Atos

TEMPPO Designer (IDATG)

Version 16.0

User´s Guide - Part II

User Interface

July 2014

Publisher:
Atos IT Solutions and Services GmbH
SI SOL TES
Siemensstrasse 92
1210 Vienna, Austria

Document Management

History of changes

Version	Status	Date	Person resp.	Reason for Change
1.0	Ready	20.10.98	S.Mohacsi	Creation
[…]				
8.0	Ready	09.02.01	S.Mohacsi	New IDATG Version V8.0
8.1	Ready	27.04.01	S.Mohacsi	New IDATG Version V8.1
8.2	Ready	01.06.01	S.Mohacsi	New IDATG Version V8.2
9.0	Ready	30.09.01	S.Mohacsi	New IDATG Version V9.0
9.1	Ready	11.01.02	S.Mohacsi	New IDATG Version V9.1
10.0	Ready	02.09.02	S.Mohacsi	New IDATG Version V10.0
10.1	Ready	01.07.03	S.Mohacsi	New IDATG Version V10.1
10.2	Ready	17.10.03	S.Mohacsi	New IDATG Version V10.2
11.0	Ready	09.07.04	S.Mohacsi	New IDATG Version V11.0
11.1	Ready	17.09.04	S.Mohacsi	New IDATG Version V11.1
11.2	Ready	31.03.05	S.Mohacsi	New IDATG Version V11.2
11.3	Ready	27.07.05	S.Mohacsi	New IDATG Version V11.3
12.0	Ready	06.12.05	S.Mohacsi	New IDATG Version V12.0
12.1	Ready	15.05.06	S.Mohacsi	New IDATG Version V12.1
12.2	Ready	02.10.06	S.Mohacsi	New IDATG Version V12.2
12.3	Ready	14.05.07	S.Mohacsi	New IDATG Version V12.3
12.4	Ready	01.10.08	S.Mohacsi	New IDATG Version V12.4
14.0	Ready	10.08.10	S.Mohacsi	New IDATG Version V14.0
14.1	Ready	23.02.11	S.Mohacsi	New IDATG Version V14.1
15.0	Ready	30.06.12	S.Mohacsi	New IDATG Version V15.0
16.0	Ready	31.07.14	S.Mohacsi	New IDATG Version V16.0

Document was created using the following tools:

Microsoft Office WinWord 2010

Microsoft Visio 2010

Corel Paintshop Pro 10.0

Contents

1 Introduction

This manual describes the use of **TEMPPO Designer** Version 16.0. The tool is also called **IDATG** (Integrating Design and Automated Test Case Generation). For historical reasons and to avoid confusion with other components of the TEMPPO framework, only the term IDATG will be used in the following.

1.1 Validity of the manual

- This user manual is valid for IDATG V16.0.
- For information about the test-execution tools QuickTest, SilkTest, TestPartner, and WinRunner please refer to the respective user guides.

1.2 Relationship with other documents

- The first part of this User's Guide contains a detailed description of how to do model-based testing.
- A tutorial containing a step-by-step description of the IDATG method is also available.
- The tool IDATG provides online help that gives a detailed description of its functionality. The help text is based on this document.

1.3 Target group

This user manual is recommended for

- Test managers
- Test case designers
- Testers
- GUI Designers

What prior knowledge is the reader expected to have?

- Basic knowledge about: GUI design, task modeling, state-transition diagrams, test case design.

2 Workspace (Application Editor)

The main tool for navigating through a project is the **Application Editor** or **Workspace** that is usually located at the left side of the IDATG GUI. It contains four tab pages showing the project's task, requirement, window, and data structure.

2.1 Task Hierarchy (Task Editor)

The editor always contains a root node called 'Project', from which the entire task hierarchy is descending. It is recommended to use the project's software requirements specification as basis for entering the tasks into the editor.

In the tree, the following types of objects may be found:

- Folder: Used for structuring the task hierarchy. Like tasks, folders can have properties like test type, test level etc. and may also be assigned to requirements.
- TEMPPO Test Manager Folder: If IDATG is called from TEMPPO Test Manager, the imported test structure is represented as blue folders to avoid confusion with normal IDATG folders. In order to guarantee consistency with TEMPPO Test Manager, some special rules and restrictions apply to editing items in this type of folder. For more information, see User's Guide Part I.
- Use case task: A complete test sequence, usually consisting of smaller building blocks. Generated test cases are always assigned to the selected use case task.
- Use case task assigned to a data set: The additional database symbol indicates that this use case task has been assigned to a data set and can be used for generating data-oriented test cases.
- Building block tasks: A re-usable sequence of test steps. May also consist of other building blocks.

Tasks whose task flows have already been defined are displayed blue. The same applies to their parent folders. If the current project is a sub project, the tasks from its master project are displayed grey and marked as "READ-ONLY".
Use case tasks for which test cases have already been generated, are displayed **bold**.

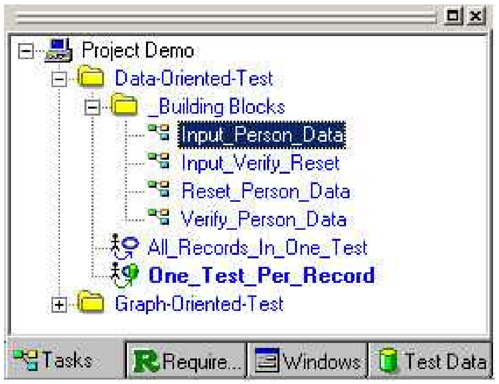


Figure 1 - Application Editor - Task Hierarchy

Adding a new Task

- Select the parent folder of the new task
- Press '**New Task**' in the toolbar or the menu. The new task will be inserted as child of the selected folder.

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 (exception: folders), 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.

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. (See **Task Properties Editor**)

Copying a Task

- 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.

Moving Tasks to a Different Folder

There are 2 ways of moving a task:

1. Drag & Drop

Simply drag the desired task with the mouse over its new folder.

2. Cut & Paste

- Select a task and press '**Cut**' in the toolbar or the menu. The selected task is displayed green.
- Select a folder
- Press '**Paste**' to move the green task to the selected folder. If you want to leave the Cut-mode without pasting the task, press '**Cut** ' a second time.

Editing the Task Flow

Double-click the desired task or select it and press the toolbar button '**Edit Task Flow**'. The **Task Flow Editor** opens where you can define the steps of the task.

Deleting a Task

The selected task may be deleted by choosing the menu '**Delete**' or by pressing the corresponding toolbar button. If the task is used as sub task in other task flows, the corresponding steps become undefined.

2.2 Requirement Hierarchy

The editor always contains a root node called 'Project', from which the entire requirements hierarchy is descending. Each requirement is represented by its ID and title.

Requirements that have already been assigned to a task are displayed blue, others black. This feature makes it very easy to track the achieved test coverage. If the current project is a sub project, the requirements from its master project are displayed grey and marked as "READ-ONLY".

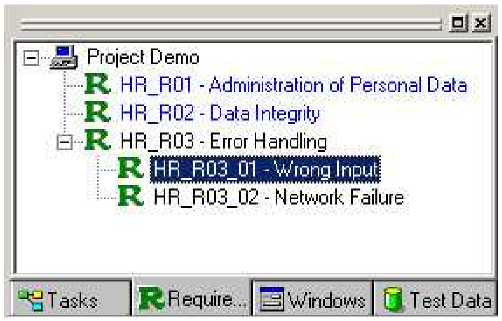

Project Demo
HR_R01 - Administration of Personal Data
HR_R02 - Data Integrity
HR_R03 - Error Handling
HR_R03_01 - Wrong Input
HR_R03_02 - Network Failure

Figure 2 - Application Editor - Requirement Hierarchy

Adding a new Requirement

- Select the parent of the new requirement
- Press '**New Requirement**' in the toolbar or the menu. The new task will be inserted as child of the selected folder.

Editing the Requirement Name

Edit the name of the requirement 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 **Requirement Editor** to edit the name.

Editing Requirement Properties

The properties of the selected requirement may be edited by choosing the menu **'Properties Editor'** or by pressing the corresponding toolbar button. (See **Requirement Editor**)

Changing Parent-Child Relationships

A parent-child relationship between two requirements can be edited very easily with drag&drop. Simply drag the desired child requirement with the mouse over its new parent. The root node itself cannot be dragged.

Displaying Tasks Assigned to a Requirement

All tasks that have been assigned to a certain requirement can be highlighted by selecting the requirement and choosing the menu '**Search References**'.

Deleting a Requirement

The selected requirement may be deleted by choosing the menu '**Delete**' or by pressing the corresponding toolbar button.

2.3 Window Hierarchy

The hierarchy levels of the tree represent the parent-child relationships between the windows. Like in the task hierarchy, the editor always contains a root node called 'Project', 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. The same applies to their parent windows. If the current project is a sub project, the windows from its master project are displayed grey and marked as "READ-ONLY".

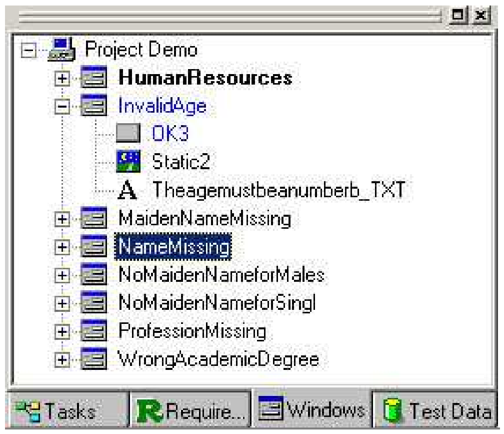

Project Demo
HumanResources
InvalidAge
OK3
Static2
Theagemustbeanumberb_TXT
MaidenNameMissing
NameMissing
NoMaidenNameforMales
NoMaidenNameforSingl
ProfessionMissing
WrongAcademicDegree

Figure 3 - Application Editor - Window Hierarchy

Selecting a Window

Simply 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. 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 '_' and be up to 50 characters long. You can also use the **Window Properties Editor** to edit the window ID.

Editing Window Properties

The properties of the selected window may be edited by choosing the menu **'Properties Editor'** or by pressing the corresponding toolbar button. (See **Window Properties Editor**)

Copying a Window

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

Changing Parent-Child Relationships

There are 2 ways of moving a window:

1. **Drag & Drop**

Simply drag the desired window with the mouse over its new parent.

2. **Cut & Paste**

- Select a window and press '**Cut**' in the toolbar or the menu. The selected window is displayed green.
- Select the new parent
- Press '**Paste**' to move the green window to the selected parentr. If you want to leave the Cut-mode without pasting the window, press '**Cut** ' a second time.

Take care that the child does not exceed the parent's size. If you want a window to have no parent, drag it over the root node ("Project"). Note: Cycles are not allowed in the window hierarchy (if A is parent of B, B cannot also be parent of A). The root node itself cannot be dragged.

Choosing a new Start Window

The start window is the one that appears first when the tested application is executed. In the **Application Editor** its name is displayed **bold**. To change it, simply select the desired window and press '**Set as Start Window**' in the '**Edit**' menu.
Note: Only a window that has no parent and is visible and enabled can be a start window.

Editing the Window Layout

There are 2 possibilities: either select the desired window and press the toolbar button '**Edit Window**' (or '**Edit Menu**' in case of a menu) or double-click on the desired window.

Deleting a Window

The selected window may be deleted by choosing the menu '**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.

Creating new parentless Windows

New windows are usually drawn in the **Window Editor** of their parent window. However, there is also a possibility to draw windows that have no parent: Just double click on the root node ("Project") to open a completely empty **Window Editor**, where you can define the new window.

Creating new Menus

Since menus are a very special type of window, they cannot be drawn 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**'. (please refer to the chapter about the **Menu Editor** for more info).

Changing the Window Sorting

By default, the windows are displayed in alphabetical order according to their name. However, you can choose a different sorting by clicking the right mouse button and choosing "Sort by Type" or "Sort by QTP Class" in the context menu. The last option is only available if QuickTest has been selected as standard output format for the project.

2.4 Test Data Hierarchy

The 3rd tab in the workspace displays the project's data model. It contains the following types of objects:

- Data Set: Corresponds to a table in a database. A data set consists of 1 or more data fields.
- Data Records: Each data set has exactly 1 item labelled *_Records*. Double-clicking on it opens the **Data Records Editor**. A record corresponds to a row in a database.
- Data Field: Corresponds to a column in a database.
- Data Effects: Each data set has exactly 1 item labelled *_Effects*. Double-clicking on it opens the **Data Effects Editor**. Effects are the basis for test data generation with the CECIL method.
- Equivalence Classes: Each data field has exactly 1 item labelled *Equivalence_Classes*. Double-clicking on it opens the **Equivalence Class Editor**. Defining equivalence classes is important for test data generation with the method of the same name.

If the current project is a sub project, the data objects from its master project are displayed grey and marked as "READ-ONLY".

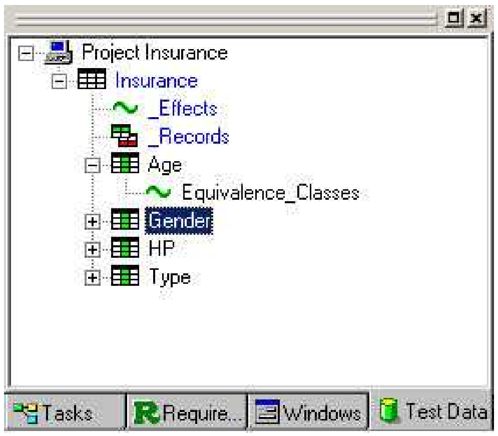


Figure 4 - Application Editor - Test Data Hierarchy

Selecting a Data Object

Simply select an object by clicking on its name in the tree. Several toolbar buttons become active that allow you to edit the properties of the object. The selection of the root node has no effect.

Adding a new Data Set

- Select the *Project* item at the root of the tree
- Press '**New Data Set**' in the toolbar or the menu. A new data set including an *_Effects* and a *_Records* subitem will be inserted.

Adding a new Data Field

- Select a data set in the tree
- Press '**New Data Field**' in the toolbar or the menu. A new data field will be inserted as a subitem of the data set.

Note: Instead of defining the data structure manually it is also possible to import it from an external file using the menu '**Project|Import|Test Data**'.

Editing a Data Object

Data sets, fields, effects, records, and equivalence classes can be edited by selecting them in the tree and pressing the "Edit Properties" tool bar button (or by using the context menu).

Deleting a Data Object

The selected object may be deleted by choosing the menu '**Delete**' or by pressing the corresponding toolbar button. An exception are the "_Effects", "_Records", and "Equivalence Classes" items that cannot be deleted. Confirm the appearing message box. ("Do you really want to delete this data object?")

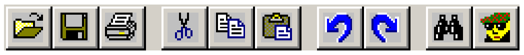
3 Toolbars and Menus

IDATG is a multi-document application, which means that there can be more than one editor window open at the same time. Most functions can be accessed both over the menu and the toolbar. To avoid erroneous user actions, all menus/buttons remain disabled until all conditions for their activation are fulfilled.

3.1 Toolbars

Several toolbars facilitate working with IDATG. Most buttons can be used in several sub-editors, which guarantees a consistent and easy workflow.

3.1.1 General Toolbar



The icons known from most Windows applications are (from left to right):
Project Open / Save / Print, Cut / Copy / Paste
The IDATG-specific icons are:

Undo Last Action

PURPOSE:

Reset the project to the state before the last user action

ACTIVATION CONDITION:

A project has been opened and something has been edited.

Redo Last Action

PURPOSE:

Set the project to the state before the last undo

ACTIVATION CONDITION:

The 'Undo' function has been used.

Search Object

PURPOSE:

Search the project for a specific window or task.

ACTIVATION CONDITION:

A project must be opened.

GUI Spy

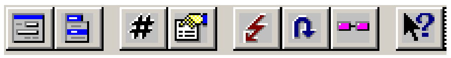
PURPOSE:

Open the **GUI Spy** to record windows directly from the screen.

ACTIVATION CONDITION:

A project must be opened.

3.1.2 Editor Toolbar



Edit Window

PURPOSE:

Open a **Window Editor** window.

ACTIVATION CONDITION:

A window is selected in any editor (exception: menus and menu items).

Edit Menu

Purpose:
Open a **Menu Editor** window.
Activation condition:
A menu or menu item is selected in any editor.

Edit Attributes

Purpose:
Open the **Attributes Editor**.
Activation condition:
A project must be opened.

Edit Properties

Purpose:
Open the **Window Properties Editor, Task Properties Editor, Step Editor, Data Set Editor** or **Data Field Editor** depending on the selected object.
Activation condition:
A window, task, step, data set or data field is selected in any editor.

Generate Test Cases

Purpose:
Generate test cases for the whole application or the selected object.
Activation condition:
A project must be opened.

Convert Test Cases

Purpose:
Convert test cases into tool-specific test scripts for the whole application or the selected object.
Activation condition:
A project must be opened.

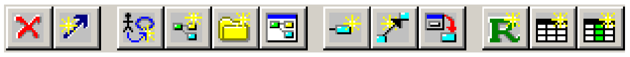
View Test Case

Purpose:
Open the **Test Case Editor** for the whole application or for the selected object.
Activation condition:
A project must be opened.

Help

Purpose:
Show context-sensitive help.
Activation condition:
None.

3.1.3 Draw Toolbar



Delete Selection

Purpose:
Delete the selected objects.
Activation condition:
Something is selected in any editor.

New Transition
PURPOSE:

Draw a new state transition in a **Window** or **Menu Editor** window.

ACTIVATION CONDITION:

A **Window** or **Menu Editor** window has the focus and is in transition mode.

New Task
PURPOSE:

Create a new use case task. The task is always inserted as a child of the currently selected folder.

ACTIVATION CONDITION:

The **Application Editor's** 'Tasks' tab is active and a folder is selected.

New Building Block
PURPOSE:

Create a new building block. The building block is always inserted as a child of the currently selected folder.

ACTIVATION CONDITION:

The **Application Editor's** 'Tasks' tab is active and a folder is selected.

New Folder
PURPOSE:

Create a new folder. The folder is always inserted as a child of the currently selected folder.

ACTIVATION CONDITION:

The **Application Editor's** 'Tasks' tab is active and a folder is selected.

Edit Task Flow
PURPOSE:

Open the **Task Flow Editor**.

ACTIVATION CONDITION:

A task must be selected in any editor.

New Step
PURPOSE:

Create a new task step or test step. If a step is selected the new step is added after this step, if a connection is selected the new step is added between the connected steps.

ACTIVATION CONDITION:

The **Task Flow** or **Test Case Editor** is active and a step or connection is selected.

Connect Steps

PURPOSE:

Connect two task steps.

ACTIVATION CONDITION:

The **Task Flow Editor** is active.

Assign Object to Step

PURPOSE:

Assign a task step or test step to a window or transition. In addition, task steps can also be assigned to tasks.

ACTIVATION CONDITION:

The **Task Flow** or **Test Case Editor** is active and a step is selected.

Edit Requirements

PURPOSE:

Open the **Requirement Editor**.

ACTIVATION CONDITION:

A project must be opened.

New Data Set

PURPOSE:

Create a new data set.

ACTIVATION CONDITION:

The **Application Editor's** 'Test Data' tab is active and the 'Project' node is selected.

New Data Field

PURPOSE:

Create a new data field.

ACTIVATION CONDITION:

The **Application Editor's** 'Test Data' tab is active and a data set is selected.

