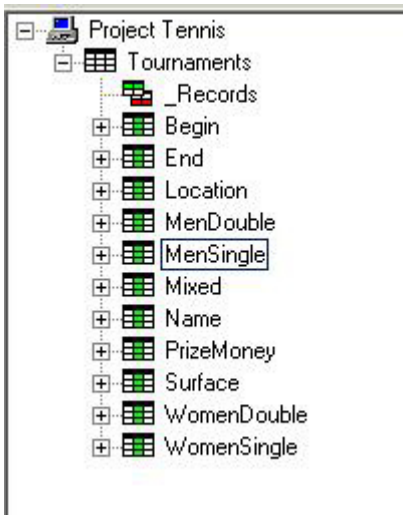


We change the data set name to “Tournaments” and click OK. Expand the tree in the Test Data panel and double click “Records”. You see three column names, but no data yet. Close the editor, as we will not edit our test data here.
We could import our prepared test data without any preparation in IDATG. This would create two records containing the lines 2 and 3 of the above Excel sheet, with column names given in line 1, and the default type “string” for every entry. But this is not very convenient for continuing, as we obviously have Boolean items as well. There are several solutions to this problem, and all need some manual definitions (the data functions are new to IDATG, and will be enhanced and improved in the versions to come). I suggest to import just the first line (=the column headers), adjust the types, and then import the data records. This method ensures that you get the correct spelling and the correct order of columns.

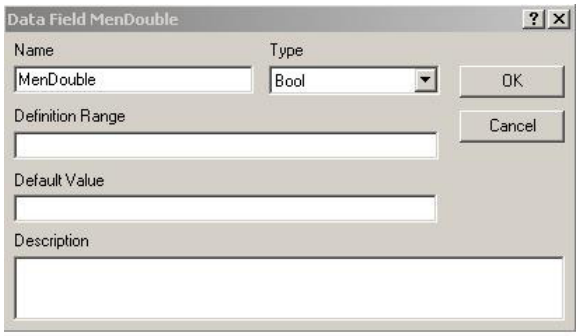
Create a .csv file from the first line of the Excel sheet. Right-click “Tournaments” in the test data tree, and select “Import Test Data”.



In the ‘Import Test Data’ window browse for your header file, then click ‘OK’.
You should see the following when you extend the node ‘Tournaments’:



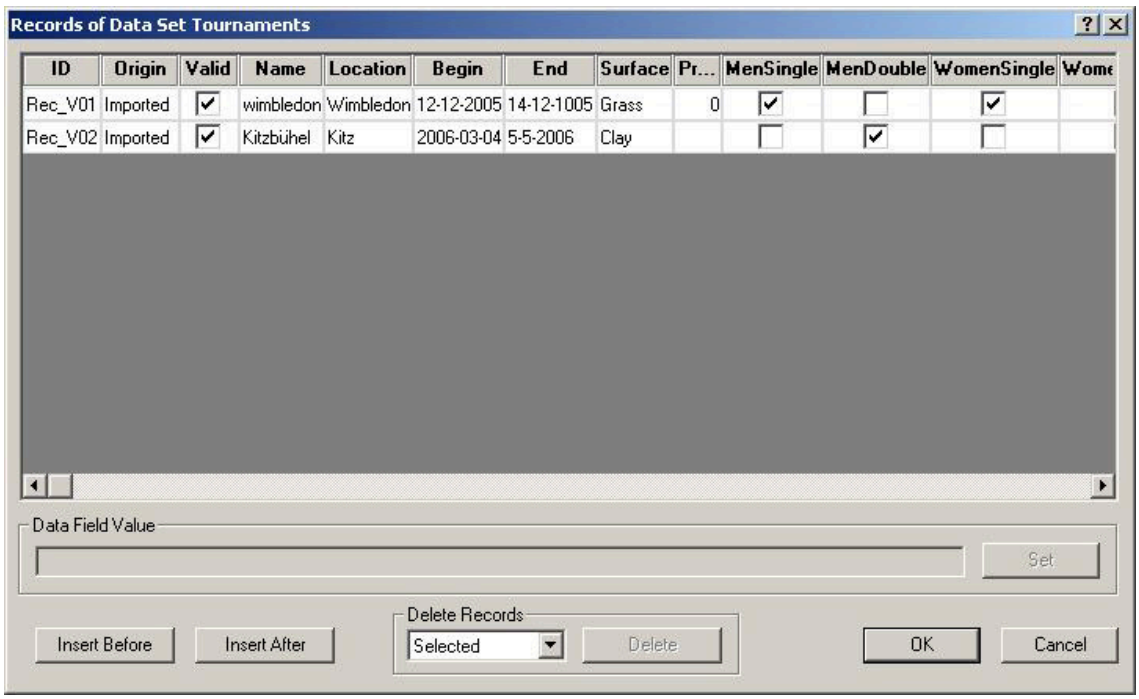
Now edit the properties of the columns that contain boolean values by double clicking them or choosing “Edit Properties” from the context menu. Select “Bool” from the “Type” drop down list and press “OK”.
Hint: When you change the type, you will be asked whether you want to generate default equivalence classes for the selected field. Confirm this question with ‘Yes’, for explanations see below (generating test data).