3.2 Menu

The IDATG menu has the following sub-menus:

3.2.1 Project

Menu Item	Purpose	Activation Condition
New	Create a new project	None
Open	Open an existing project	None
Save	Save all changes	A project must be opened
Save as	Save the current project under a different name	A project must be opened
Close	Close the current project	A project must be opened
Administration => Project Properties	Change the project settings	A project must be opened
Administration => Create Sub Project	Create a new project that is a sub project of the current one	A project must be opened that is not a sub project itself

Administration => Merge with Master Project	Merge all data from the current sub project with the data from its master project.	A sub project must be opened
Import => Test Data	Import test data from a text file into a data set	A project must be opened
Import => Project Data	Adds data from an XML file to the currently opened project.	A project must be opened
Import => Java-GUI Info	Import an .idj file that has been generated by IDATG's **Java GUI Spy**.	A project must be opened
Import from HP QuickTest => Import QTP Object Repository	Import GUI info from a .tsr file that has been recorded with HP QuickTest	A project must be opened
Import from HP QuickTest => Initialize QTP Object Repository Options	Configure HP QuickTest so that coordinate and size information is recorded for each window	A project must be opened
Import from HP QuickTest => Reset QTP Object Repository Options	Undo the HP QuickTest configuration changes performed by the previous function	A project must be opened
Print	Print the contents of the current editor. Note: To print the entire GUI specification, it is recommended to use the HTML/WinWord interface instead	The **Window** or **Menu Editor** is active
Print Preview	View the appearance of the print-out	The **Window** or **Menu Editor** is active
Print Setup	Configure your printer	None
Recent Files (1-6)	Open a file that was recently used	None
Exit	Quit IDATG	None

3.2.2 Edit

Menu Item	Purpose	Activation Condition
Undo	Reset the project to the state before the last user action. It is possible to undo the last 5 changes.	A project has been opened and something has been edited.
Redo	Set the project to the state before the last undo. It is possible to undo and redo the last 5 changes.	The 'Undo' function has been used.
Cut	Delete the selected objects into the clipboard	A task is selected OR 1 or more task steps are selected OR A window is selected OR An input field has the focus

Copy	Copy the selected objects into the clipboard	A task is selected OR 1 or more task steps are selected OR A window is selected OR An input field has the focus
Paste	Insert the contents of the clipboard	Something is on the clipboard and an appropriate destination object is selected OR An input field has the focus
Delete	Delete the selected objects	Something is selected in any editor
Change Window Type	Change the type of the selected window. Note that all type-specific properties of the window are no longer valid after the conversion!	A window is selected in any editor. Menus and menu items are excluded.
Replace Window	Redirect all references to the selected window to another window	A window is selected in any editor. Menus are excluded.
Set as Start Window	Mark the selected window as the application's start window	The **Application Editor** is active and a window is selected that (1) has no parent and (2) is enabled and visible
Search Object	Search the project for a specific window or task (depends on the active tab in the **Application Editor**)	A project must be opened
Search References	Search the project for references to a specific task, requirement, or window	A project must be opened

3.2.3 Draw

Menu Item	Purpose	Activation Condition
New Task	Create a new task. The task is always inserted as a child of the currently selected folder.	The **Application Editor's** 'Tasks' tab is active and a folder is selected.
New Building Block	Create a new building block. The building block is always inserted as a child of the currently selected folder.	The **Application Editor's** 'Tasks' tab is active and a folder is selected.
New Folder	Create a new folder. The folder is always inserted as a child of the currently selected folder.	The **Application Editor's** 'Tasks' tab is active and a folder is selected.
New Step	Create a new task step or test step. If a step is selected the new step is added after this step, if a connection is selected the new step is added between the connected steps.	The **Task Flow** or **Test Case Editor** is active and a step or connection is selected
Connect Steps	Connect two task steps	The **Task Flow Editor** is active
Assign Object to Step	Assign a task step or test step to a window or transition. In addition, task	The **Task Flow** or **Test Case Editor** is active and a step is

	steps can also be assigned to tasks.	selected
New Requirement	Create a new requirement. The requirement is always inserted as a child of the currently selected item.	The **Application Editor's** 'Requirements' tab is active and an item is selected.
New Data Set	Create a new data set	The **Application Editor's** 'Test Data' tab is active and the 'Project' node is selected.
New Data Field	Create a new data field	The **Application Editor's** 'Test Data' tab is active and a data set is selected.
New Menu	Create a new menu. If a window is selected, the menu is added as its child.	The **Application** or **Window Editor** is active and the selected window doesn't have a menu
New Menu Item	Create a new menu item. If another menu item is selected, the new item is added as its child.	The **Menu Editor** is active
New Transition	Draw a new state transition	A **Window** or **Menu Editor** window has the focus and is in transition mode
Redirect Transition	Change the start and/or destination state of a transition	A **Window** or **Menu Editor** window has the focus and a transition is selected

3.2.4 View

Menu Item	Purpose	Activation Condition
Workspace	Show/hide the application bar containing the **Application Editor**.	None
Tool-Bars => General Tool Bar	Show/hide the tool bar that contains the general functions (e.g. Open Project)	None
Tool-Bars => Editor Tool Bar	Show/hide the tool bar that controls the IDATG editors	None
Tool-Bars => Draw Tool Bar	Show/hide the tool bar that contains the drawing functions	None
Tool-Bars => Window Tool Bar	Show/hide the tool bar for window type selection	A **Window Editor** is active
Tool-Bars => Status Bar	Show/hide the status bar at the bottom of the main frame.	None
Show Step Info	Show/hide step information when moving the mouse over a step in the **Task Flow** or **Test Case Editor**	None
Show Grandchildren	Show/hide all successors of a window in the **Window Editor**. If disabled, only the direct children are shown.	None

Show Transitions	Switch all **Window** and **Menu Editors** to transition mode, i.e. all transitions are shown as blue arrows	A project must be opened
Show Coverage	Switch all **Window** and **Menu Editors** to coverage mode, i.e. transitions covered by test cases are shown as green arrows, others as dark gray arrows	A project must be opened
Language => English	Switches to the English version of the IDATG user interface. A restart of the tool is required.	A different language is currently selected
Language => German	Switches to the German version of the IDATG user interface. A restart of the tool is required.	A different language is currently selected

3.2.5 Window

Menu Item	Purpose	Activation Condition
Cascade	Arrange all editors in an overlapping style	At least 1 editor is open
Tile	Arrange all editors one beside the other	At least 1 editor is open
Arrange Icons	Arrange all minimized editors one beside the other	At least 1 editor is open
GUI Spy	Open the **GUI Spy**	A project must be opened
Window Wizard	Open the **Window Wizard** to create a standard window (e.g. Message Box)	A project must be opened
Window Editor	Open a **Window Editor** window	A window is selected in any editor (exception: menus and menu items)
Menu Editor	Open a **Menu Editor** window	A menu or menu item is selected in any editor
Task Flow Editor	Open the **Task Flow Editor**	A task must be selected in any editor
Attributes Editor	Open the **Attributes Editor**. If a window is selected, the window attributes are edited, else the global attributes.	A project must be opened
Properties Editor	Open the appropriate editor for the selected object.	A window, task, step, requirement, data set, data field, or equivalence class is selected in any editor.
User-defined Fields Editor	Open the **User-defined Fields Editor**	A project must be opened
Meta-Condition Editor	Open the **Meta-Condition Editor**	A project must be opened

GUI Builder Editor	Open the **GUI Builder Editor**	A project must be opened

3.2.6 Generate

Menu Item	Purpose	Activation Condition
Generate Test Cases	Generate test cases for the selected object	A project must be opened
Convert Test Cases	Convert test cases into tool-specific test scripts for selected object	A project must be opened
View Test Cases	Open the **Test Case Editor** for the selected object	A project must be opened
Delete All Test Cases	Delete all test cases of this project	A project must be opened
Generate Data Records	Generate data records for the selected data set	A data set or one of its underlying tree items is selected
Generate Window Tags	Generate window tags for the selected window and its children	A project must be opened
Generate HTML Documentation	Export the GUI specification as HTML files	A project must be opened

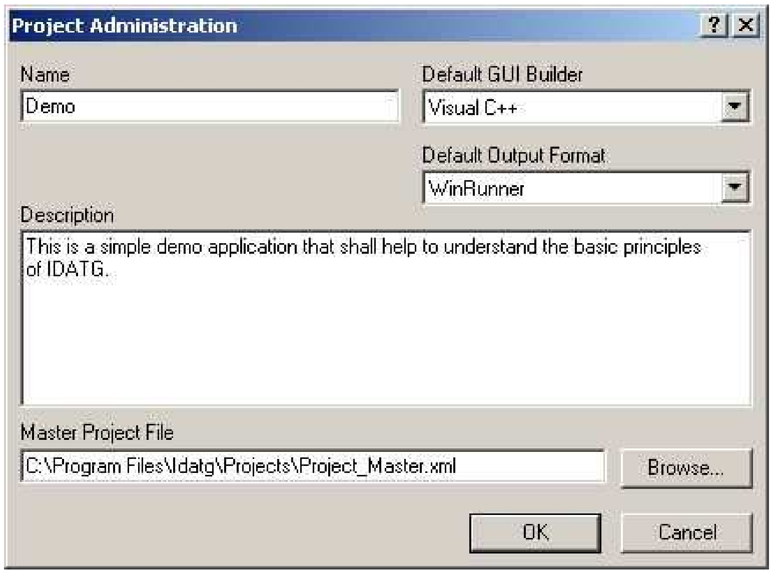
3.2.7 Help

Menu Item	Purpose	Activation Condition
Help Topics	Call the IDATG online help	None
About IDATG...	View version no. of IDATG and copyright information	None

4 Project Administration

4.1 Project Administration Dialog

After creating a new project or clicking on the menu ‘**Project | Administration | Project Properties**’ the **Project Administration Dialog** appears.



Name: Demo
Default GUI Builder: Visual C++
Default Output Format: WinRunner
Description: This is a simple demo application that shall help to understand the basic principles of IDATG.
Master Project File: C:\Program Files\Idatg\Projects\Project_Master.xml

Figure 5 - Project Administration Dialog

Changing the project name

You can enter a new project name in the field ‘**Project**’. The name must be unique and may be up to 30 characters long.

Defining the default GUI builder

Select the principal GUI builder with that your application has been created. If you are later adding new windows in IDATG, they will automatically get the correct classnames. If your GUI builder doesn´t appear in the list, you can define it with the **GUI Builder Editor**.

Defining the default output format

Apart from exporting test cases in plain text format, IDATG can generate test scripts for specific test execution tools. If you are later adding new windows in IDATG, they will automatically get the correct tags to ensure that they can be recognized by the test tool. The tool selection may be changed later on, but in this case don´t forget to generate new tags for the existing windows using the **Window Tag Generator**!

Entering a description

An arbitrary description of the project may be entered.

Changing the path to the master project

You can use the '**Browse**' button to change the path to the master project file. Note that this is only recommended if the master file has been renamed or moved. Setting the path to a completely different file will usually result in serious data inconsistencies. See also User's Guide Part I for information on creating and managing sub projects.

4.2 Project Open Dialog

IDATG projects are saved as XML files. In case a project consists of more than one XML file, you have to open the projects's main file (usually called "Project_*Name*.xml"). All other files belonging to the project can usually be found in a separate directory (called "Data_Project_*Name*") and should not be opened directly with IDATG.

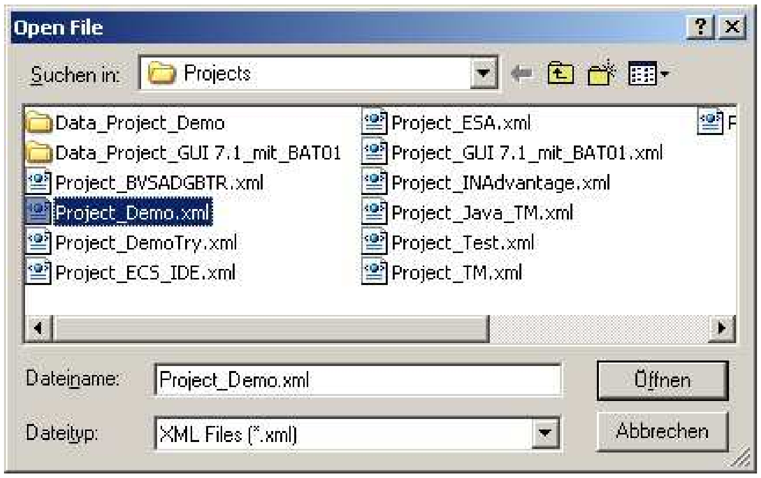
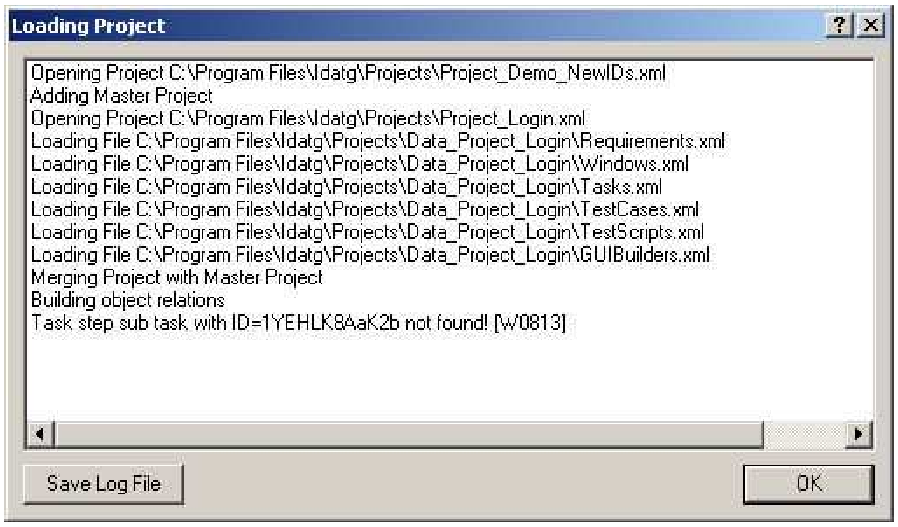


Figure 6 - Project Open Dialog

After selecting a project file a status window informs about the loading process. In general, it disappears automatically as soon as loading has completed. However, it stays open if errors have occurred to give the user the chance to read the messages and save them to a log file. The displayed text can be saved to a file by pressing '**Save Log File**'. The file is named "*Logfile.txt*" and is placed in the *My Documents\Idatg\Logs* directory.

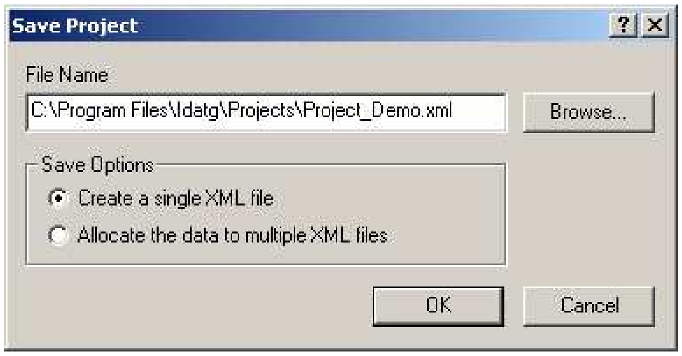


```
Opening Project C:\Program Files\Idatg\Projects\Project_Demo_NewIDs.xml
Adding Master Project
Opening Project C:\Program Files\Idatg\Projects\Project_Login.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\Requirements.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\Windows.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\Tasks.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\TestCases.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\TestScripts.xml
Loading File C:\Program Files\Idatg\Projects\Data_Project_Login\GUIBuilders.xml
Merging Project with Master Project
Building object relations
Task step sub task with ID=1YEHLK8AaK2b not found! [W0813]
```

Figure 7 - Loading Status Dialog

4.3 Project Save Dialog

IDATG offers 2 different methods for saving your project in XML format: You can either save all data to a **single XML file** or allocate the data to **multiple XML files**. The first option is recommended for smaller projects because it facilitates configuration management. However, for larger projects the second option may help to reduce loading/saving times and to keep a better overview of the data structure.

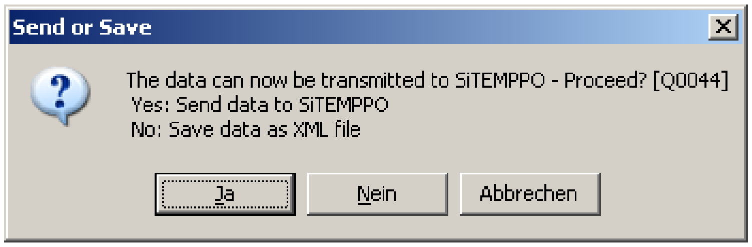


File Name: C:\Program Files\Idatg\Projects\Project_Demo.xml

Save Options
- [✓ Create a single XML file]
- Allocate the data to multiple XML files

Figure 8 - Project Save Dialog

If IDATG was called from TEMPPO Test Manager, the following Project Save Dialog will be shown:



The data can now be transmitted to SiTEMPPO - Proceed? [Q0044]
Yes: Send data to SiTEMPPO
No: Save data as XML file

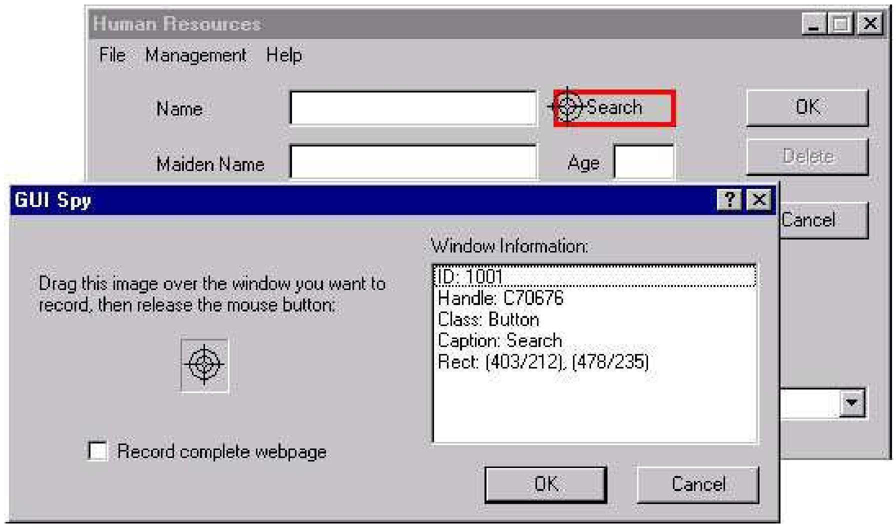
Figure 9 - Project Send or Save Dialog

If you confirm this dialog with “Yes” the data will be transmitted to TEMPPO Test Manager and IDATG will be closed. If you confirm this dialog with “No” the upper “Project Save Dialog” will be offered as next step and the IDATG project will be kept open.

5 Interfaces

5.1 GUI Spy

The **GUI Spy** is a very efficient tool that makes even the specification of complex windows dead easy. No resource scripts or other source code is required, GUI information can be recorded directly from the screen. The method is the same as known from other GUI Spies like those contained in the Microsoft Developer Studio or in capture/replay tools.



```
ID: 1001
Handle: C70676
Class: Button
Caption: Search
Rect: (403/212), (478/235)
```

Figure 10 - GUI Spy and recorded window

Which windows can be recorded?

- All windows that are administered by the MS Windows operating system can be recognized by the **GUI Spy**. This covers all windows created with Visual C++, Visual Basic, .NET, Delphi and many other GUI builders.
- In addition, HTML pages can be recorded directly from MS Internet Explorer (IE Version 5.5 and up). We are proud to say that it is **not** necessary that you open the IE after IDATG like it is required by some test execution tools. Due to a very sophisticated COM technique it is also possible to record web pages containing frames and even hotspots inside an image. IE windows that are embedded in other applications (e.g. online help) are not supported, please open the desired page with the ordinary IE instead.
- Some GUI builders like Java have a window management of their own. Therefore, these objects are not visible to the operating system and the **GUI Spy**. For recording Java windows, a special **Java Interface** is available that has to be used instead of the **GUI Spy** (see 5.4).

How is the window type determined?

- For most standard window classes the window type is determined using the classname.
- Some classnames like *BUTTON* are used for more than one window type (push button, check box, radio button etc.). In this case other window properties have to be analyzed to determine the actual type.
- Self-defined classes that are derived from standard classes are usually recognized correctly even if they have a different classname (using a quite sophisticated mechanism).
- For other self-defined classes, it is recommended that you specify their classnames in the **Window Class Editor** before using the **GUI Spy** to guarantee that the window type can be

correctly recognized. If the **GUI Spy** is unable to determine the type, it creates a Custom Window.

- The type of a recorded window can also be changed afterwards using the menu '**Edit|Change Window Type**'.

Recording a window

- Call your application (a GUI prototype is sufficient) and open the window you wish to record. For webpages, open the IE and navigate to the desired page.
- Open the **GUI Spy**. While it is active, all other IDATG windows are hidden in order to facilitate recording.
- Click on the cross-hair symbol in the **GUI Spy** and **keep the mouse button pressed.**
- Drag the mouse cursor over the screen. Note that the window under the cursor is highlighted with a red frame and that information about that window is displayed in the list box.
- If you have reached the window you wish to record, release the mouse button. Note that exactly the window under the cursor and its successors are recorded, but not the parent.
- You may change your selection by dragging the image another time or confirm it by pressing **'OK'**. In this case, the new window is added to the IDATG project and a **Window Editor** opens for it.

Recording a complete webpage

- If you wish to record all HTML objects in a webpage, just select '**Record complete webpage**' before dragging the mouse over the IE window.

Displayed Information

The following window information is displayed in the list while recording:

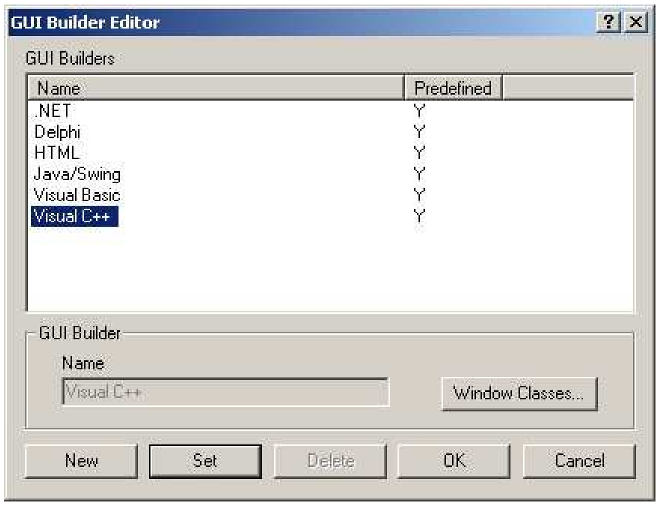
- Numeric window ID (decimal, can only be used as tag for windows that have a parent)
- Window handle (hexadecimal)
- Class name
- Caption (may be empty)
- Coordinates measured from the upper left corner of the screen (left/top), (right/bottom). May be negative, if the upper left corner is not positioned in the visible area of the screen.

For HTML windows, the following information is provided:

- HTML Tag (type of window, e.g. *IMG*)
- Class (class expected by WinRunner, e.g. *html_rect*)
- Caption (the displayed text or the object name defined in the HTML source code)
- Coordinates measured from the upper left corner of the IE client window (left/top), (right/bottom).

5.2 GUI Builder Editor

The **GUI Builder Editor** can be opened with the menu ‘**Window|GUI Builder Editor**’. It contains a list of pre-defined GUI builders and also allows you to specify additional ones. GUI builder information is used by the **GUI Spy** to determine the correct type of the recorded windows. Thus, it is recommended to specify the window classes of the used GUI builder <u>before</u> using the **GUI Spy**.



Name	Predefined
.NET	Y
Delphi	Y
HTML	Y
Java/Swing	Y
Visual Basic	Y
Visual C++	Y

Name: Visual C++

Figure 11 - GUI Builder Editor

Note that the pre-defined GUI builders cannot be renamed or deleted, but it is possible to add new window classes to them.

Creating a New GUI Builder

- Press ‘**New**’
  Enter the unique name of the new GUI builder. It may only contain alphanumeric characters and ‘_’ and may be up to 30 characters long.
- Press ‘**Window Classes...**’ to specify the windows of the new GUI builder with the **Window Class Editor**.
- Press ‘**Set**’ ➔ it is checked whether the name is unique. If not, a message box appears and the focus moves to the invalid field.

Selecting a GUI Builder

- Click on a GUI builder in the list box ➔ Its name is shown in the field.

Editing a GUI Builder

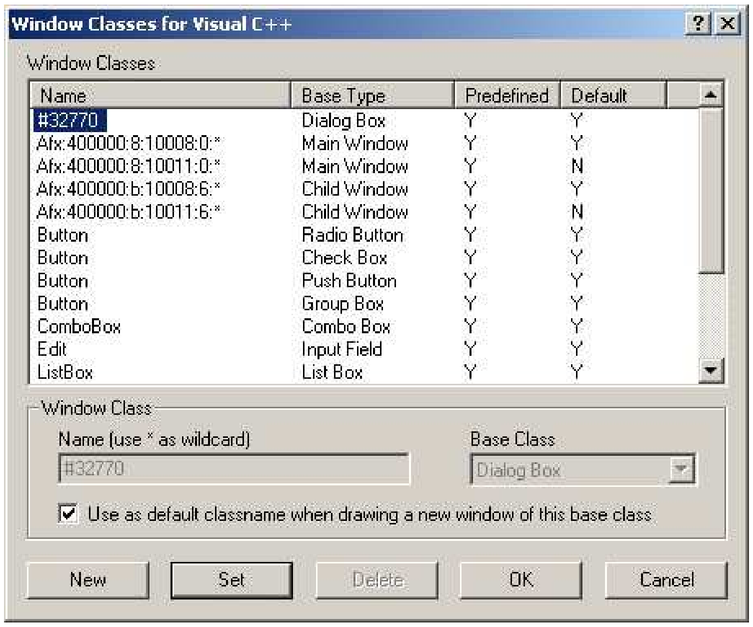
- Select a GUI builder in the list box
- The name can only be edited, if the GUI builder is not pre-defined. The **Window Class Editor** can be opened for all GUI builders using ‘**Window Classes...**’.
- Press ‘**Set**’ ➔ it is checked whether the name is unique. If not, a message box appears and the focus moves to the invalid field.

Deleting a GUI Builder

- Select a self-defined GUI builder in the list box. Pre-defined GUI builders cannot be deleted.
- Press ‘**Delete**’ ➔ a message box appears that has to be confirmed by the user. (”Do you really want to delete this GUI builder ?“)

5.3 Window Class Editor

The **Window Class Editor** is a part of the **GUI Builder Editor** described in the previous chapter. It can be reached by selecting a GUI builder and pressing '**Window Classes...**'. If a pre-defined GUI builder has been selected, the editor displays a set of pre-defined window classes for it that cannot be edited. However, it is possible to add new classes.



Name	Base Type	Predefined	Default
#32770	Dialog Box	Y	Y
Afx:400000:8:10008:0:*	Main Window	Y	Y
Afx:400000:8:10011:0:*	Main Window	Y	N
Afx:400000:b:10008:6:*	Child Window	Y	Y
Afx:400000:b:10011:6:*	Child Window	Y	N
Button	Radio Button	Y	Y
Button	Check Box	Y	Y
Button	Push Button	Y	Y
Button	Group Box	Y	Y
ComboBox	Combo Box	Y	Y
Edit	Input Field	Y	Y
ListBox	List Box	Y	Y

Window Class
Name (use * as wildcard): #32770
Base Class: Dialog Box
☑ Use as default classname when drawing a new window of this base class

Figure 12 - Window Class Editor

When recording a window from the screen with the **GUI Spy**, IDATG compares the classname of the new window with the names that are defined in the window class database. If the name is found, IDATG is able to create a window of the correct type, otherwise a Custom Window is created. For example, if IDATG records a window with the classname '#32770', the window class database provides the information that this window is a Visual C++ Dialog Box and makes IDATG create a Dialog Box. To ensure that this also works for your self-defined window classes, you should define them in the **Window Class Editor** <u>before</u> using the **GUI Spy**.

Using Wildcards (*):
Some classnames are changing dynamically or depend on the version of the GUI builder. For example, Visual Basic check boxes are named *ThunderRT4CheckBox*, *ThunderRT5CheckBox* etc. In this case it is easier to specify the name using a wildcard: *ThunderRT*CheckBox*. The wildcard * stands for an arbitrary string (that can also be empty) and may be used anywhere in the classname. If you are using wildcards, you have to take care that no ambiguities are possible. For instance, *ThunderRT** would match both *ThunderRT5CheckBox* and *ThunderRT5List Box*.

Although self-defined classnames have to be unique, you may notice that the classname *Button* is used for different window types in Visual C++. The cause for this exception is the dubious design of the MFC classes that use the same name for completely different window types (Check Box, Push Button, Radio Button, Group Box (!)). The IDATG **GUI Spy** treats these windows separately and solves the problem by analyzing the button style.

Creating a New Window Class

- Press ‘**New**’
  Enter the name of the new class and select the IDATG base class that best matches the behavior of it. The name may contain arbitrary characters and be up to 50 characters long. ‘*’ can be used as a wildcard.
- If you wish that this classname should be used for new windows of this base class drawn with the **Window Editor**, check the corresponding box. For instance, if you have entered a new class *MyDialog* and assigned it the base class Dialog Box, every time you draw a new Dialog Box in the **Window Editor** it gets the classname *MyDialog*. This is only necessary if there is more than one class of the same base type. Note that if another class of the same base type has been set as default earlier, its default flag is cleared automatically.
- Press ‘**Set**’ ➔ it is checked whether the name is unique. If not, a message box appears and the focus moves to the invalid field.

Selecting a Window Class

- Click on a window class in the list box ➔ Its properties are shown in the fields.

Editing a Window Class

- Select a window class in the list box
- The name and base type can only be edited, if the class is not pre-defined. The default flag may always be edited.
- Press ‘**Set**’ ➔ it is checked whether the name is unique. If not, a message box appears and the focus moves to the invalid field.

Deleting a Window Class

- Select a self-defined window class in the list box. Pre-defined classes cannot be deleted.
- Press ‘**Delete**’ ➔ a message box appears that has to be confirmed by the user. (“Do you really want to delete this window class ?“)

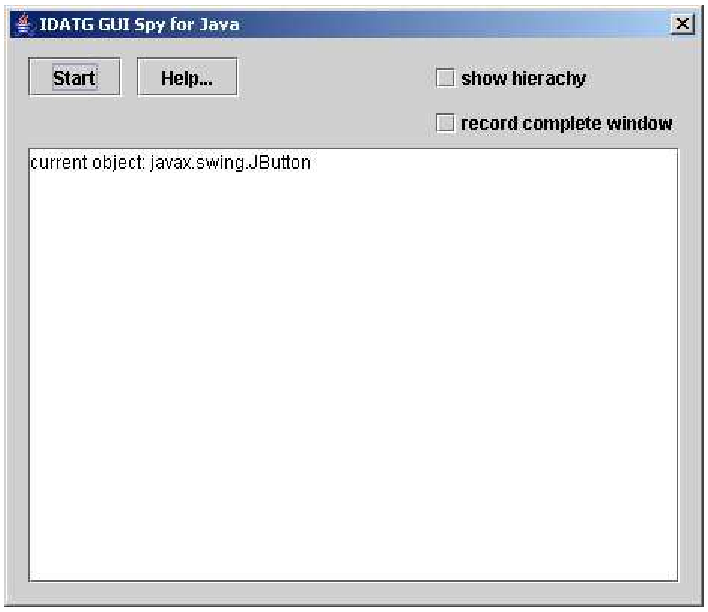
5.4 Java Interface

Since Java objects are invisible to the normal GUI Spy, a special AddOn is required that allows you to import information about Java GUI´s. IDATG supports Java applications as well as Java applets that run in a browser like Netscape or Internet Explorer.

5.4.1 Java Applications

The following steps are necessary to record windows of a Java application:

1. Copy the files *GUISpy.bat* and *GUISpyApplication.jar* into the working directory of your Java application.
2. Set the following parameters in *GUISpy.bat*:
   JavaHome: The full path to your JRE (e.g., *"C:\Program Files\JavaSoft\jre1.4\bin"*)
   AppClasspath: Classpath for the Java application
   AppMain: Start class of the Java application (e.g., *foo.MyApp*)
   *AppParameters: C*ommand line parameters for the Java application (optional)
3. Call *GUISpy.bat* which will open your Java application as well as the IDATG GUI Spy window
4. Click '**Start**' and move the mouse cursor over the screen. You will notice that information about the window under the cursor will be displayed inside the GUI Spy. If you select '**Show Hierarchy**', additional information about the children will be displayed.
5. Hold the Ctrl key and left click on the window you wish to record. A file save dialog opens and lets you specify name and path of the recorded file. The default extension is *.idj* (IDATG Java). Press '**Save**'.
6. If you encounter problems while recording a dialog because its entire client area is filled with child windows, you can check the option '**Record complete window**'. After this, a click on an arbitrary child window will record the entire dialog.
7. The generated file can be imported into IDATG (see 5.4.3). After this step, the behavior of the Java GUI can be specified in the same way as for other GUI builders.



current object: javax.swing.JButton

Figure 13 - GUI Spy for Java Applications

5.4.2 Java Applets

The following steps are necessary to record windows of a Java applet:

1. Edit the HTML code that calls your Java applet. Insert a call to IDATG´s Java GUI Spy applet that comes with the AddOn (*GUISpyApplet.jar*).
2. Change the security restrictions of your JRE (Java Runtime Environment) to allow the applet to write to your hard disk.
3. Open the HTML page containing the two applets in your browser. Since the Java GUI Spy applet is running in the same browser as the tested one, it has access to its methods and can record information about its GUI. The information is saved to a .idj ("IDATG Java") file.
4. The generated file can be imported into IDATG. After this step, the behavior of the Java GUI can be specified in the same way as for other GUI builders.

Adding the GUI Spy Applet to a HTML page

In order to be able to access the applet under test, IDATG´s GUI Spy Applet has to run in the same Java Virtual Machine. The following lines have to be added to your HTML code to call the GUI Spy Applet:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="465" height="85" codebase=" http://java.sun.com/products/plugin/1.2/jinstall-12-
win32.cab#Version=1,2,0,0">
<PARAM name="code" value="GUISpy.class">
<PARAM name="codebase" value="<YOURCODEBASE/INSTALLATIONPATH>">
<PARAM name="archive" value="GUISpyApplet.jar">
<PARAM name="type" value="application/x-java-applet;version=1.2">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2" code="GUISpy.class"
codebase="<YOURCODEBASE/INSTALLATIONPATH>" archive="GUISpyApplet.jar" width="465" height="85"
pluginspage=" http://java.sun.com/products/plugin/1.2/jinstall-12-
win32.cab#Version=1,2,0,0"><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>
```

Please replace *YOURCODEBASE* and *INSTALLATIONPATH* with the values that match the configuration on your system.

The same code has to be used to call the applet under test. Please modify the **bold** sections accordingly. Additional parameters that shall be passed to your applet can be specified in the *<PARAM>* section.
If the required plug-in is not already installed on your computer, it is downloaded automatically from the SUN homepage the first time you view the HTML page in your browser (Please make sure you are connected to the internet and follow the instructions on the screen).

Changing the security restrictions of the JRE

The GUI Spy applet saves the recorded information as *.idj* files. However, Java´s default security restrictions do not permit applets to write to the hard disk. To change the settings, add the following statement to your *java.policy* file located in the *lib/security* subfolder of your JRE:

```
grant {
  permission java.io.FilePermission "<<ALL FILES>>", "read, write";
  permission java.util.PropertyPermission "user.dir", "read";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.lang.RuntimePermission "modifyThread";
};
```

Warning: Use these settings only while recording your applet and make sure to remove the statement before accessing other webpages! Otherwise any applet from the internet would have the right to access your hard disk!!

Recording a Java GUI

- Open the HTML page containing both applets in a browser.
- If the page contains other applets as well, select the one you wish to record using the combo box '**Selected applet:**' in the GUI Spy applet.
- If you want to record the applet´s main window:
  - Check '**Record applet main window**' and press '**Record GUI**'.
- To record any other window:
  - Open the window you wish to record in the tested applet.
  - Click '**Record GUI**' in the GUI Spy applet. (Do not leave the mouse button pressed like in IDATG´s normal GUI Spy).
  - Move the mouse cursor over the screen. Note that the window under the cursor is highlighted with a red frame. Windows that do not belong to the selected applet are ignored.
  - Click on the window you wish to record. Like in the other **GUI Spy**, the window and all of its children are recorded. Therefore, clicking on a button inside a dialog only records the button but not the rest of the dialog. To record the entire dialog, click on its title bar.
- A file save dialog opens and lets you specify name and path of the recorded file. The default extension is *.idj* (IDATG Java). Press '**Save**'.

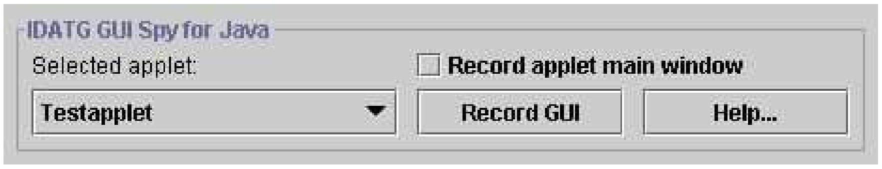


Figure 14 - Java GUI Spy Applet

5.4.3 Importing an .idj file into IDATG

After calling IDATG.EXE, open a project and choose '**Import | Java GUI Info**' from the '**Project**' menu. Select an *.idj* file and press '**Open**'. The file is imported into IDATG and the windows contained within are displayed in the **Application Editor**. After this step, the behavior of the Java GUI can be specified in the same way as for other GUI builders.

Note that Java classes that are not relevant for the specification are ignored by the GUI Spy to decrease the GUI´s complexity (e.g., most JPanes).

5.5 Import from HP QuickTest Object Repositories

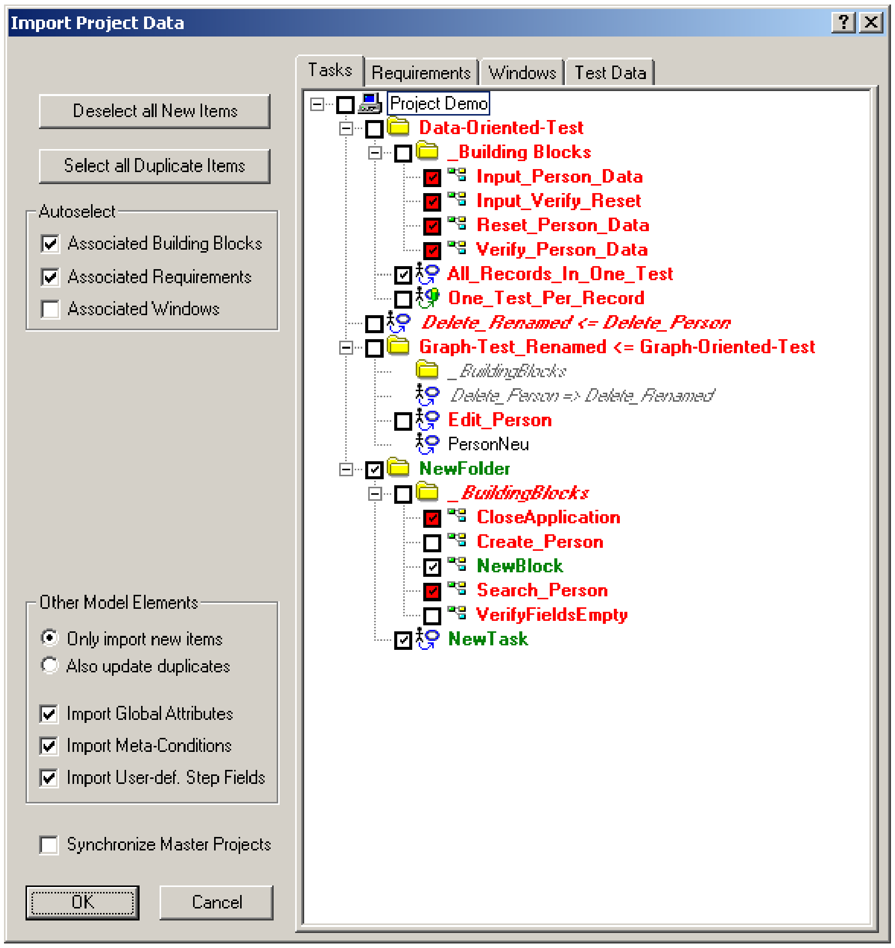
As an alternative to using the built-in GUI-Spy of IDATG, GUI info can be imported directly into IDATG from Object Repositories recorded with HP QuickTest Professional. For detailed information, see User's Guide Part I.

5.6 Project Data Import

IDATG stores all project data in convenient XML files. Thus, exchanging data between IDATG projects or even with other applications is quite straightforward. For instance, it might be useful to import a list of requirements from an external source or to re-use tasks from a different project.

5.6.1 Exchanging Data between IDATG Projects

If you wish to copy parts of an IDATG project into another project, you can use IDATG's powerful Import function. Just use the menu '**Project | Import | Project Data**' and select the XML file you wish to import. A dialog is displayed that shows a tree hierarchy of the existing items in the original project ("project 1") and the new / duplicate items in the imported project ("project 2"). As in the IDATG workspace, there are 4 tabs labelled Tasks, Requirements, Windows and Test Data.



Autoselect
- [✓] Associated Building Blocks
- [✓] Associated Requirements
- [ ] Associated Windows

Other Model Elements
- (●) Only import new items
- ( ) Also update duplicates
- [✓] Import Global Attributes
- [✓] Import Meta-Conditions
- [✓] Import User-def. Step Fields

[ ] Synchronize Master Projects

- [ ] Project Demo
  - [ ] Data-Oriented-Test
    - [ ] _Building Blocks
      - [✓] Input_Person_Data
      - [✓] Input_Verify_Reset
      - [✓] Reset_Person_Data
      - [✓] Verify_Person_Data
    - [✓] All_Records_In_One_Test
    - [ ] One_Test_Per_Record
  - [ ] *Delete_Renamed <= Delete_Person*
  - [ ] Graph-Test_Renamed <= Graph-Oriented-Test
    - *_BuildingBlocks*
    - *Delete_Person => Delete_Renamed*
    - [ ] Edit_Person
    - PersonNeu
  - [✓] NewFolder
    - [ ] *_BuildingBlocks*
      - [✓] CloseApplication
      - [ ] Create_Person
      - [✓] NewBlock
      - [✓] Search_Person
      - [ ] VerifyFieldsEmpty
    - [✓] NewTask

Figure 15 - Project Data Import Dialog

5.6.1.1 Graphical Representation

The items are displayed according to the following rules:

- Items existing in project 2 are displayed in **bold** letters. A checkbox is displayed near each project 2 item that allows the user to select it for import.
- Unchanged items existing in project 1 but not in project 2 are displayed in black, normal letters. No checkbox is displayed near project 1 items.
- New items existing in project 2 but not in project 1 are displayed in **green** letters.
- Duplicate items both existing in project 1 and project 2 are identified by their internal ID that is only visible in the XML file. There are two possibilites:
  a. The items have the same parent folder => item is displayed only once, in **red** letters. If the item has been renamed, the label "**new name <= old name**" is used.
  b. The items have different parent folders => the project 1 item is displayed in *grey italic* letters, the project 2 item in ***red italic*** letters. If the item has been renamed, the labels "*old name => new name*" and "***new name <= old name***" are used. If the item is a folder containing children, they are only displayed below the project 2 item (otherwise the children would appear twice in the hierarchy).
- **Master and sub projects**: If project 1 or 2 are sub projects, the items of their master projects are not displayed. However, master project tasks and requirements will be considered during the import if the checkbox '**Synchronize Master Projects**' is checked. The box is only displayed if both projects 1 and 2 have master projects and these master projects have different file paths.

  In the special case that project 2 itself contains duplicate items of project 1's master project, they are not displayed and cannot be imported. Otherwise severe inconsistencies would be produced.

5.6.1.2 Selecting Items for Import

The user can select the items that shall be imported from project 2 into project 1. Beside each tree item, a checkbox is displayed that shows its current selection state. The following rules apply:

- Only items from project 2 can be selected.
- The user can use the button '**Select All New Items**' to select all **green** items in the current tab. Once pressed, the button label changes to '**Deselect All New Items**' which gives the user the opportunity to reset his selection.
- The user can use the button '**Select All Duplicate Items**' to select all **red** items in the current tab. Once pressed, the button label changes to '**Deselect All Duplicate Items**' which gives the user the opportunity to reset his selection.
- Default settings are: all new items are selected, all others deselected.
- The user can also select individal items from the tree.
  a. If a folder/parent item is (de-)selected, all of its children are (de-)selected as well.
  b. If a **green** item in a **green** folder is selected, the folder is automatically selected as well (otherwise it would be unclear in which folder to put the new item).

c. In all other cases, the folder is not automatically selected. This gives the user the opportunity to update the item while leaving the folder hierarchy untouched.

Tab 'Tasks'

- If the user marks the checkbox '**Autoselect - Associated Building Blocks**', all building blocks that are referenced by the currently selected tasks are selected automatically as well (recursively). This avoids that important information is lost during the import.
  Caution: while this checkbox is marked, it is impossible to deselect building blocks that are referenced by another selected task. This locked selection state is illustrated by a red checkbox ☑ near the referenced building block.
- If the user marks the checkbox '**Autoselect - Associated Requirements**', all requirements that are associated with the currently selected tasks are selected automatically in the Requirements tab. This avoids that coverage information is lost during the import.
  Caution: while this checkbox is marked, it is impossible to deselect requirements that are referenced by a selected task. This locked selection state is illustrated by a red checkbox ☑ near the referenced requirement.
- If the user marks the checkbox '**Autoselect - Associated Windows**', all windows that are used in the currently selected tasks as start or destination window of a step are selected automatically in the Windows tab.
  Caution: while this checkbox is marked, it is impossible to deselect windows that are referenced by a selected task. This locked selection state is illustrated by a red checkbox ☑ near the referenced window.

Tab 'Requirements'

- A radio group '**Identify Duplicates by…**' with the choices '**XML ID**' and '**User ID**' is displayed. If '**XML ID**' is chosen, the internal ID that is only visible in the XML file is used for identifying duplicate requirements. If '**User ID**' is chosen, the requirement ID defined by the user in the **Requirements Editor** is taken instead.
  Caution: Changing this setting causes the requirements tree to be redrawn and resets all previous selections in it.

Tab 'Windows'

- No additional options are displayed in this tab.

Tab 'Test Data'

- A radio group '**Duplicate Data Sets**' is displayed and allows the user to specify what should happen to data sets that exist in both projects. If '**Only import new records**' is chosen, the data set structure (field types, equivalence classes etc.) in project 1 remains unchanged and only new data records are imported to existing sets. If '**Replace entire data set**' is chosen, the project 1 version of the data set is completely deleted and replaced by the version in project 2.

Select other Model Elements for Import

Since various auxiliary model elements can be referenced inside a task, it can become necessary to import these elements as well. This concerns:

- Global Attributes
- Meta-Conditions
- User-defined Step Fields

The import dialog provides a check box for each of these element types. If checked, all new elements of this type will be imported (option '**Only import new items**'). In addition, if the option '**Also update duplicates**' is selected, existing items will be overwritten.

5.6.1.3 Performing the Import

After pressing '**OK**', the dialog closes. Several rules and special cases have to be taken into account during the import:

- In order to avoid problems with outdated information, all existing test cases have to be deleted (you can generate them again later). Also, all editors that might be open at that time are closed automatically.
- If the name of a duplicate (**red**) item has changed and the name is used as a designator, it is updated in all affected expressions. For instance, if the name of a building block has changed, the name is automatically updated in all conditions etc. in which the name appears as designator. (e.g., *#@OldBBName:Parameter#* is changed to *#@NewBBName:Parameter#*)
- If a project 1 item and a project 2 item have the same name, but different XML-IDs, a number is appended to the name of the project 2 item (e.g., *MyTask* is changed to *MyTask1*). If the item name is used as designator in project 2, the name is updated in all affected expressions.
- **Master Projects:** The option '**Synchronize Master Projects**' is reserved for advanced users only. The original master project of project 1 will be changed, so it should only be used if you know exactly what you're doing! The following will happen:
  - It is determined which tasks and requirements in project 2's master project (MP2) are referenced by the selected tasks in project 2 (recursively). Windows, data sets, and other items are not considered by the function.
  - It is checked whether these tasks and requirements already exist in project 1's master project (MP1).
  - Only new tasks and requirements are imported from MP2 into MP1. Existing tasks and requirements are left unchanged.
  - Caution: It is possible that project 2 tasks are not compatible with the building block versions in MP1. On the other hand, it is possible that project 1 tasks are not compatible with the building block versions in MP2. There is no automated resolution possible for this conflict. If necessary, please adapt the affected tasks by hand or open MP1 and explicitly import the desired building blocks from MP2.

During the import, a message dialog informs the user about the progress. The name of each imported or updated item is displayed. After the import is completed, the user has the possibility to save the message log to a text file.

5.6.2 Exchanging Data with other Applications

In order to import data from an external source into IDATG, you only have to implement a simple adapter that converts your data into the XML scheme expected by IDATG. The current XML scheme description *IdatgData.xsd* can be found in the *Idatg\Projects* directory. If required, there is also a detailed specification of the various XML elements available from the IDATG support. Once the data has been converted, it can be imported to your IDATG project in the usual way by using the Import function.

Of course, the data exchange also works in the other direction. Since the data in IDATG's XML files are not encrypted, you can extract any kind of information you require. For instance, it is possible to get information about the generated test cases or the GUI of a certain project.

5.7 Test Data Import

For data-oriented testing, IDATG provides a possibility for importing test data from an external file. It can be accessed using the menu ‘**Project | Import | Test Data**’.

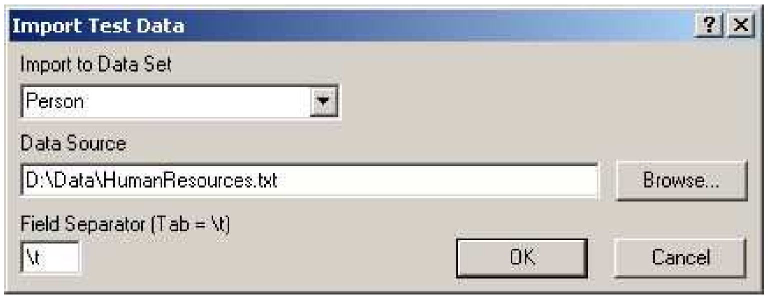


Figure 16 - Test Data Import Dialog

- First, the name of a data set has to be specified. The user may either select an existing data set from the list or enter a new name. The name may only contain alphanumeric characters and ‘_’ and be up to 30 characters long. If a specific data set had been selected before opening the dialog, it is used as default value.
- The file from which the data shall be imported can be selected using the '**Browse…**' button.
- The character combination that is used to separate one field from the next in the file can be specified as '**Field Separator**'. Typical values are the tabulator character (represented by \t), colon and semi-colon.

Supported File Formats

The import mechanism can be used for text files, but not for binary files. However, most database systems support exporting data in text format. Frequently used text formats are "tabulator-separated" and .csv (comma-separated values, in many countries it's actually a semicolon).

IDATG expects that the first line in the file contains the field (column) names. The data fields are created automatically if they are not yet existing. For each other line, IDATG creates a valid data record.

Example file:
```
Name        Age   Profession        <= Column Names
Smith       33    Engineer          <= Data Record 1
Miller      66    Student           <= Data Record 2
```

5.8 Interface to ESA Test Commander

Due to its singular functionality, IDATG has become part of the test automation framework of the European Space Agency ESA. IDATG is used for generating easily maintainable test cases that can be executed by the ESA Test Commander tool. The following section is intented for testers of the Spacecraft Operating System SCOS-2000.

Workflow:

1. When starting IDATG, the file *\bin\TecoDriverData.xml* is loaded automatically. It contains information about the test templates that are supported by ESA's test execution tool Test Commander. If you wish to change this data, just open the file in a text editor and apply your changes. Make sure that the file is compatible with the XML scheme *TecoDriverData.xsd*.

2. For each test item in the original SCOS-2000 test plans, create a separate project. In general, we recommend the following mapping of SCOS test plans to IDATG test specifications:

SCOS	**IDATG**
Test Item	Project
Test Design	Folder
Test Case	Use Case Task
Test Step	1..n Task Steps

3. In the **Project Administration Dialog** choose the Output Format "XML for ESA Test Commander". Apart from changing the output format this choice also affects the appearance of the **Step Editor**.

4. Create the task hierarchy as suggested in the screenshot. Create a folder on the first hierarchy level for each test design and insert one use case task for each test case. Organize the building blocks in an extra folder.

5. Specify the software requirements or import them from an existing source and assign them to individual use case tasks or folders.

6. For each use case, define the task flow. Try to construct it out of re-usable building blocks (yellow). Use parameters if possible. Going top-down, refine the building-blocks into smaller blocks until you have only atomic steps (blue). Do not use branches unless absolutely necessary. The aim is to have exactly 1 test case for each use case task.

7. Define individual steps. Usually the steps in the SCOS test plans consist of more than one action, so it might be necessary to divide them into various IDATG steps. When you double-click on a task step the **Step Editor** should look like this:

8. Choose one of the Test Templates from the list. IDATG then displays information about the corresponding instruction and the expected parameters. Note that there are general templates (starting with "_", e.g. "_SHELL") as well as specific templates (e.g. "C-Shell Script").

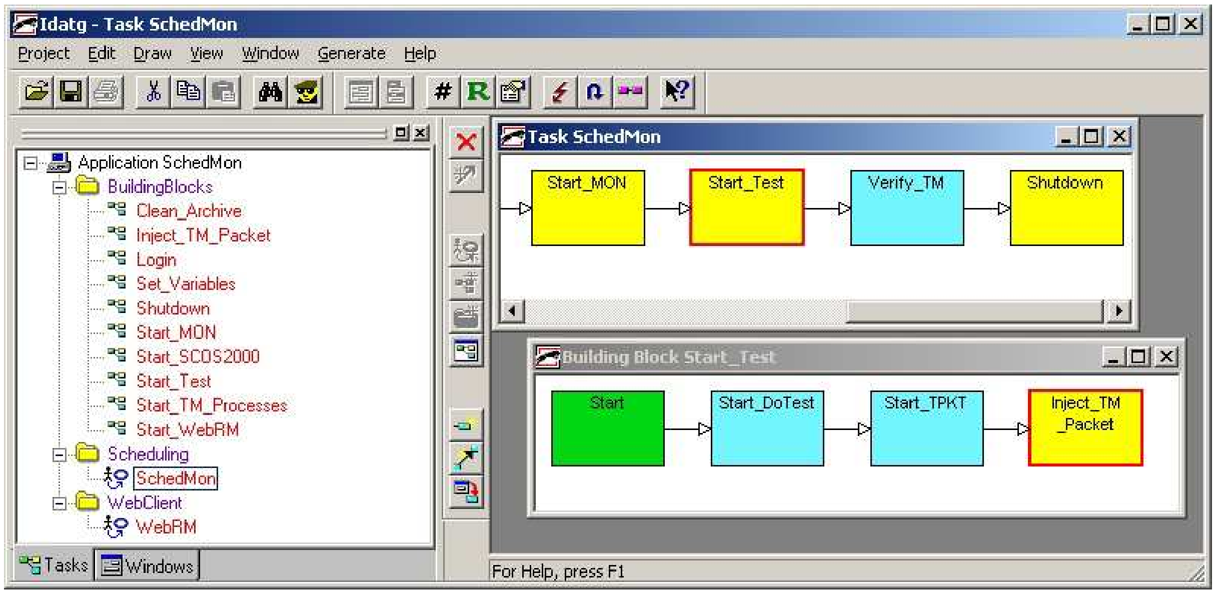


Figure 17 - Sample Task Hierarchy for ESA Tests

9. Define values for the parameters. For some templates, it is also allowed to add new parameters or delete existing ones.

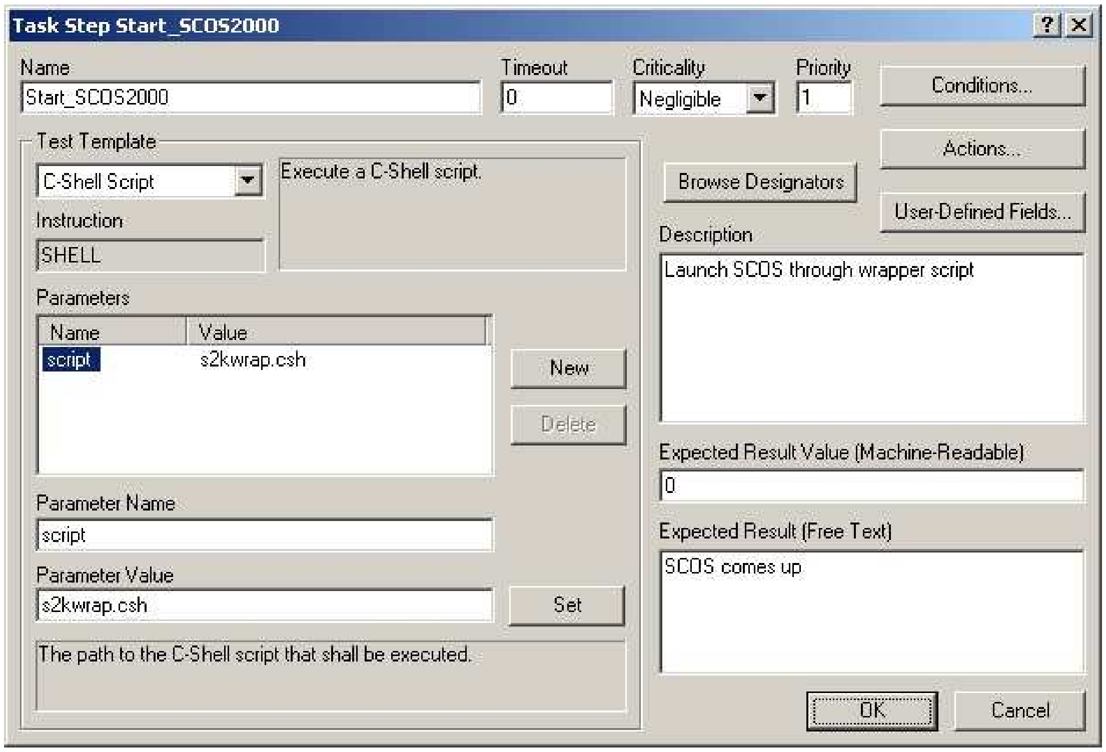
10. Specify the timeout (in milli-seconds), choose the criticality (i.e. how critical a failure of this step would be) and write a textual description.

11. Specify the expected result of the step as plain text. If the step will be executed automatically, it is also important to specify the result in an machine-readable form (usually the expected return value of the invoked function).

12. It is possible to use designators (variable names) in the fields Parameter Value, Description, Expected Result Value and Expected Result. During test case generation, these designators will be replaced by their actual values.

13. Generate test cases

14. Convert the test cases into the format "XML for ESA Test Commander". One XML file is produced for the entire project. It can now be imported into ESA Test Commander.


Task Step Start_SCOS2000

Name: Start_SCOS2000
Timeout: 0
Criticality: Negligible
Priority: 1

Test Template: C-Shell Script
Execute a C-Shell script.
Instruction: SHELL

Parameters

Name	Value