During the import of the test data, Boolean values are transformed into the values TRUE and FALSE if possible, which are later displayed as check boxes.

Let us import the actual records now, following the same procedure as before: right-click ‘Tournaments’ and select ‘Import Test Data’. In the ‘Import Test Data’ dialog box the data set ‘Tournaments’ is displayed in the combo box “Import to Data Set”. Browse for the .csv file containing all lines of your Excel sheet, and set the Field Separator correctly (default for .csv is the semicolon “;”). Press ‘OK’ and accept the success message.

You can check the imported data by expanding ‘Tournaments’ in the Test Data panel, and double clicking ‘_Records’. The first two columns are created by IDATG and contain a unique ID and the origin (manual, imported, generated) of the record.



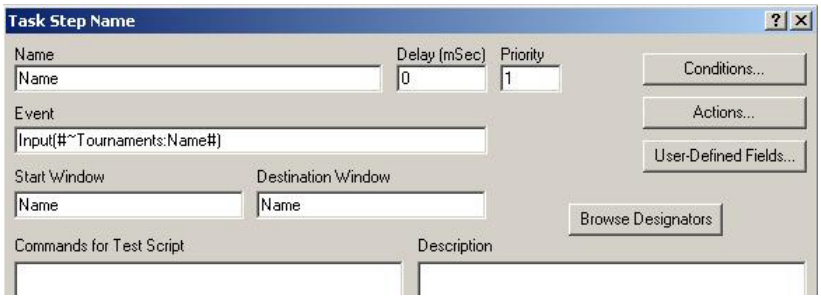
As you can see, true/false, on/off and 1/0 have all been interpreted as boolean values. You can use these data in test steps just like any other designators.

Exercise 22

Create a building block for a new tournament, using the imported data.

Solution 22:

1. Create a folder in the task panel and call it ‘Data oriented building blocks’
2. Create a new building block in this folder, and call it ‘NewTournament_Data’
3. Copy all steps of the building block ‘NewTournament’ to ‘NewTournament_Data’
4. Replace the static values of the ‘Input’ events by designators for the test data in the records by doing the following:
   - Open the step editor for ‘Name’
   - Select “Wimbledon”, then click ‘Browse Designators’
   - In the ‘Designator Selection’ window expand TEST DATA | Tournaments and double click ‘Name’



   - Click ‘OK’
5. Repeat step 4 for the other fields ‘Location’ thru ‘Women double’

6. Delete the conditions in the steps ‘MenSingle’ and ‘WomenSingle’; the step ‘SetFemale’ is not necessary any longer; it does not really harm, because IDATG will create two test cases because of the two data records anyhow.
7. Instead of steps 3 thru 6 you can create a step for each input field, and assign the corresponding designators to their events.
8. Insert a step for the selection of menu ‘View’ >> ‘Tournaments’ as the first step of this building block.

You can use this building block in any test case just like all other building blocks. However, if you want IDATG to generate test cases for all your data records, you have to assign the data set to the task: Right click ‘Project Tennis’ in the Tasks pane, select ‘New Task’. In the properties editor of the new task, change the name to ‘TC_R1_NewTournament’, and assign it to Data Set Tournaments; click ‘OK’
The task flow of ‘TC_R1_NewTournament’ will consist of only 2 building blocks, that is ‘NewTournament_Data’ and ‘Close_Tennis_Tournaments’.

Exercise 23

Create the test case ‘TC_R1_NewTournament’, generate and convert it, and view the result.

Solution 23:

If you have not done the steps described above yet, do them now. Then add a step to the task flow of ‘TC_R1_NewTournament’, and assign it to the building block ‘NewTournament_Data’. Add another step assigned to the building block ‘Close_Tennis_Tournaments’.

Select ‘TC_R1_NewTournament’, then generate test cases (click ).
Confirm the messages.

You can view the two test cases now by clicking  .

7.2 Generating test data records

IDATG offers the possibility to generate a set of test data records automatically, based on your definition of equivalence classes for each data field. The aim of this record generation is to combine representative values systematically to get a high coverage of code with few test cases. The values themselves are not generated, but defined by the user in the Equivalence Classes Editor.

An **equivalence class** of an input variable i is a set of values that fulfills the following:

- the set covers a **contiguous value range**
- a **similar calculation rule applies** (input equivalence) OR
- **similar outputs** are to be generated (output equivalence)

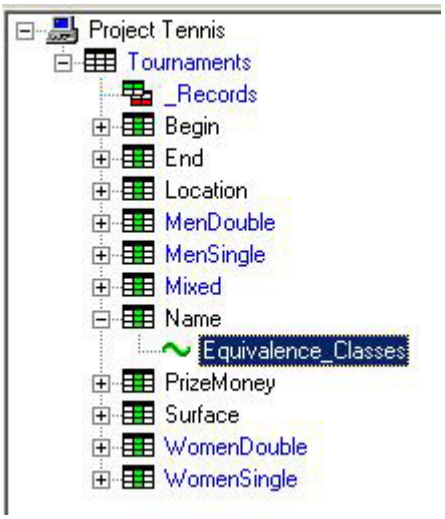
An example for defining equivalence classes:

```
int MyFunc(int i) {
        if (i<0) return -1;              // error
        else if (i<20) return sqrt(i);   // square root(i)
        else return i*i;                 // i squared
}
```

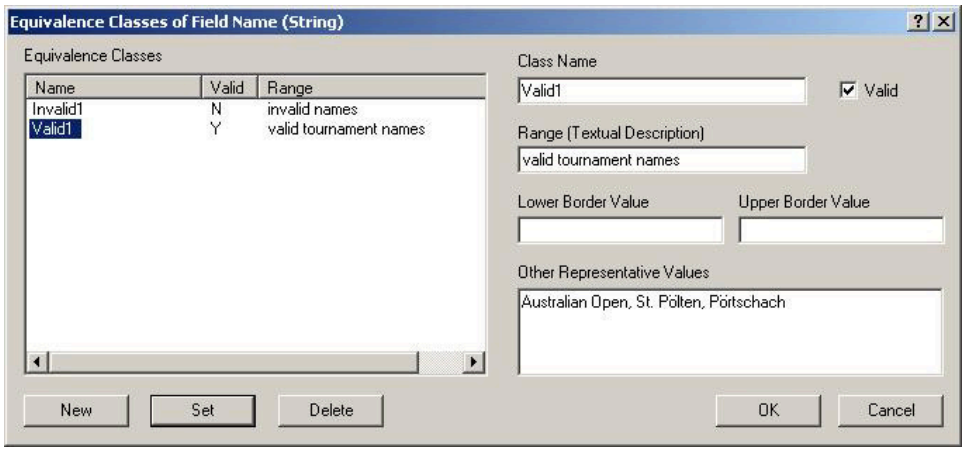
ID	Equivalence class	Expected result
I1	[MININT..-1]	-1
V1	[0..19]	√i
V2	[20.. √ MAXINT]	i^2
I2	[√ MAXINT+1..MAXINT]	undefined (overflow error)

7.2.1 Defining equivalence classes

To create equivalence classes for a data field extend the node of the data field (e.g. ‘Tournament|Name’), so that the tree item ‘Equivalence_Classes’ becomes visible.



Open the Equivalence Class Editor in one of these ways:
1) Double click on the tree item
2) Right click on the tree item (pop-up menu) and choose ‘Edit Properties’
3) Select the tree item and select the menu item ‘Window’ >> ‘Properties Editor’



Now you can define valid and invalid equivalence classes for the selected data field. To create an invalid class, deselect the 'Valid' checkbox. The class name must start with a character. In the 'Range' edit field you may write a textual description of this equivalence class (comment). Enter values into 'Lower Border Value', 'Upper border Value' and 'Other Representative Values' according to your analysis of the requirements. The values are used for the generation of test data records.
Save your entries clicking the 'Set' button.