script	s2kwrap.csh

Parameter Name: script
Parameter Value: s2kwrap.csh
The path to the C-Shell script that shall be executed.

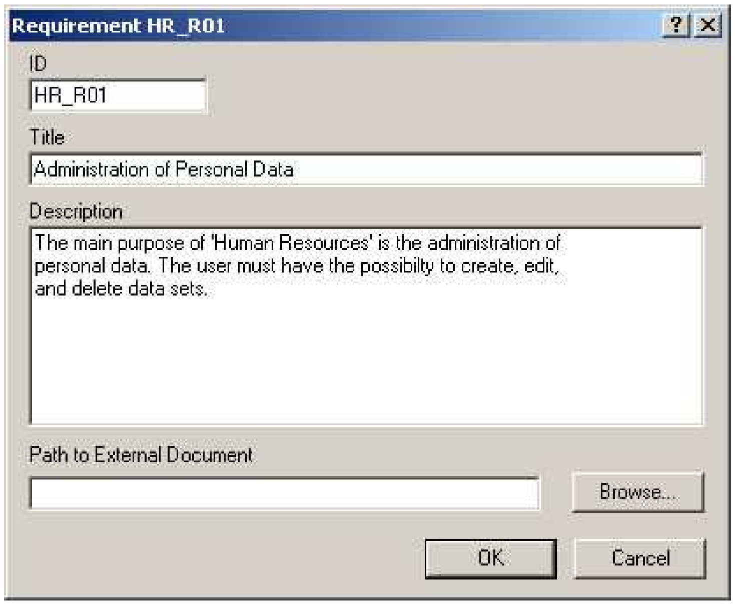
Description: Launch SCOS through wrapper script
Expected Result Value (Machine-Readable): 0
Expected Result (Free Text): SCOS comes up

Figure 18 - Step Editor for ESA Tests

6 Editors for Defining the Task Model

6.1 Requirement Editor

You can edit the properties of a requirement by selecting it in the **Requirements Hierarchy** and pressing the toolbar button '**Edit Properties**'. You may either define each requirement in detail or just specify a path to an external document where additional information about it can be found.


Requirement HR_R01

ID: HR_R01

Title: Administration of Personal Data

Description: The main purpose of 'Human Resources' is the administration of personal data. The user must have the possibilty to create, edit, and delete data sets.

Path to External Document:

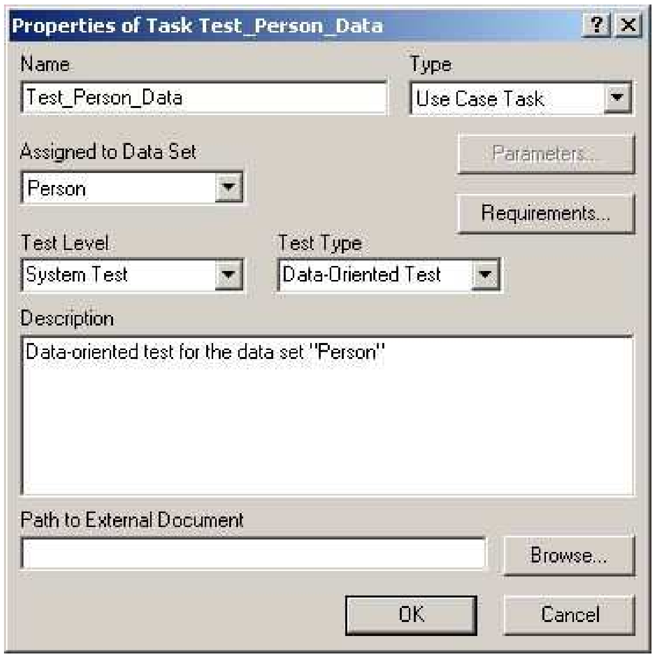
Figure 19 - Requirement Editor

Editing Requirement Properties

- The **ID** must be unique. In order to be compatible with your project-specific naming conventions, the ID may contain any characters and be up to 50 characters long.
- The **Title** can be used to define a more intelligible identification for the requirement.
- You can enter an arbitrary **Description** that explains the purpose of the requirement.
- In addition, you may enter the **Path to an External Document** in which information about this requirement can be found. (e.g. the requirements specification document). The '**Browse…**' button helps you to select the correct file.

6.2 Task Properties Editor

You can edit the properties of a task by selecting it in the **Task Editor** and pressing the toolbar button '**Edit Properties**'.



Properties of Task Test_Person_Data

Name: Test_Person_Data
Type: Use Case Task
Assigned to Data Set: Person
Test Level: System Test
Test Type: Data-Oriented Test
Description: Data-oriented test for the data set "Person"
Path to External Document:

Figure 20 - Task Properties Editor

Editing Task Properties

- The **Name** must be unique, may only contain alphanumeric characters and '_' and be up to 50 characters long.
- The **Type** defines the context of the object. It can either be 'Folder', 'Building Block' or 'Use Case Task'. Note: Once you change the type to 'Folder', the selection cannot be reversed.
- If you assign a use case task to a **Data Set** it gets the icon and you will be able to generate data-oriented test cases for it. See User's Guide Part I for more information.
- 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.

Editing the Parameters of the Task (only for building blocks)

Using the button '**Parameters…**' you can reach the **Task Parameters Editor** that allows you to define the parameters of this task.

Assigning Requirements to the Task (only for folders and use-case tasks)

Using the button '**Requirements…**' you can reach the **Task Requirements Editor** that allows you to select the requirements that should be associated with this task.

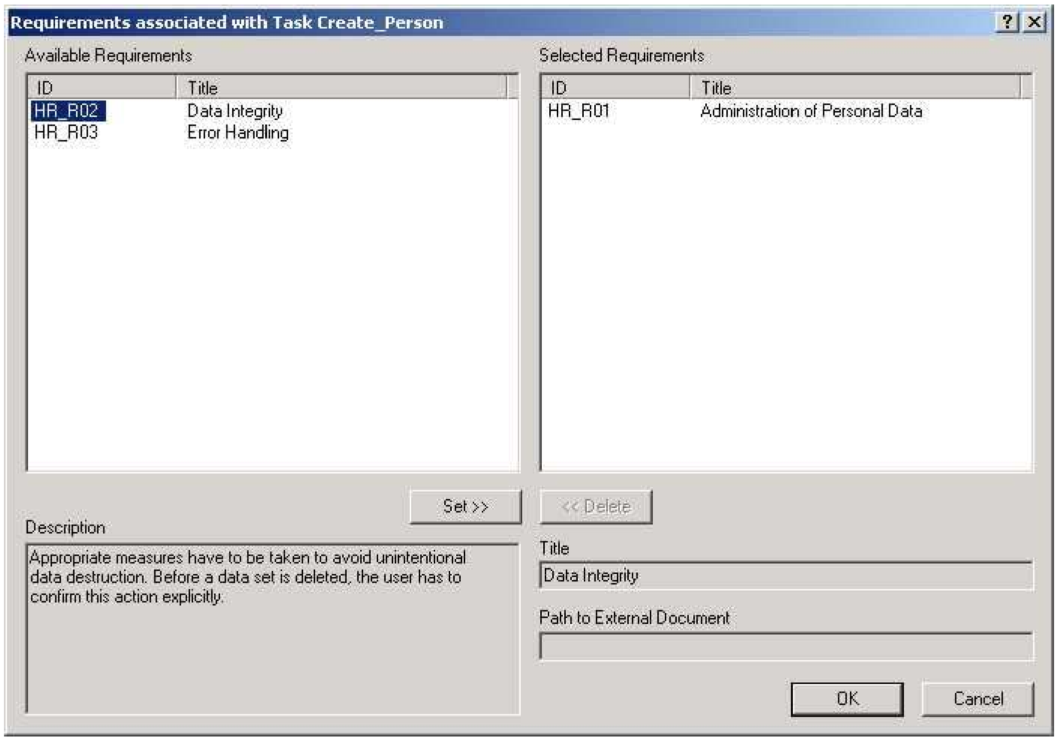
6.3 Task Parameters Editor

A building block task 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 in the parameters. See also section 6.8 about the **Sub Task Editor**.

The **Task Parameters Editor** has essentially the same appearance and functionality as the **Attributes Editor** (see 6.13).

6.4 Task Requirements Editor

If you have defined your project's requirements in the **Requirement Editor** (see 6.1), you can later associate these requirements with one or more tasks. This gives you the advantage that you know exactly which requirements are already covered by your test specification.
The assignment of requirements to a task is the purpose of the **Task Requirements Editor** that can be reached from the **Task Properties Editor**.


Requirements associated with Task Create_Person

Available Requirements

ID	Title
HR_R02	Data Integrity
HR_R03	Error Handling

Selected Requirements

ID	Title
HR_R01	Administration of Personal Data

Description: Appropriate measures have to be taken to avoid unintentional data destruction. Before a data set is deleted, the user has to confirm this action explicitly.

Title: Data Integrity

Path to External Document:

Figure 21 - Task Requirements Editor

The editor contains two lists: In the left one you see all requirements that have been defined for the project, the right one contains only the requirements that have been assigned to the current task.
Note: It is possible to assign the same requirement to more than one task.

Assigning a Requirement to the Task

- Select a requirement in the left list
- Press '**Set >>**'. The requirement moves from the left list into the right.

Removing a Requirement from the Task

- Select a requirement in the right list
- Press '**<< Delete**'. The requirement moves from the right list into the left.

Details about the selected requirement are displayed in the fields at the bottom, but cannot be edited directly. Please use the **Requirement Editor** to change this information.

6.5 Task Flow Editor

The **Task Flow Editor** serves for defining the sequence of steps that has to be performed in order to fulfill a task. The task flow is represented as a directed graph.

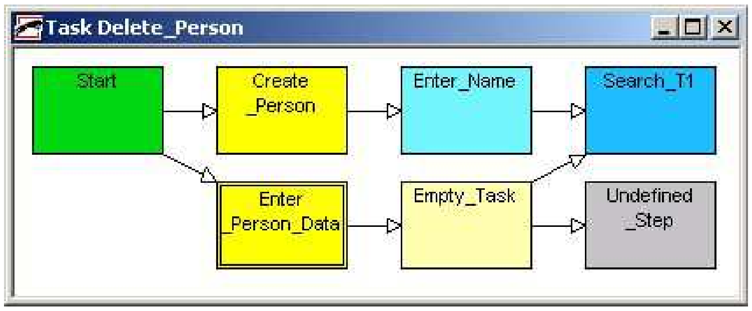


Figure 22 - Task Flow Editor

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 particuarly 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** or **sub tasks** that refer to a building block. The building block name is used as step name and cannot be edited. If the sub task has been defined as **loop**, the step has a double-lined border.

🟨 Light Yellow: Same as above. The lighter color signifies that the **task flow** of the building block 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.

The most important properties of a step are displayed in a **tooltip-window** as soon as the mouse cursor is moved over it. The abbreviations signify: A = Action, C = Condition, E = Event, L = Loop Settings, P = Parameter, S = Script Commands

Steps and connections cannot be dragged around by the user but are instead placed automatically by the editor. This avoids awkward manual adjustments after inserting or deleting objects. It is not allowed to draw isolated steps that cannot be reached from the start step. Therefore, if the last connection to a step is deleted, the editor connects the isolated step to the start step automatically. Likewise, if a step in the middle of a task flow is deleted, all of its predecessors are connected automatically to all of its successors.

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 border.

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 ‘**Insert Step**’
- 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 ‘**Insert Step**’
- An undefined step is created between the two steps and connected to both of them.

Editing a Step

- Select the step and press the toolbar button '**Edit Properties**'. For atomic steps you can also double-click on the step.
- The **Step Editor** appears and allows you to change the step properties. Depending on the type of step, different fields can be edited.

Assigning an Object to a Step

- Select the step
- Press the toolbar button ‘**Assign Object to Step**’
- 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 either select 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.

Viewing the Window assigned to a Step

If you want to view the window or transition that has been assigned to an atomic step (blue), just select the step and press the toolbar button '**Edit Window**' or '**Edit Menu**'.

Viewing the Building Block assigned to a Step

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

Finding a Task in the Task Hierarchy

If you want to find the displayed task in the **Task Hierarchy**, simply right-click the white background of the editor and select '**Show in Task Tree**'. If you want to find one of the yellow building blocks in the **Task Hierarchy**, right-click on the building block instead.

Copying Steps

- Select one or more steps and press '**Copy**' in the toolbar or the menu. All connections between the selected steps are also copied (but no others). The tasks's start step may 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.

Adding a new Connection

This works very much like drawing a new transition in the **Window Editor**:

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

Deleting a Connection

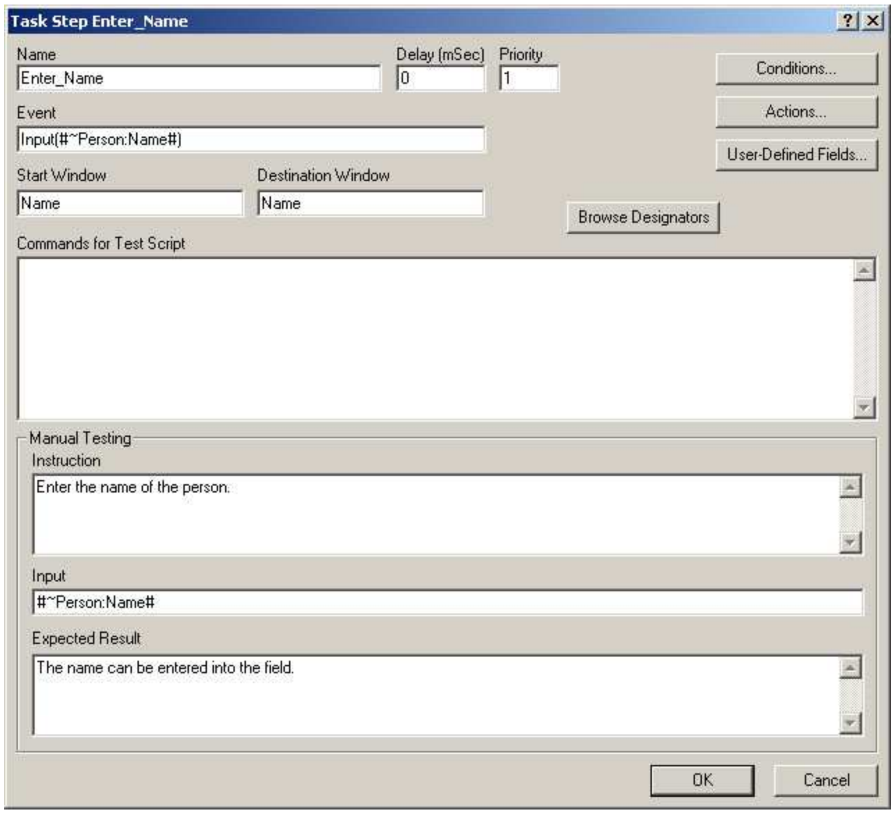
The selected connection may be deleted by choosing the menu **'Delete'** or by pressing the corresponding toolbar button. If the destination step is now isolated (has no predecessors), it is connected automatically to the start step.

6.6 Step Editor

The **Step Editor** gives you the possibility to specify the semantic details of your test specification. 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 task step in the **Task Flow Editor**
- Double click on a test step in the **Test Case Editor**
- Double click on a transition (blue arrow) in the **Window-** or **Menu 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 (see section 6.8).



Name: Enter_Name
Delay (mSec): 0
Priority: 1
Event: Input(#~Person:Name#)
Start Window: Name
Destination Window: Name
Commands for Test Script:
Manual Testing
Instruction: Enter the name of the person.
Input: #~Person:Name#
Expected Result: The name can be entered into the field.

Figure 23 - Step Editor

Specifying a Step Name

Step names make it easier for you to identify the steps in the generated test script. If you assign a new step to a window, its ID is used as step name. Of course you may change this name later on. It may only contain alphanumeric characters and '_' and be up to 50 characters long.

Specifying a Delay

In some cases the test execution tool reports an error because the tested application did not respond quickly enough (may happen e.g., when loading a web page). You can solve this problem by specifying a delay for time-consuming steps (0 - 3.600.000 Milliseconds). An additional wait-command will be inserted in the generated test script **after** the step. This causes the test execution tool to wait for the specified time before continuing with the next step.

If you select the 'Optimize Wait' option when converting the test cases for WinRunner, the delay is used as parameter for the following *set_window* command if possible. In this case, the test continues as soon as the specified window is available. To wait until a window property reaches a certain value, you can use the *Wait* event instead of specifying a delay.

Specifying an Event

If the step is assigned to a window, you can specify the trigger event of the step using the event language (please consult User's Guide Part I for more information). The event is later on translated into a command for the selected test tool. For instance, the IDATG event *Input* is translated into *edit_set* for WinRunner and *SetText* for SilkTest. The event always refers to the start window of the step. If you wish to use a designator as parameter for the event, you can insert it conveniently into the field using the button '**Browse Designators**'.
For non-GUI testing or for specifying self-defined test commands, please use the field '**Commands for Test Script**' instead.

Specifying a Start and Destination Window

Usually you define the start and destination window of a step by pointing on it either during its creation (in case of a transition) or when using the toolbar button '**Assign Object to Step**' (in case of a task or test step). However, you may change this selection by writing different window IDs into the corresponding fields. You may also use '**Browse Designators**' to copy a window ID into the active field.
Note that the start and destination window of a transition may also be changed with the menu '**Draw|Redirect Transition**'.

Specifying Commands for the Test Script

For non-GUI testing or for specifying self-defined test commands, you can enter arbitrary text into this field. If a text is entered here, the field '**Event**' will be ignored. The text is exported directly without translation and therefore you are responsible for the correct syntax. There are 2 exceptions:

- Designator names that can be conveniently inserted into the field using the button '**Browse Designators**'. They are replaced by their actual values during generation. For instance, you may want to specify a Tcl command like *puts #:myvar#* that refers to a global attribute. During generation, the designator name is replaced by the current value of *#:myvar#* resulting e.g. in *puts Hello World*. Of course you can also use more than one designator in this field.
- For WinRunner commands, keyboard key names can either be entered in WinRunner syntax (e.g. <kPgUp_E>) or in IDATG syntax (e.g. <pgup>, see 'Type' event in User's Guide Part I). The second method has the advantage that the key name is not country-dependend and will be translated into the appropriate WinRunner name during conversion (e.g. <kPgUp_E> for US keyboards and <kBild_auf_E> for German ones.

Specifying Instructions for Manual Testing

For manual testing, the fields '**Event**' and '**Commands for Test Script**' are typically left empty. Instead, the three fields '**Instruction**', '**Input**', and '**Expected Result**' can be filled with a textual description of the step. They correspond to the fields with the same name in TEMPPO Test Manager. The text will be exported directly without translation. The only exceptions are designator names that can be conveniently inserted into the fields using the button '**Browse Designators**'. They are replaced by their actual values during generation.

Specifying Conditions

Semantic conditions that have to be fulfilled before the step can be executed can be defined in the **Condition Editor**. It can be opened by pressing '**Conditions...**'. Since it doesn't make sense to enter conditions for a test case that has already been generated, the button is not available for test steps.

Specifying Actions

GUI changes as a result of the step can be defined in the **Action Editor**. It can be opened by pressing '**Actions...**'. Since it doesn't make sense to enter actions for a test case that has already been generated, the button is not available for test steps.

Specifying Additional Information in User-defined Fields

In some projects it may be required to save additional data for each step (e.g. the test driver that should be used or the person who is responsible for this step). The button '**User-Defined Fields…**' opens the **User-defined Fields Editor** that allows you to enter this information.
Note: First you have to use the IDATG menu '**Window|User-Defined Fields Editor**' to define the general structure of these additional fields (see).

Marking a Transition as Error Case (only for Transitions)

If a step is executed as a result of wrong user input, the box '**Error Case**' should be checked. Such steps typically have an error message as destination window. If checked, the generated test cases are also marked as error cases and can be distinguished easier from normal test cases.

Assigning a Priority to a Task Step (only for Task Steps)

If the task flow includes branches, it may be helpful to assign different priorities to the steps (1 = highest, 999 = lowest). The test case generator works according to the following rules:

1. Use connections that are not already covered by a test case
2. Use step with highest priority
3. Use steps with undefined task flow only if no other possibilities are available

6.7 Designator Selection Dialog

For your convenience, whenever a window ID or other designator can be entered into a field, the button '**Browse Designators**' can be used to display all available designators.

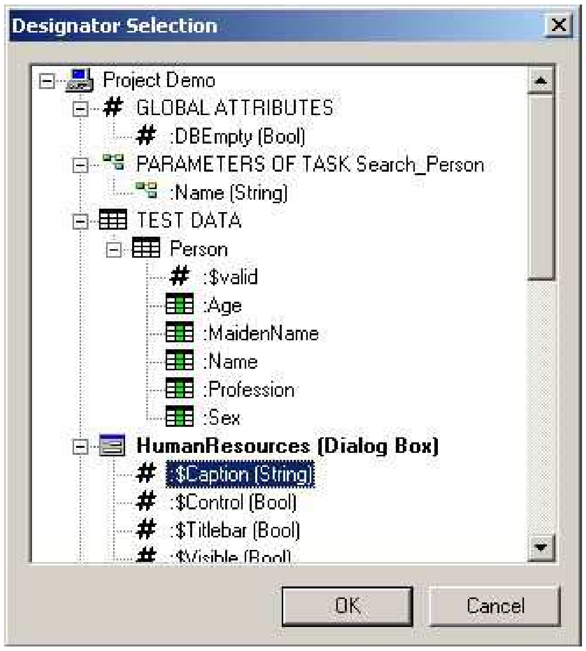


Figure 24 - Designator Selection Dialog

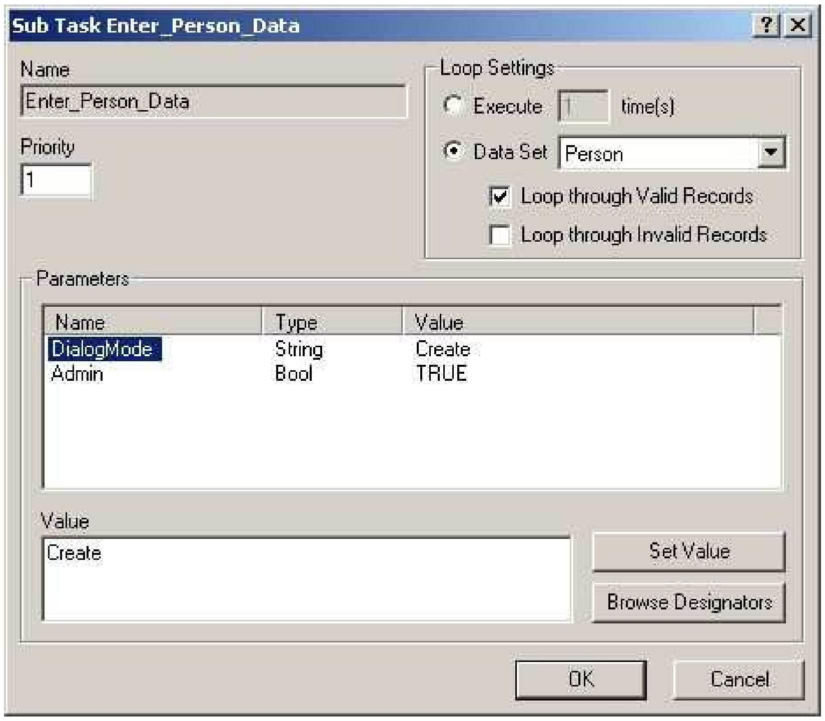
If just a window ID is required, only the window hierarchy is displayed. Otherwise, the tree also contains the global attributes, data sets and fields, the pre-defined window properties (begin with a "$" sign) and the user-defined window attributes. If the currently edited object is a task step, the parameters of its task are also displayed.

Selecting a Designator

Just double-click on an item or select it and press '**OK**'. The full name of the selected designator will be copied into the active field. In the example above, this would be the string *#HumanResources:$Caption#*.

6.8 Sub Task Editor

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


Sub Task Enter_Person_Data

Name: Enter_Person_Data

Priority: 1

Loop Settings
- Execute [1] time(s)
- [✓ Data Set] Person
- [✓] Loop through Valid Records
- [ ] Loop through Invalid Records

Parameters

Name	Type	Value
DialogMode	String	Create
Admin	Bool	TRUE

Value: Create

Figure 25 - Sub Task Editor

Assigning a Priority to a Step

If the task flow includes branches, it may be helpful to assign different priorities to the steps (1 = highest, 999 = lowest). The test case generator works according to the following rules:

1. Use connections that are not already covered by a test case
2. Use step with highest priority
3. Use steps with undefined task flow only if no other possibilities are available

Defining the Loop Settings

If a sub task should be executed more than once, it can be defined as **loop**. The simplest way is to define the number of times it should be repeated (any number between 1 and 100).

However, it is also possible to define loops for data-oriented testing. In this case, the loop is repeated once for each valid and/or invalid record in a data set. This enables you to use more than one record in the same test case. In contrast, the option '**Data-oriented Test**' in the **Test Case Generator** generates a separate test case for each record and should therefore not be combined with data-oriented loops (choose '**Graph-oriented Test**' instead).

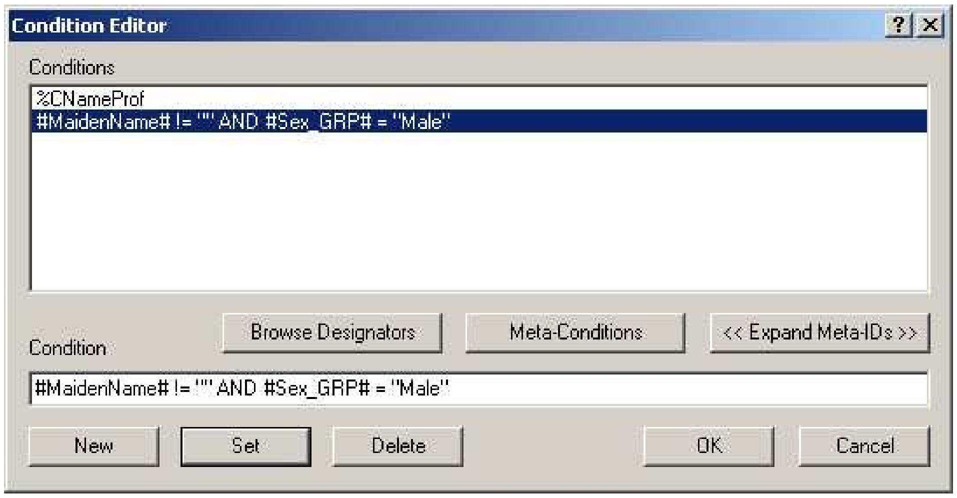
Editing the Parameter Values

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 everytime 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.
- If you wish to use a designator in the expression, you can insert it conveniently into the field using the button ‘**Browse Designators**’.
- Press '**Set Value**'. The value is checked for syntactic correctness.
- If more parameters are available, the next one is selected automatically.

6.9 Condition Editor

Steps often require that special conditions are fulfilled before they can be executed. These conditions have to be entered in the **Condition Editor** using an IDATG-specific language. Conditions can depend on the state or contents of a window, a user-defined attribute etc. The syntax required is called the condition syntax. (see User's Guide Part I)
All displayed conditions have to be TRUE to execute the transition. Thus, complicated conditions containing ‘AND’ operators can be divided into separate parts.