Exercise 24

Create at least one valid and one invalid equivalence class for the data field 'Tournaments|Name'.

Solution 24:

1. Extend the tree node 'Tournaments|Name'
2. Open the equivalence class editor by double clicking on the tree item 'Equivalence_classes' of the data field''Name'
3. Insert 'Class name'="Valid1", 'Range' = "valid tournament names" and 'Other representative values'= "Australian Open, St. Pölten, Pörtschach"
4. Save the entries with the 'Set' button
5. Push the 'New' button to define another class.
6. Fill out the fields 'Class name' = "Invalid1", 'Range' = "invalid names" and 'Other representative values' = "error4test"
7. Save the entries with the 'Set' button
8. Click 'OK' to confirm the changes.

For further exercises we need to define equivalence classes for each data field. At the beginning of the "Data oriented tests" section (Importing test data) a hint was given to generate default equivalence classes. So the data fields 'Begin', 'End', 'MenDouble', 'MenSingle', 'Mixed', 'WomenDouble', WomanSingle', 'PrizeMoney' and 'Name' (Exercise 24) have default equivalence class entries.

For generating records based on equivalence classes all data fields must have at least one equivalence class. So we have to create also equivalence classes for the data fields 'Location' and 'Surface' manually.

Exercise 25

Create valid (optionally invalid) equivalence classes for the data fields 'Tournaments|Surface', 'Tournaments|Location' and Tournaments|PrizeMoney.

☑ Solution 25:

1. Extend the tree node ‘Tournaments|Surface’
2. Open the equivalence class editor by double clicking on the tree item ‘Equivalence_classes’ of the data field ’’Surface’
3. Fill out at least the fields ‘Class name’=”Valid1”, ‘Range’ = “valid surfaces” and ‘Other representative values’= “Carpet, Clay, Grass, Hard, Rebound Ace”
4. Save the entries with the ‘Set’ – button
5. Push the ‘New’ – button to enter a new entry.
6. Fill out the fields ‘Class name’ = “Invalid1”, ‘Range’ = “invalid surface” and ‘Other representative values’ = “PVC”
7. Save the entries with the ‘Set’ – button
8. Click ‘OK’ to confirm the changes.
9. Extend the tree node ‘Tournaments|Location’
10. Open the equivalence class editor by double clicking on the tree item ‘Equivalence_classes’ of the data field ’’Location’
11. Fill out at least the fields ‘Class name’=”Valid1”, ‘Range’ = “valid locations” and ‘Other representative values’= “Wimbledon, Paris, London, Miami”
12. Save the entries with the ‘Set’ – button
13. Click ‘OK’ to confirm the changes.
14. Extend the tree node ‘Tournaments|PrizeMoney’
15. Open the equivalence class editor by double clicking on the tree item ‘Equivalence_classes’ of the data field ’’PrizeMoney’
16. Now we edit the existing valid equivalence class ‘V1’ → select the list entry.
17. Fill out or change the fields, ‘Lower Border Value’ = “100”, ‘Upper Border Value’ = “25000000”and ‘Other representative values’= “345000, 1000000”
18. Save the entries with the ‘Set’ – button
19. Click ‘OK’ to confirm the changes.

7.2.2 Record generation

Test record generation is always for one data set. So in order to avoid explosion of possible combinations, keep the data sets as small as possible (e.g. separate set for each building block). Depending on your focus and on requirements you can choose various types of generation.

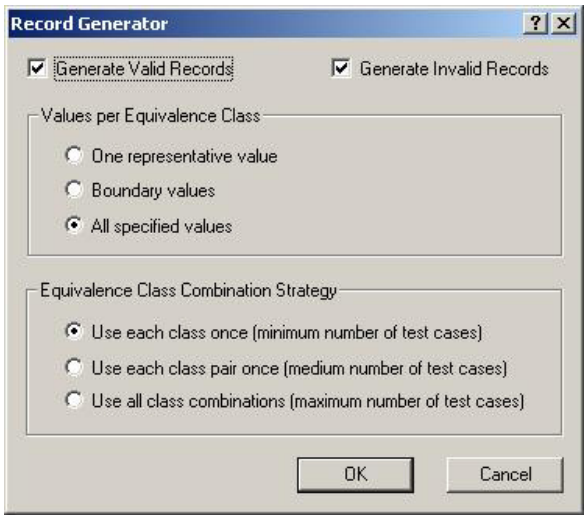
To open the ‘Record Generator’ right-click on any item in the data set and select the menu item ‘Generate Data Records’ from the pop-up menu.

Exercise 26

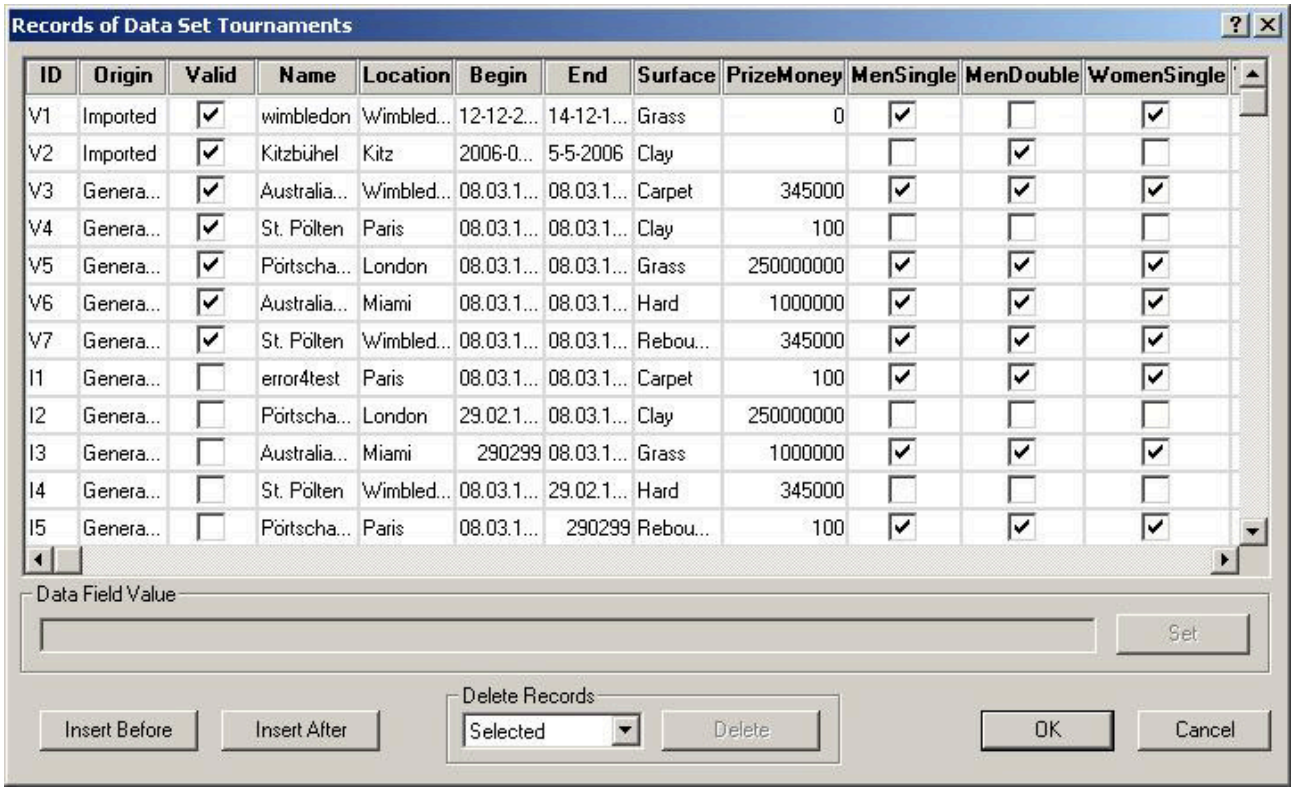
Generate the minimum number of records using all specified values.

☑ Solution 26:

1. Open the Record Generator.
2. Select the option ‘Generate Valid Records’
3. Select the option ‘Generate Invalid Records’
4. Select the option ‘All specified values’
5. Select the option ‘Use each class once (minimum number of test cases)’
6. Confirm your choice with the ‘OK’ button.
7. Confirm the message that 14 records were generated by clicking the ‘OK’ button.