```
%CNameProf
#MaidenName# != "" AND #Sex_GRP# = "Male"
```

Condition: #MaidenName# != "" AND #Sex_GRP# = "Male"

Figure 26 – Condition Editor

The order of the conditions in the list can affect the duration of test case generation, because IDATG tries to fulfill the conditions in the specified order. If possible, try to put the conditions in a logical sequence.

Imagine the following scenario: Condition C1 is *#A# = TRUE*, C2 is *#B# = TRUE*.
Let’s assume that setting A to TRUE has no side-effect, but setting B to TRUE automatically sets A to FALSE. If we write the conditions in the order C2, C1 IDATG would need more time to find a solution than for the order C1, C2.

Adding a New Condition

- Press ‘**New**’
- Enter the condition
- If you wish to use a designator in the condition, you can insert it conveniently into the field using the button ‘**Browse Designators**’.
- Press ‘**Set**’ ➔ the condition parser is called to check the condition for syntactic correctness. If an error occurs, a message box with a detailed description of the error appears and the focus moves to ‘**Condition**’.

Editing a Condition

- Select a condition
- Edit the condition.
- Press ‘**Set**’ ➔ the condition parser is called to check the condition for syntactic correctness. If an error occurs, a message box with a detailed description of the error appears and the focus moves to ‘**Condition**’.

Deleting a Condition

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

Inserting a Meta-Condition into a condition

The ‘**Meta Conditons**‘ button leads to the **Meta Condition Editor**. If you select a meta-condition and press OK its ID is copied into the edit box.
Meta Condition IDs are always shown with a leading ‘%’ (e.g. %C1).

Expanding Meta-Conditions

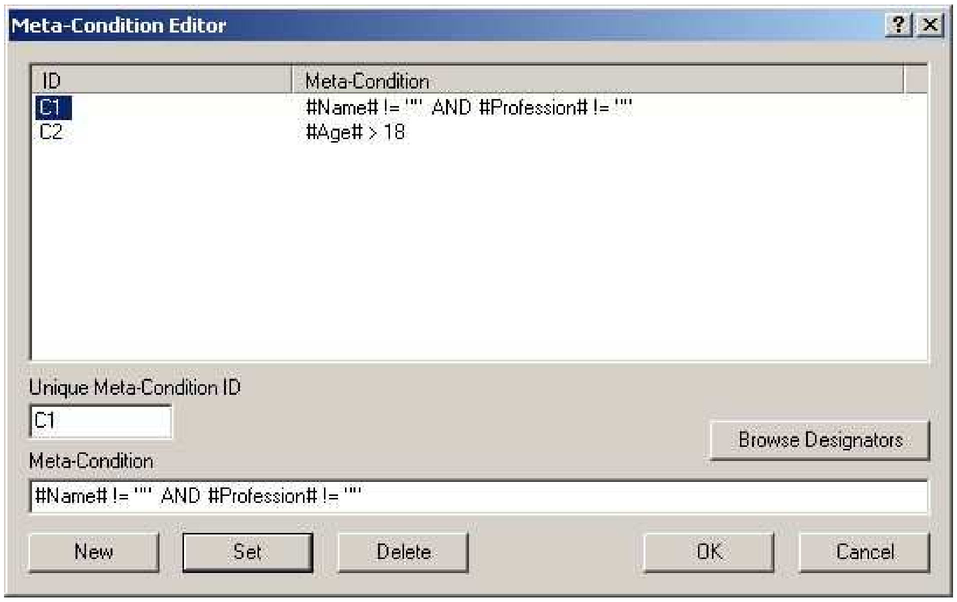
Often it can be helpful to see the entire meta-conditions instead of their IDs.
The **Expand Meta IDs / Implode Meta IDs** button switches between two views: it shows either only the meta condition IDs or the entire meta-conditions.

Example: Let’s assume a meta condition C1 that is *#A# = TRUE.*
If it is used in a condition like *#B# = TRUE AND %C1*
the expanded view would be *#B# = TRUE AND #A# = TRUE.*
Note: in the expanded view it is not possible to edit the condition.

6.10 Meta-Condition Editor

If a condition appears more than once in the GUI specification, you can save time by defining it as a **Meta-Condition**. Meta-conditions are like normal conditions but can be used in other conditions like macros. They have a unique meta-condition ID. To address meta conditions insert a "%" before the ID e.g. *%c5 OR (#Name# = "")*

To reach this editor, choose the menu command '**Window | Meta-Condition Editor**' or choose the '**Meta-Conditions**' button in the **Condition Editor**.



ID	Meta-Condition
C1	#Name# != "" AND #Profession# != ""
C2	#Age# > 18

Unique Meta-Condition ID: C1

Meta-Condition: #Name# != "" AND #Profession# != ""

Figure 27 – Meta-Condition Editor

The **Meta Condition Editor** includes a list containing the meta condition IDs and the corresponding meta conditions. These meta conditions can be edited like normal conditions in the **Condition Editor**. The meta condition IDs have to be unique.

Adding a New Meta-Condition

- Press '**New**'
- Enter a unique meta-condition ID. It may only contain alphanumeric characters and '_' and have a maximum length of 30 characters.
- Enter the meta-condition.
- If you wish to use a designator in the meta-condition, you can insert it conveniently into the field using the button '**Browse Designators**'.
- Press '**Set**' ➔ the condition parser is called to check the meta-condition for syntactic correctness. If an error occurs, a message box with a detailed description of the error appears and the focus moves to '**Meta-Condition**'.

Selecting a Meta-Condition

Simply click on the meta-condition or its ID in the list box. If the editor was called from the **Condition Editor**, pressing '**OK**' causes the meta-condition ID to be copied into the currently edited condition.

Editing a Meta-Condition

- Select a meta-condition.
- Edit the meta-condition.
- Press ‘**Set**’ ➔ the condition parser is called to check the meta-condition for syntactic correctness. If an error occurs, a message box with a detailed description of the error appears and the focus moves to ‘**Meta-Condition**’.

Deleting a Meta-Condition

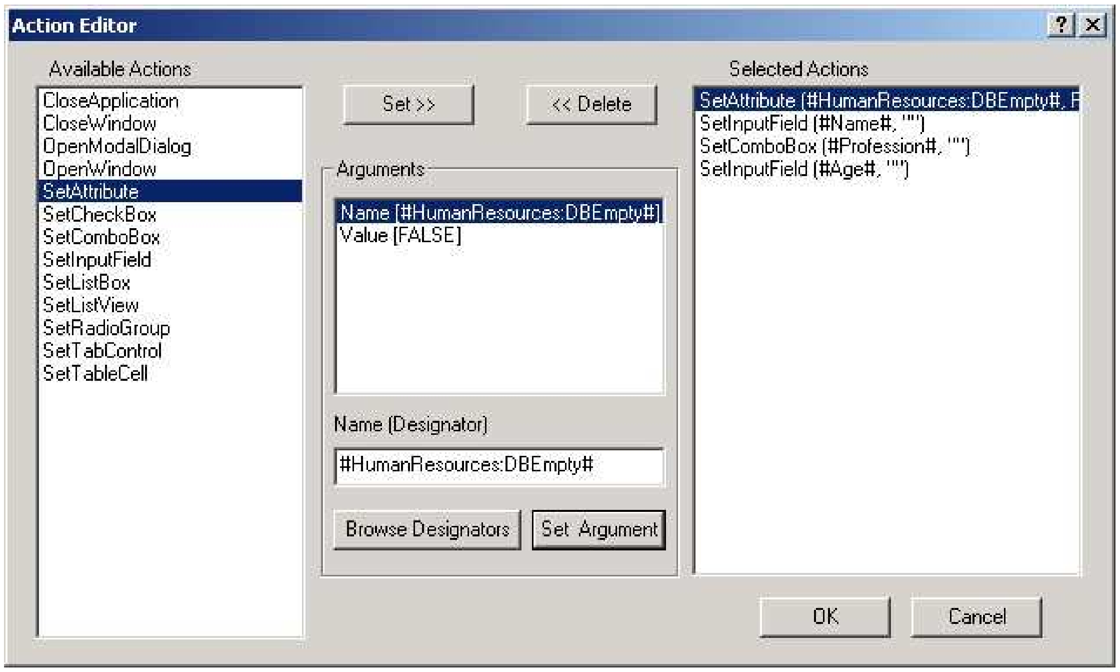
- Select a meta-condition.
- Press ‘**Delete**’ ➔ a message box appears that has to be confirmed by the user. (”Do you really want to delete this meta-condition ?“)

6.11 Action Editor

Actions are simple functions like calling a window, setting a designator to a new value etc. and are necessary to obtain a complete specification of the GUI behavior. Depending on the type of function, they can have various arguments. Arguments do have the same syntax as conditions and therefore also can be complex expressions.

For example: *SetInputField(#Age#, 5 + (3 * #Aboutbox:Calls#)).*

The **Action Editor** allows you to specify an arbitrary sequence of actions for each step.


Action Editor

Available Actions: CloseApplication, CloseWindow, OpenModalDialog, OpenWindow, SetAttribute, SetCheckBox, SetComboBox, SetInputField, SetListBox, SetListView, SetRadioGroup, SetTabControl, SetTableCell

Arguments: Name [#HumanResources:DBEmpty#], Value [FALSE]

Name (Designator): #HumanResources:DBEmpty#

Selected Actions:
SetAttribute (#HumanResources:DBEmpty#, F
SetInputField (#Name#, "")
SetComboBox (#Profession#, "")
SetInputField (#Age#, "")

Figure 28 - Action Editor

The left list box contains the complete list of available actions in alphabetical order. In case of a new transition the right list box is empty, otherwise it contains the actions that have already been selected for the transition. The right list is NOT ordered alphabetically, since the actual sequence of the selected actions is important. It is also possible that it contains more than one call for the same action.

Adding an Action

- Select the desired action by clicking on it in the left list box ➔ The arguments appear in the **Arguments** list box. The first argument in the box is selected automatically and the input field under the argument list box is labeled with the name and type of this argument.
- Enter a value for the argument. If you wish to use a window ID or designator in the argument, you can insert it conveniently into the field using the button '**Browse Designators**'.
- Press '**Set Argument**' ➔ the condition parser is called to check the syntax of the argument. It is also checked, whether the type of the value matches the required argument type.
  In case of success, the value of the argument is updated in the argument list box and the next argument is selected automatically. In case of an error, a message box appears and the cursor returns to the argument field.
- It is also possible to select a specific argument in the argument list manually. In this case, the input field under the argument list box is labeled with the name and type of this argument. The cursor moves to this field.
- Press '**Set**' ➔ if not all arguments have been defined, an error message is displayed. The action is added to the right list box and the editor returns to its initial state.

Editing an Action

- Select the desired action in the right list box ➔ The action is highlighted in the left list box. The first argument in the box is selected automatically and the input field under the argument list box is labeled with the name and type of this argument.
- Edit the argument values as usual.
- Press '**Set**' ➔ if not all arguments have been defined, an error message is displayed. The action is updated in the right list box and the editor returns to its initial state.

Deleting an Action

- Select the desired action in the right list box.
- Press '**Delete**' ➔ a message box has to be confirmed, then the action is removed from the right list box, and the editor returns to its initial state.

6.12 User-Defined Fields Editor

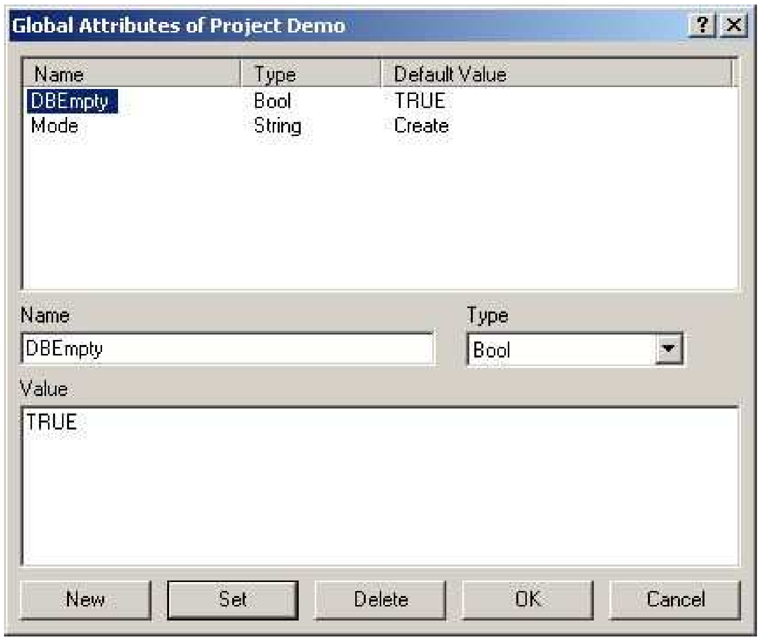
In some projects it may be required to save additional data for each step (e.g. the test driver that should be used or the person who is responsible for this step). Just use the IDATG menu '**Window|User-Defined Fields Editor**' to open the **User-defined Fields Editor** that allows you to define the general structure of these additional fields (name, type and default value). After that, the new fields become available in the **Step Editor**.

The **User-defined Fields Editor** has essentially the same appearance and functionality as the **Attributes Editor** (see 6.13).

6.13 Attributes Editor

This dialog is used for editing different kinds of objects:

- Global attributes (toolbar button '**Edit Attributes**')
- Window-specific attributes (a window is selected before pressing '**Edit Attributes**')
- Task Parameters (button '**Parameters…**' in the **Task Properties Editor**)
- User-defined step fields (button '**User-Defined Fields…**' in the **Step Editor**)
- Effect variables (button '**Effect Variables**' in the **Data Effects Editor**)



Name	Type	Default Value
DBEmpty	Bool	TRUE
Mode	String	Create

Name: DBEmpty
Type: Bool
Value: TRUE

Figure 29 - Attributes Editor (for Global Attributes)

Creating a New Attribute

- Press '**New**'
- Enter name, type and start value. The name may only contain alphanumeric characters and '_' and may be up to 30 characters long. The syntax for the start value depends on the selected type:
  String - arbitray text (do not quote it with "")
  Num - a signed integer (the + may be omitted) e.g. 15, +15, -15
  Bool - TRUE or FALSE
  Date - a valid date in IDATG format e.g. 12.05.1988
- Press '**Set**'. It is checked whether the name is unique and the start value corresponds to the selected type. If not, a message box appears and the focus moves to the invalid field.

Editing an Attribute

- Select an attribute in the list.
- Edit name, type and start value.
- Press '**Set**'. It is checked whether the name is unique and the start value corresponds to the selected type. If not, a message box appears and the focus moves to the invalid field.

Deleting an Attribute

- Select an attribute in the list.

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

7 Editors for GUI Specification

7.1 Window Editor

For windows of every type, except for menus and menu items, a **Window Editor** may be opened. The window and all its direct children are displayed in a way similar to the dialog editor in MS Visual Studio. The **Window Editor** enables the user to change the appearance of the window itself and its children, and to edit transitions between the windows.
It can be called by selecting a window in the **Application Editor** and pressing the '**Edit Window**' Icon or simply by double-clicking on it. It is possible to open several **Window Editors** at the same time, but only one for each window.

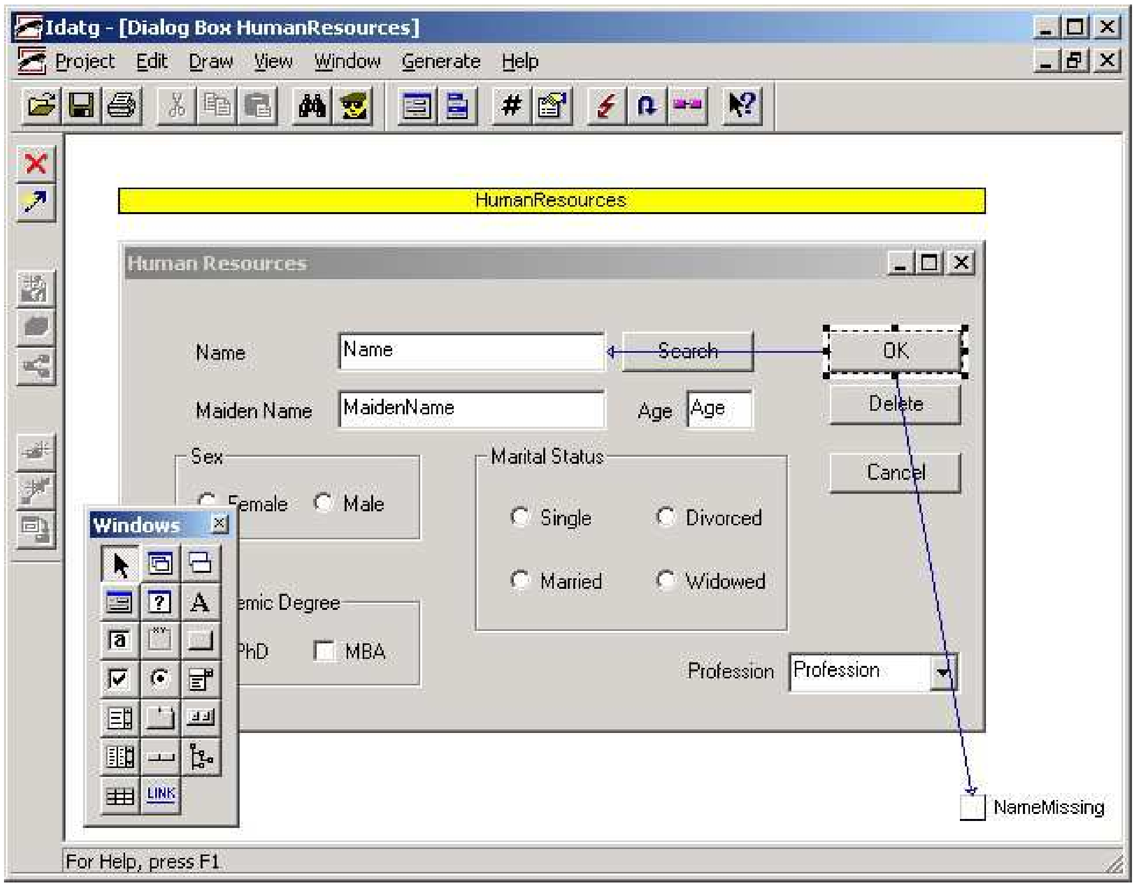


Figure 30 - Window Editor

As we can see, the window and its children are displayed as realistically as possible. The toolbar labeled "Windows" can be used to select a window type before drawing a new window.
By default, all successors of the window are displayed in the editor to provide a realistic view. If you wish to see only the direct children, you may uncheck the menu item '**View|Show Grandchildren**'.

Creating a New Window

- To create a child of an existing window, open the **Window Editor** for its parent. To create a window without a parent, double click the root node in the **Application Editor** to open an empty **Window Editor**.
- Make sure the **Window Toolbar** is visible (Use menu '**View|Window Tool Bar**'). Select the symbol for the desired window type. The cursor shape will change into a pencil.

- Move the mouse cursor to the desired position of the new window and press the left button. While keeping it pressed, move the mouse until the new window has the desired size. If you are creating a child window, you are restricted by the client area of the parent window.
- The new window is created with default properties and with an automatically generated ID. The classname is set depending on the default value specified in the **Window Class Editor**. You can change this values later with the **Window Properties Editor**.

Selecting a Window

- Select the arrow symbol in the **Window Toolbar** to get into selection mode.
- Click on the window or one of its children. Always the innermost window is selected. Note: Windows lying inside the currently selected window cannot be selected. You have to click outside of the parent window before you are able to select them.

Selecting more than 1 Window

Multiple windows can be selected in different ways:

1. Click on an empty area and drag a rectangle over the desired windows
2. Hold the ‘Ctrl’ key while clicking on a window to add it to the selection. This can be useful when selecting children from different areas of the parent window. By holding ‘Ctrl’ it is also possible to remove windows from the current selection.

Note: The parent window cannot be added to a multi-selection.

Copying Windows

- Select one or more windows and press ‘**Copy**’ in the toolbar or the menu. All children of the selected windows are also copied.
- Select a parent window (may be the same or a different one)
- Press ‘**Paste**’ to insert the copied windows as children of the parent.

Moving Windows

Select one or more windows and drag them to the desired position. In addition, it is possible to move the selected windows using the cursor keys.

Restrictions for moving windows:

- You can't drag&drop the parent window. If you want to change its position inside its own parent window, please open the corresponding editor.
- All children must remain in the client area of their parent window.

Resizing a Window

Resizing is only possible if exactly one window is selected. There is a trackable border drawn around the selected window that can be used to track an edge or corner of the window to resize it.

Restrictions for resizing windows:

- You can't track the top/left sides of the parent window. To resize it, please use the bottom/right side. But you can't resize a parent window at all, if it has a parent of its own. In this case you have to open the **Window Editor** for its own parent.
- By resizing the parent window, all its children must still remain in the client area of the parent window.

Editing Window Properties and Attributes