8. Then open the ‘Records’ dialog to see the generated records.



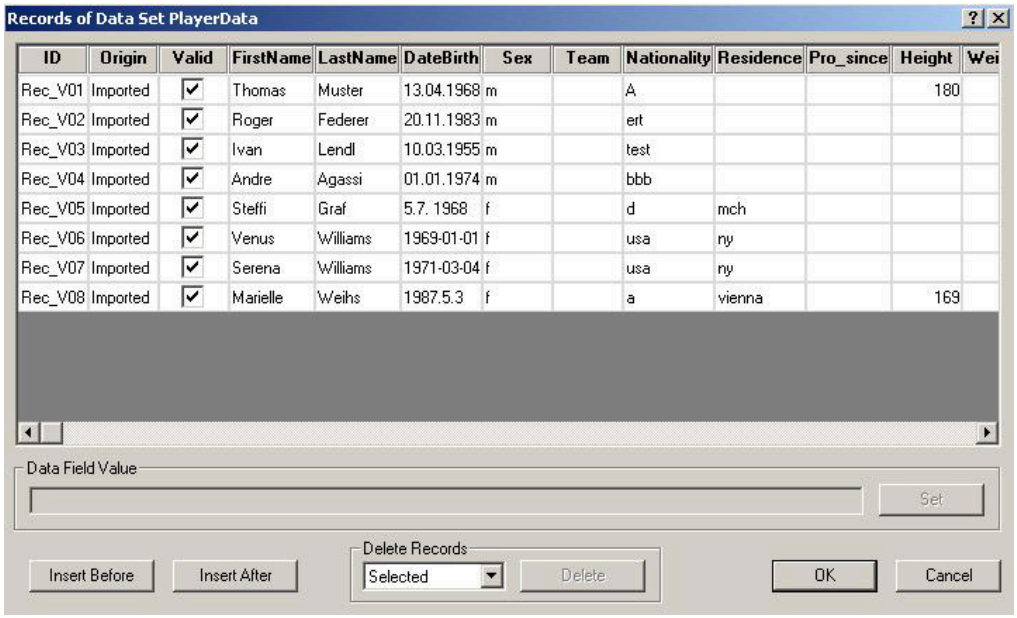
7.3 Working with loops

In the exercise above we created one test case for each record of our data set, but sometimes you will wish to run through a data set within one test case. For this purpose use the SubTask Editor to define a loop.

7.3.1 Data sets

Our Tennis application serves for defining tournaments, including the players. As one player is not enough, we enter several players in a loop. Let us start creating the data records, using Excel. We need columns for each edit field in the window “Player Data”, including the three tab pages.

Invent several players and their data (at least the obligatory ones), then import the data into your IDATG project, eg:



and call the data set “Players”.

Loops can be defined for subtasks only. As we already have a building block to enter player data (NewPlayers), we could use this building block to enter several players.
Again, there is more than one way to define the input data for that building block. The easiest way seems to be to assign the already existing parameters to the designators of the data set.

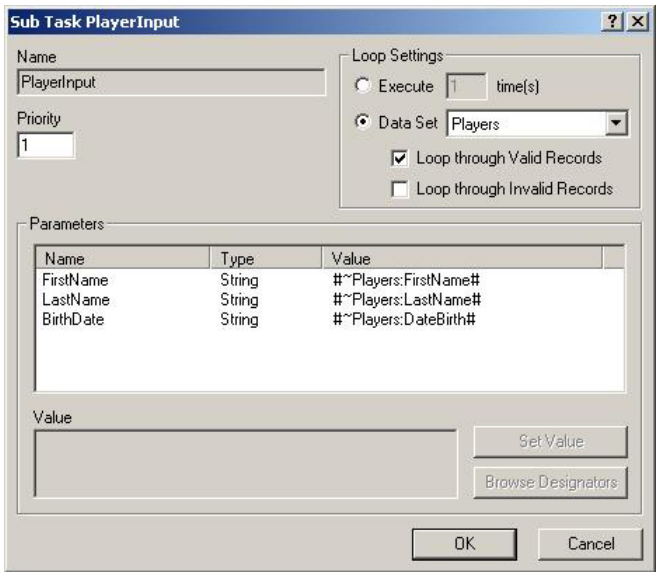
It would work for the 3 parameters defined; but the block was not designed to set values according to a data set, and needs some adaptations.