(Pre-defined) Properties of the selected window may be edited by choosing the menu **'Properties Editor'** or by pressing the corresponding toolbar button.

(User-defined) Attributes of the selected window may be edited by choosing the menu **'Attributes Editor'** or by pressing the corresponding toolbar button.

Deleting Windows

- The selected windows may be deleted by choosing the menu **'Delete'** or by pressing the corresponding toolbar button. The windows will be deleted with all their children. Semantic dependencies must be removed manually.

Creating a New Transition

The idea behind the editor is a state-transition diagram. Every window represents a state, state transitions are represented by blue arrows, a bold arrow symbolizes the transition that is currently selected. If a transition points outside of the window to a rectangle symbol, double-clicking on this symbol opens a **Window Editor** for the window represented by it. When drawing a transition, the yellow rectangle on the top containing the window ID can be clicked to select the parent window itself.

There can be more than one transition between the same two windows. For example, this may be necessary if the transitions have different effects (actions), but the cursor always moves to the same window.

- Switch to transition mode with '**View|Show Transitions**'.
- Press '**New Transition**' The shape of the mouse cursor changes into a pencil. In this mode, it is only possible to draw transitions. Press the button again if you want to leave the mode.
- Click on the start window of the transition or the yellow title bar.
- Click on the destination window of the transition, which may also lie in another **Window Editor**.
- The **Step Editor** appears and allows you to specify the properties of the transition. When finished, press '**OK**'.

Selecting a Transition

- Switch to transition mode with '**View|Show Transitions**'.
- Click on the arrow representing the desired transition, which is then displayed bold.

Editing a Transition

- Switch to transition mode with '**View|Show Transitions**'.
- Double click on the transition. The **Step Editor** appears and allows you to change the transition properties.
- Press '**OK**'.

Redirecting a Transition

- Switch to transition mode with '**View|Show Transitions**'.
- Select an existing transition with the mouse.
- Click '**Redirect Transition**' in the '**Edit**' menu. The shape of the mouse cursor changes into a pencil. In this mode, it is only possible to draw transitions.
- The rest of the procedure is very similar to drawing a new transition. Just click on the new start and destination window of the transition. All conditions, actions and other properties of the transition remain unchanged.

Deleting a Transition

- Switch to transition mode with '**View|Show Transitions**'.
- Select an existing transition with the mouse.
- Press '**Delete**'.

- Confirm the appearing message box.("Do you really want to delete this transition?")

Printing

The window and its transitions may be printed using the corresponding toolbar button.

7.2 Menu Editor

The **Menu Editor** allows you to specify the behavior of a window's menu. Basically, the behavior of this editor is quite similar to that of the **Window Editor**. To open this editor, choose a menu in the **Application Editor** and press the '**Edit Menu**' toolbar button.

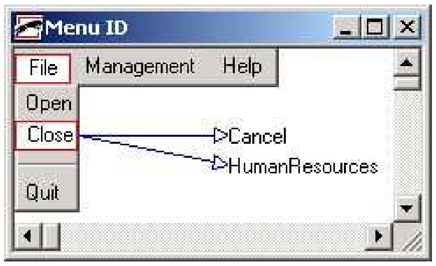


Figure 31 - Menu Editor

Initial State of the Editor

The editor shows the top level menu items of the menu. All sub-menus are closed, they can be opened like in a normal MS Windows menu. Transitions are only shown when a sub-menu is opened and the corresponding menu item has been selected.

Creating a new Menu Item

New menu items can be created using '**Draw|Create Menu Item**'. The new item is always added as child of the currently selected item. For example, if *File* is selected in our sample figure, the new menu item will be added to this sub-menu (behind *Quit*). If nothing is selected, the new menu item will be added as top-level item (right of *Help*). The parent-child relationships can be changed in the **Application Editor**.

Selecting a Menu Item

A menu item can be selected by opening the corresponding sub-menu and clicking on it. Selected items are displayed with a red rectangular border together with their transitions.

Editing Menu Item Properties and Attributes

(Pre-defined) Properties of the selected menu item may be edited by choosing the menu **'Properties Editor'** or by pressing the corresponding toolbar button.

(User-defined) Attributes of the selected menu item may be edited by choosing the menu **'Attributes Editor'** or by pressing the corresponding toolbar button.

Deleting a Menu Item

The selected menu item may be deleted by choosing the menu **'Delete'** or by pressing the corresponding toolbar button. The menu item will be deleted with all its children and transitions. Semantic dependencies must be removed manually.

Creating a New Transition

With the editor you are able to define transitions that are triggered by menu items. Transitions are represented by blue arrows, a bold arrow symbolizes the transition that is currently selected. After creating a new transition or selecting an existing one, the **Step Editor** appears and enables you to specify the properties of the transition.

Note that menu items can only be the start window of a transition, but never the destination window. Therefore, the arrow always points away from a menu item to the ID of the destination window.

- Press '**New Transition**'. The shape of the mouse cursor changes into a pencil. In this mode it is only possible to draw transitions.
- Open a sub-menu and click on the menu item that triggers the transition. Only items that do not have sub-items can be the start of a transition. Items that are only separators (horizontal lines) are of course also excluded.
- Click on the destination window in a **Window Editor**. An arrow that points from the menu item to the name of the window is displayed.
- After drawing the transition, the mouse cursor shape goes back to normal and the **Step Editor** appears.
- Finally, the new transition can be stored by pressing the '**OK**' button.

Selecting a Transition
- Switch to transition mode with '**View|Show Transitions**'.
- Click on the arrow representing the desired transition, which is then displayed bold.

Editing a Transition
- Switch to transition mode with '**View|Show Transitions**'.
- Double click on the transition. The **Step Editor** appears and allows you to change the transition properties.
- Press '**OK**'.

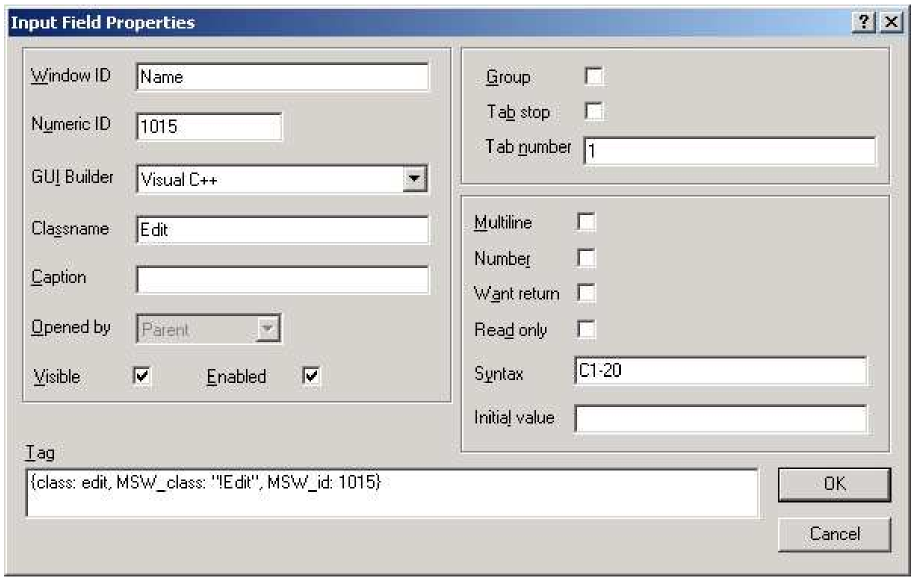
Redirecting a Transition
- Switch to transition mode with '**View|Show Transitions**'.
- Select an existing transition with the mouse.
- Click '**Redirect Transition**' in the '**Edit**' menu. The shape of the mouse cursor changes into a pencil. In this mode, it is only possible to draw transitions.
- The rest of the procedure is very similar to drawing a new transition. Just click on the new start and destination window of the transition. All conditions, actions and other properties of the transition remain unchanged.

Deleting a Transition
- Switch to transition mode with '**View|Show Transitions**'.
- Select an existing transition with the mouse.
- Press '**Delete**'.
- Confirm the appearing message box.("Do you really want to delete this transition?")

Printing
The menu and its transitions may be printed using the corresponding toolbar button.

7.3 Window Properties Editor

The appearance of the **Window Properties Editor** depends on the type of window that the user wishes to manipulate. Some properties exist for all window types, others only for specific ones. Please refer to User's Guide Part I for a detailed description of all window properties.
If you wish to change the type of the window you can use the menu ‘**Edit|Change Window Type**’ before calling the **Window Properties Editor**.


Input Field Properties

Window ID	Name
Numeric ID	1015
GUI Builder	Visual C++
Classname	Edit
Caption	
Opened by	Parent
Visible	✓
Enabled	✓
Group	
Tab stop	
Tab number	1
Multiline	
Number	
Want return	
Read only	
Syntax	C1-20
Initial value	
Tag	{class: edit, MSW_class: "!Edit", MSW_id: 1015}

Figure 32 - Window Properties Editor (for Input Fields)

Editing Window Properties

- Change the desired window properties.
- Press ‘**OK**’ => all fields are checked for correctness (see the details below).
  If an error is detected, a message box appears and the cursor moves to the erroneous field. Press ‘**Cancel**’ to discard your changes.

The most important properties are:

- **Window ID** The unique name of the window used to address it in IDATG. It may only contain alphanumeric characters and ‘_’ and be up to 50 characters long. Depending on the available properties the caption, attached text, or numeric ID is used by default, often followed by a number to guarantee uniqueness (OK1, OK2,..). The window ID is also used in the generated test scripts as identifier of the window. When changing the ID of a window all references to it in conditions, actions etc. are updated automatically, thus no manual editing is necessary.
- **Numeric ID** The numeric window ID is used during the execution of your application (only in MS Windows-based GUIs). This ID can be used as a reliable tag for window recognition. <u>Exception:</u> For parentless windows the ID is invalid and may not be used as tag!
- **GUI Builder** The GUI builder that has been used for implementing the window (some applications include windows created with different GUI builders)
- **Classname** The classname of a window can be recorded with the GUI Spy. It is used to determine the window type and is passed to the test execution tool. No length restriction.
- **Caption** The label of the window. It is not necessary for TCG to enter a caption for windows that have been added manually. However, working with the **Window Editor** becomes easier, if all windows have correct captions. No length restriction.

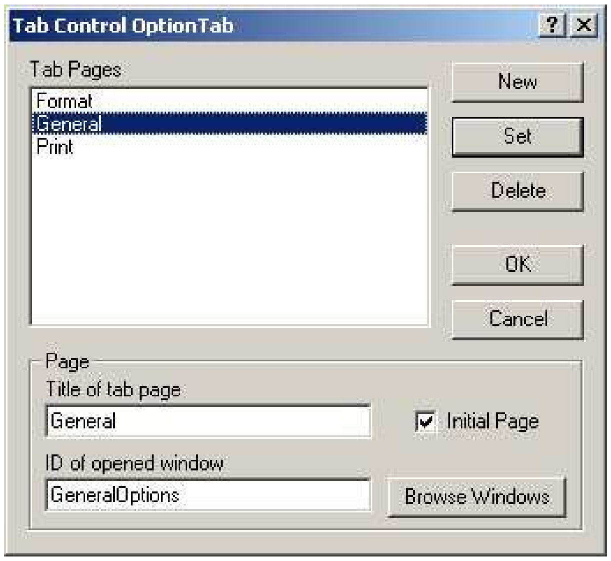
- **Opened by** Specifies whether the window is opened explicitly (like a message box) or whether it is opened automatically when its parent is opened (like the OK-Button in a message box). Very important for test case generation.
- **Visible** and **Enabled** Specifies whether the window is initially visible and enabled. Also important for TCG.
- **Initial value** The value that appears in the window when it is first shown. No length restriction.

Tag Definition
The tag is required by the test execution tool to find the window on the screen. It is very important to enter a correct tag for each window, otherwise the generated test scripts won't run! In most cases, the **Tag Generator** will create correct tags for you, so you don't have to define them here. Please refer to User's Guide Part I if the test execution tool reports an error like "Object XY not found".

Note: If you have chosen "QuickTest Professional" as output format, you will see a button labelled '**QTP Properties…**' instead of the **Tag** field. Likewise, there will be a button '**Ranorex Properties…**' if you have chosen Ranorex as output format. See User's Guide Part I for more information.

7.4 Tab Control Editor

The **Window Properties Editor** for tab controls contains a button '**Tab Pages...**' that opens the **Tab Control Editor**. Tab controls are GUI objects that allow the user to switch between a set of windows. However, in many GUI builders the windows displayed as tab pages are not statically linked to the tab control in any way. This means that it is impossible for the **GUI Spy** to find out which windows belong to a certain tab control. Therefore, a special editor is necessary that allows the user to conveniently specify the pages of a tab control.


Tab Control OptionTab

Tab Pages: Format, General (selected), Print

Page
Title of tab page: General
Initial Page: ✓
ID of opened window: GeneralOptions

Figure 33 - Tab Control Editor

Adding a New Tab Page

- Press 'New'.
- Enter the exact title of the tab page (i.e. the text on which you have to click to open the page). The title has to be unique for this tab control.

- If this page is the default page of the tab control, check '**Initial Page**'. If another page has previously been defined as initial page, it automatically loses this property.
- Enter the ID of the window that is displayed in the page or use '**Browse Windows**' to select it. Usually tab pages are dialog boxes that have the same parent as the tab control (in Java they are children of the tab control). Each window may only be used once as a tab page.
- Press '**Set**' ➔ all fields are checked for correctness. If an error is detected, a message box appears and the cursor moves to the erroneous field. Press '**Cancel**' to discard your changes.

Selecting a Tab Page

Simply click on its title in the list box.

Editing a Tab Page

- Select a tab page.
- Edit the properties of the tab page.
- Press '**Set**' ➔ all fields are checked for correctness. If an error is detected, a message box appears and the cursor moves to the erroneous field. Press '**Cancel**' to discard your changes.

Deleting a Tab Page

- Select a tab page.
- Press '**Delete**' ➔ a message box appears that has to be confirmed by the user. ("Do you really want to delete this tab page ?"*)*

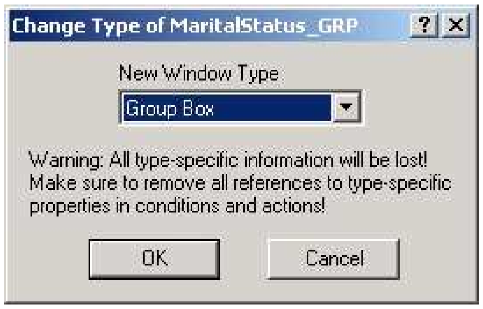
Effects of specifying the pages of a tab control

Specifying which windows belong to a tab control is very important for test case generation because it allows IDATG to generate test cases that correctly manipulate the tab control. Defining a window as a tab page has the following effects:

The window´s open method is automatically set to '**Parent**'. If the window is the initial page of the tab, '**Visible**' is set to TRUE, otherwise to FALSE.

7.5 Change Window Type Dialog

After selecting a window and pressing the menu item '**Edit|Change Window Type**' a small dialog appears that allows you to change the type of the window. This is particularly helpful for unidentified Custom Windows recorded with the **GUI Spy**, but should only be used with care for other windows.


Change Type of MaritalStatus_GRP

New Window Type: Group Box

Warning: All type-specific information will be lost!
Make sure to remove all references to type-specific properties in conditions and actions!

Figure 34 - Change Window Type Dialog