Exercise 27

Create a test case that enters first name, last name and date of birth of all players of your data set into the tennis application.

Solution 27:

1. Create a new data set ‘PlayerData’ (for help see Importing Test Data)
2. Import the data records you defined in Excel into this new data set.
3. Create a new test case TC_R2_Players_minimal
4. Add a step to click the menu “View > Players”
5. Add the building block “PlayerInput” to the task flow of TC_R2_Players_minimal
6. Open the “Sub Task Editor” (Window >> Properties Editor) of the building block “PlayerInput”
7. Edit the “Loop Settings”
   a. Activate the radio button for “Data Set”
   b. Select the data set ‘Players’ from the combo box
   c. Check “Loop through Valid Records”
8. Edit the parameter entries ‘FirstName’, ‘LastName’ and ‘DateBirth’. Change the values to the corresponding datafields of the dataset “Players” using the “Browse Designators” function. E.g. FirstName gets value ‘#~Players:FirstName#’ (see Parameters in Embedded Tasks for help).



9. Close the SubTask Editor with “OK”

If you generate this test case and look at the test steps, you will realize that all players are considered to be male, as we never set the sex attribute to “female”. Now we will make the sex depending on the value in the data record.

- Select the step “SexFemale” of the task “PlayerInput”, open it for editing, and click the button “Conditions…” Replace the old condition #:Sex#="female" with a new one:
- click the existing condition and delete it
- click “Browse Designators”, and select “Project Tennis > TEST DATA > Players > Sex” from the Designator Selection window.
- click “OK”
- edit the condition: #~Players:Sex#="f" and click “Set”, then click “OK” in the Condition Editor, then “OK” in the step editor.

Note that the value must match the values in your data records exactly. If you have values “f” and “F” and “Female” and “FEMALE”, create a compound condition.

Exercise 28

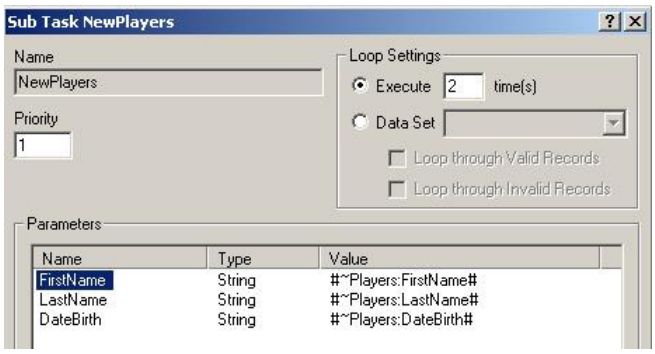
Edit the step “SexMale” so that it will be generated for values “m” and “M” in the column “Sex” of the data set “Players”.

Solution 28:

Open the step “SexMale” of the building block “PlayerInput”
Click “Conditions…”
Select the condition #:Sex#="male", and mark it in the editing line
click “Browse Designators”, and select “Project Tennis > TEST DATA > Players > Sex” from the Designator Selection window.
edit the condition: #~Players:Sex#="m" OR #~Players:Sex#="M" and click “Set”, then click “OK” in the Condition Editor, then “OK” in the step editor.

7.3.2 Fixed number of executions

The simplest way of repeating several steps is identical execution a preset number of times. In our Tennis example we use this for an error test case, where we try to create 2 identical players.



To make it a proper test case, you will need a few steps to handle the error message.

7.3.3 Selecting records out of a data set

IDATG has several actions to select a record in a data set. They are
ResetDataRecord (setting the first record)
GotoDataRecord (using the RecordID)
NextDataRecord

For error test cases you often need a special record to provoke a certain error message. To set this record, use the action GotoDataRecord.

7.4 Summary data-oriented test cases

Your methods to enter data are:

- constants
- parameters
- data sets
    - created manually
    - imported from a file
    - generated, using equivalence classes

8 Conclusion

Congratulations! You have successfully worked your way through the IDATG tutorial and should now be able to create simple test specifications on your own. With a bit of practice, you will be able to automate test cases even for huge projects. After a few regression tests, you (and your managers) will find that the effort for test execution and maintenance has been reduced considerably!