All common properties of the window (ID, visible etc.) are preserved during the conversion, but the type-specific information is lost. For instance, converting an Input Field into a Push Button deletes the Input Field-specific properties like read-only and initial value. On the other hand, the Push Button-specific properties are initialized with default values. The classname automatically changes depending on the default specified in the **Window Class Editor**.

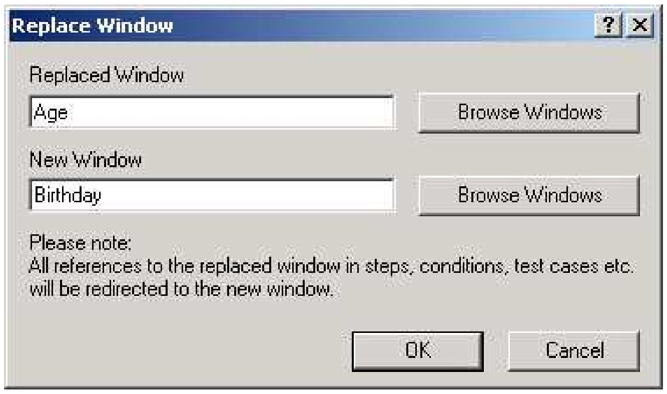
Note: If you have specified references to the type-specific properties of the window in conditions or actions, they may get invalidated and cause the **Test Case Generator** to report an error. For instance, the condition *#FormerInputField# = "Text"* is not longer valid for a Push Button because it doesn´t have a value. Make sure to delete such references before generating test cases.

Changing the Window Type

- Select the window in any editor. Menus and Menu Items cannot be converted.
- Select the new type in the combo box.
- Press '**OK**'.

7.6 Replace Window Dialog

Changes in the GUI of the tested application have also to be considered in the IDATG specification. You may either change the properties of the affected windows manually in the **Window Properties Editor** or record them again with the **GUI Spy**. In the latter case it is necessary to redirect all references from the old window to the new one. This means changing all steps, conditions, actions etc. where the old window has been used.
Fortunately, IDATG provides a convenient function that saves most of this effort. Just select the old window and use the menu '**Edit|Replace Window**' to open the following dialog:



Age

Birthday

Please note:
All references to the replaced window in steps, conditions, test cases etc. will be redirected to the new window.

Figure 35 - Replace Window Dialog

Just select the old and new window using the '**Browse Windows**' buttons and press OK. The old window will NOT be deleted, but all references to it will be redirected to the new window. For instance, task steps will automatically be assigned to the new window or conditions like *#Age# != ""* will now read *#Birthday# != ""*.

To avoid inconveniencies, please make sure that you have selected the correct windows before using this powerful function.

7.7 Window Wizard

The **Window Wizard** is an alternative to the **GUI Spy**, if no GUI prototype of your application is available. It can be used for the creation of standard windows, saving the user a lot of time. Just use the menu '**Window|Window Wizard**', select the type of window you wish to create and the wizard will do all the work for you.

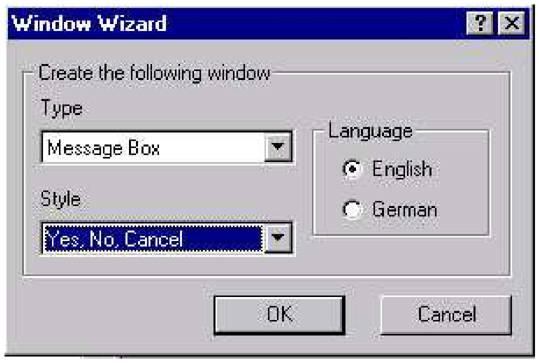


Figure 36 - Window Wizard

The Window Wizard can create the following windows:

- File Dialog
  - File Open Dialog
  - File Save Dialog
- Message Box
  - Buttons: Abort, Retry, Ignore
  - Button: OK
  - Buttons: OK, Cancel
  - Buttons: Retry, Cancel
  - Buttons: Yes, No
  - Buttons: Yes, No, Cancel
- Print Dialog

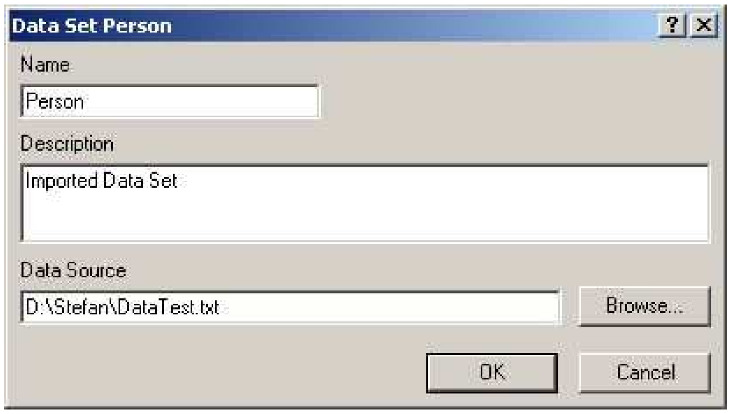
All these windows can be created with English or with German captions depending on the language of the tested system.

After pressing '**OK**', the new window is displayed in the **Window Editor**.

8 Editors for Defining the Data Model

8.1 Data Set Editor

The **Data Set Editor** opens when a new data set is created or when an existing data set is selected and the user presses the tool bar button '**Edit Properties**'.



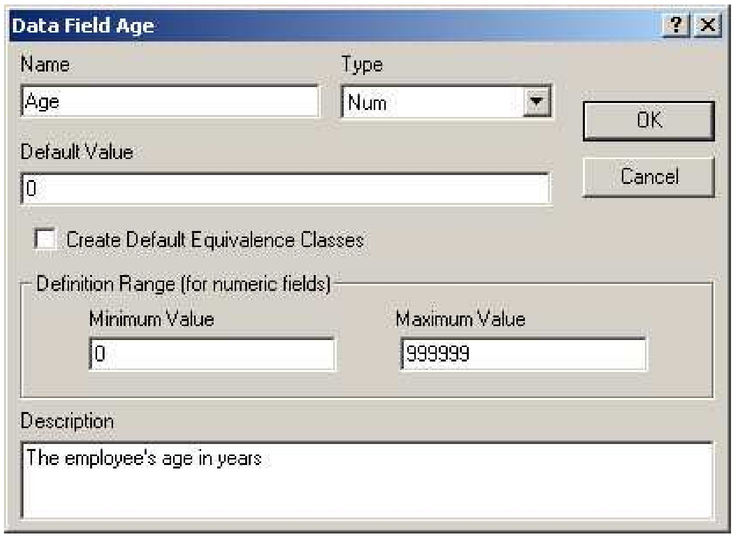
Name: Person
Description: Imported Data Set
Data Source: D:\Stefan\DataTest.txt

Figure 37 - Data Set Editor

- The **Name** must be unique, may only contain alphanumeric characters and '_' and be up to 30 characters long.
- An arbitrary **Description** may be entered for the data set.
- If test data has been imported from an external file, the field **Data Source** contains the path to the file. It can be changed using the **Browse…** button (this has no effect on the test data).

8.2 Data Field Editor

The **Data Field Editor** opens when a new data field is created or when an existing data field is selected and the user presses the tool bar button '**Edit Properties**'.



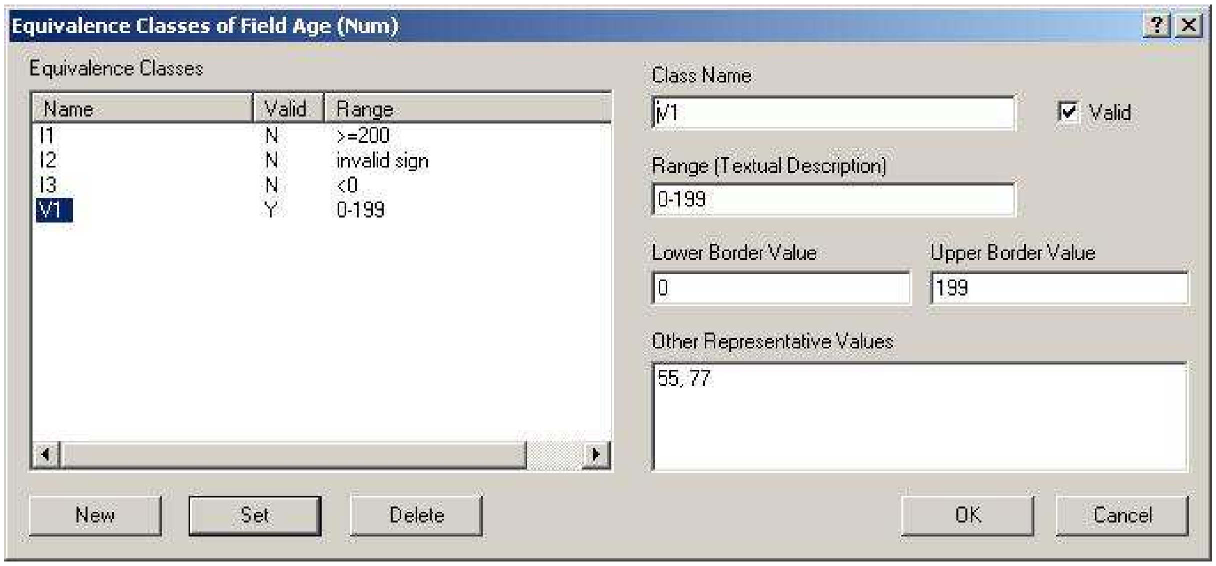
Name: Age
Type: Num
Default Value: 0
☐ Create Default Equivalence Classes
Minimum Value: 0
Maximum Value: 999999
Description: The employee's age in years

Figure 38 - Data Field Editor

- The **Name** must be unique, may only contain alphanumeric characters and '_' and be up to 30 characters long.
- A data field has one of the **Types** String, Num, Bool or Date.
- The **Default Value** will be used for this field when a new data record is created.
- If **Create Default Equivalence Classes** is checked, IDATG automatically creates a set of default equivalence classes depending on the type of the field.
- For numeric fields, the **Definition Range** has to be entered. This is the range of values that a field can theoretically assume, including invalid ones. For example, it may be possible to enter any 3 digits into an "Age" field (0-999) even if only ages above 18 are considered valid. Details regarding the validity are defined in the **Equivalence Class Editor** or **Data Effects Editor**.
- An arbitrary **Description** may be entered for the data field.

8.3 Equivalence Class Editor

The **Equivalence Class Editor** opens when the user double-clicks on the *Equivalence_Classes* item that can be found below each data field in the data hierachy tree. For detailed information on simple equivalence classes and how to work with them please refer to User's Guide Part I.



Name	Valid	Range
I1	N	>=200
I2	N	invalid sign
I3	N	<0
V1	Y	0-199

Class Name: V1
Valid: ✓
Range (Textual Description): 0-199
Lower Border Value: 0
Upper Border Value: 199
Other Representative Values: 55, 77

Figure 39 – Equivalence Class Editor

Creating a new equivalence class

- Click the '**New**' button.
- The **Class Name** must be unique, may only contain alphanumeric characters and '_' and be up to 30 characters long.
- Please choose if the class contains **Valid** or invalid values.
- The **Range** of the equivalence class can be described as an arbitrary string (examples: *0-100, a-z, prime numbers, English words*)
- If applicable, a **Lower** and **Upper Border Value** can be defined. These values can later be used by the data generator to create boundary value test cases.
- In addition, one or more **Other Representative Values** can be entered using the following syntax:
  - Values are separated by commas, e.g. *value1, value2, value3*
  - Empty strings or strings containing blanks can be defined by enclosing them in quotes, e.g. *"", "   StringWithLeadingBlanks"*

  - If a comma, quote, or any other character is preceded by a backslash, the backslash is ignored and the character interpreted as part of the value, e.g. *val\"ue1, val\,ue2, val\\ue3*
- Click the '**Set**' button. The following rules are checked:
  - At least one value (either a border value or a representative value) must be entered
  - If **Valid** has been checked, all values must be compatible with the type of the data field
  - Each value must be unique for the entire data field

Editing an equivalence class

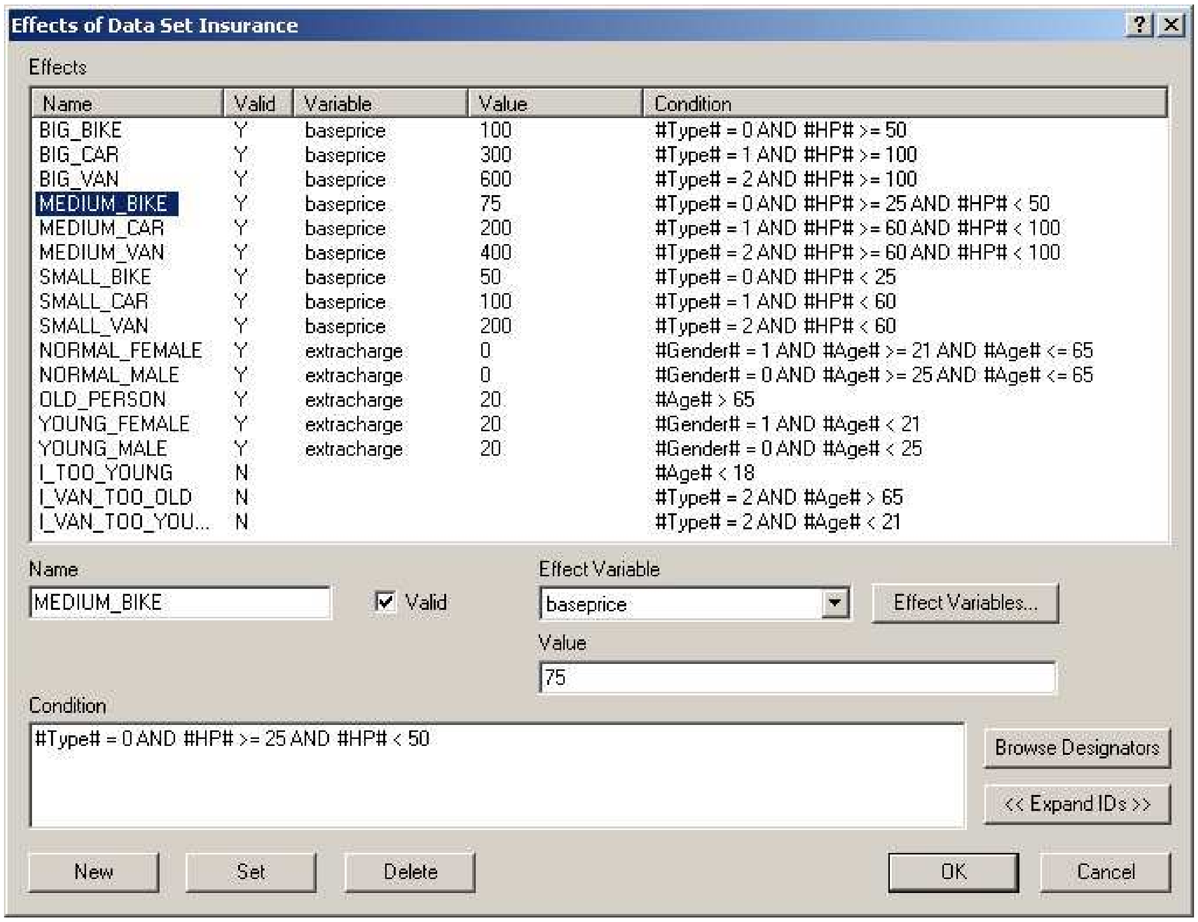
- Select a class in the list.
- Edit the data
- Click the '**Set**' button. The cursor automatically moves to the next row.

Deleting an equivalence class

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

8.4 Data Effects Editor

The semantic dependencies (causes and effects) of a data set can be edited by selecting the tree item *_Effects* in the workspace and pressing the tool bar button '**Edit Properties**'. For detailed information on effects and how to work with them please refer to User's Guide Part I.


Effects of Data Set Insurance

Effects

Name	Valid	Variable	Value	Condition
BIG_BIKE	Y	baseprice	100	#Type# = 0 AND #HP# >= 50
BIG_CAR	Y	baseprice	300	#Type# = 1 AND #HP# >= 100
BIG_VAN	Y	baseprice	600	#Type# = 2 AND #HP# >= 100
MEDIUM_BIKE	Y	baseprice	75	#Type# = 0 AND #HP# >= 25 AND #HP# < 50
MEDIUM_CAR	Y	baseprice	200	#Type# = 1 AND #HP# >= 60 AND #HP# < 100
MEDIUM_VAN	Y	baseprice	400	#Type# = 2 AND #HP# >= 60 AND #HP# < 100
SMALL_BIKE	Y	baseprice	50	#Type# = 0 AND #HP# < 25
SMALL_CAR	Y	baseprice	100	#Type# = 1 AND #HP# < 60
SMALL_VAN	Y	baseprice	200	#Type# = 2 AND #HP# < 60
NORMAL_FEMALE	Y	extracharge	0	#Gender# = 1 AND #Age# >= 21 AND #Age# <= 65
NORMAL_MALE	Y	extracharge	0	#Gender# = 0 AND #Age# >= 25 AND #Age# <= 65
OLD_PERSON	Y	extracharge	20	#Age# > 65
YOUNG_FEMALE	Y	extracharge	20	#Gender# = 1 AND #Age# < 21
YOUNG_MALE	Y	extracharge	20	#Gender# = 0 AND #Age# < 25
I_TOO_YOUNG	N			#Age# < 18
I_VAN_TOO_OLD	N			#Type# = 2 AND #Age# > 65
I_VAN_TOO_YOU...	N			#Type# = 2 AND #Age# < 21

Name: MEDIUM_BIKE
[✓ Valid]
Effect Variable: baseprice
Value: 75
Condition: #Type# = 0 AND #HP# >= 25 AND #HP# < 50

Figure 40 - Data Effects Editor

Creating a New Effect

- Click the '**New**' button.
- The **Name** must be unique, may only contain alphanumeric characters and '_' and be up to 30 characters long.
- Effect **Conditions** are slightly different from other conditions in IDATG:
  - They may only refer to data fields of the current data set and must not contain any other designators like global variables and the like.
  - References to data fields can simply be written in the format *#FieldName#* instead of *#~SetName:FieldName#* thus making the conditions both easier to write and understand.
  - While meta-conditions may not be used, it is possible to refer to effects that have already been defined for this data set using the format *%EffectName*. For instance, you may write *#Age# > 65 AND NOT (%MEDIUM_BIKE)*. You can use the '**Expand IDs**' button to view the complete condition without effect names.
  - Instead of learning these rules by heart, it is far more convenient to use the button '**Browse Designators**' for inserting designators into the condition.
  - All conditions must be **linear** in nature. This means that it must be possible to transform them into the form "*a*#var1# + b*#var2# + … < n*" where a,b,.. n are constant numbers. In particular, it is not allowed to multiply or divide two variables (*5 * #Age# < #HP#* is allowed, *#HP# * #Age# < 5* is not).
- Specify if data sets fulfilling the condition are **Valid** or not.
- Each valid effect is supposed to set an **Effect Variable** to a certain **Value**. An effect variable can be seen as a provisional result and is useful for expressing semantic dependencies. You can define the effect variables of a data set in the **Attributes Editor** that can be reached with the '**Effect Variables…**' button.
- Press '**Set**' ➔ the condition parser is called to check the condition for syntactic correctness. If an error occurs, a message box with a detailed description of the error appears and the focus moves to '**Condition**'.

Editing an Effect

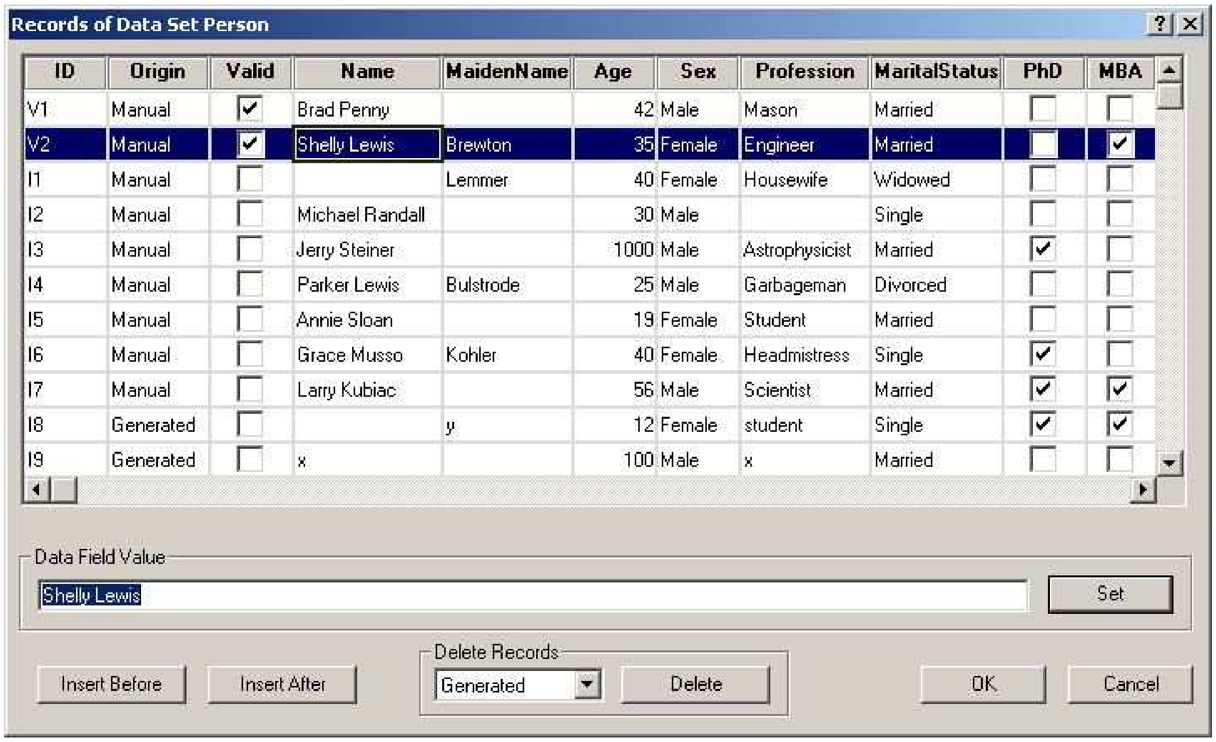
- Select an effect in the list.
- Edit the data
- Click the '**Set**' button.

Deleting an Effect

- Select an effect in the list.
- Press '**Delete**'. A message box appears that has to be confirmed by the user.

8.5 Data Records Editor

The records of a data set can be edited by selecting the tree item _*Records* in the workspace and pressing the tool bar button '**Edit Properties**'.



ID	Origin	Valid	Name	MaidenName	Age	Sex	Profession	MaritalStatus	PhD	MBA
V1	Manual	✓	Brad Penny		42	Male	Mason	Married	☐	☐
V2	Manual	✓	Shelly Lewis	Brewton	35	Female	Engineer	Married	☐	✓
I1	Manual	☐		Lemmer	40	Female	Housewife	Widowed	☐	☐
I2	Manual	☐	Michael Randall		30	Male		Single	☐	☐
I3	Manual	☐	Jerry Steiner		1000	Male	Astrophysicist	Married	✓	☐
I4	Manual	☐	Parker Lewis	Bulstrode	25	Male	Garbageman	Divorced	☐	☐
I5	Manual	☐	Annie Sloan		19	Female	Student	Married	☐	☐
I6	Manual	☐	Grace Musso	Kohler	40	Female	Headmistress	Single	✓	☐
I7	Manual	☐	Larry Kubiac		56	Male	Scientist	Married	✓	✓
I8	Generated	☐		y	12	Female	student	Single	✓	✓
I9	Generated	☐	x		100	Male	x	Married	☐	☐

Data Field Value: Shelly Lewis

Delete Records: Generated

Figure 41 - Data Records Editor

- The first column of the grid always contains the record **ID**. The ID must be unique for this data set, may only contain alphanumeric characters and '_' and be up to 30 characters long. It is recommended to use 'V' for valid and 'I' for invalid data records as the first character. The record ID can be referenced using the designator *#DataSet:$id#*.

- The second column shows information about the **Origin** of the record. A record may either be created *manually*, *imported* from an external file or generated with the *Equivalence Class Method* or *CECIL Method*.

- Only if the list contains data sets generated with the CECIL method, there are 2 additional columns that show the **Effects** based on which the data sets that have been generated and the value of the **Effect Variables** (Vars).

- Another column contains a check box that defines whether the record is **Valid** or not. This setting can be referenced in conditions using the designator *#DataSet:$valid#*.

- The other columns represent the data fields of the set.

Moving the Cursor

The cell cursor may be moved using the following keys:

- Tab = right, Shift-Tab = left
- CrsrUp = up, CrsrDown = down
- PgUp = first record, PgDown = last record

Creating a New Record

- If you wish to insert a record at a certain position, select an existing record.
- Press '**Insert Before**' or '**Insert After**'.
- Depending on which button you pressed, a new line is added before or after the selected one. All fields are initialized with their default values.
- It is impossible to leave the first column until you entered a valid ID.

Editing a Record

- Select any cell in the list.
- The value of a Boolean field can be toggled using the Space key. For all other field types, the cell value is displayed in the field "Data Field Value" and can now be edited.
- Press the Enter key or click the '**Set**' button. The new value is now visible in the table.
- The cursor automatically moves to the next row.

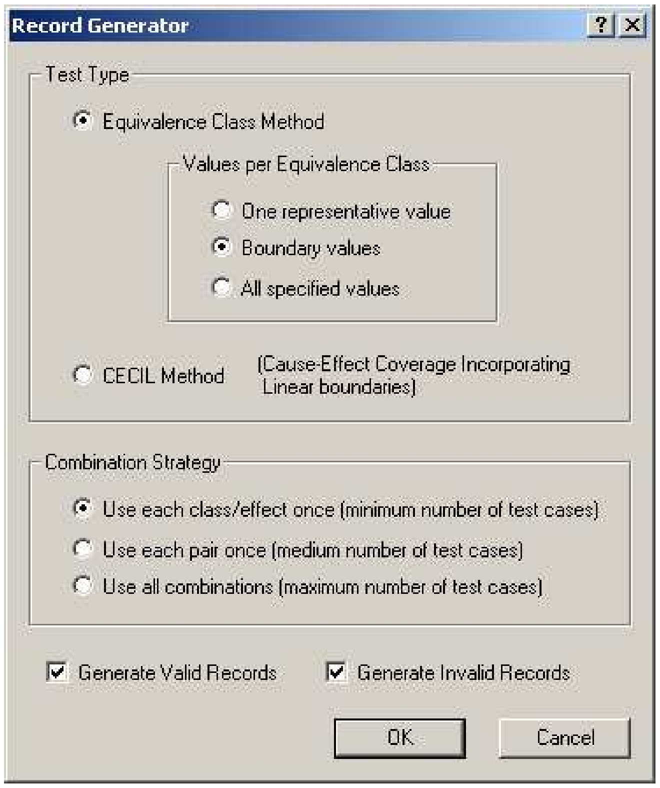
Deleting a Record

- You can either delete the currently selected record, the complete list, or all records of a certain origin. You can select the desired mode in the '**Delete Records**' combo box.
- Press '**Delete**'. A message box appears that has to be confirmed by the user.

Note: Instead of defining test data manually, it is also possible to generate it or to import it from an external file (see User's Guide Part I).

8.6 Record Generator Dialog

The **Record Generator Dialog** can be opened by selecting any of the tree items of a data set and selecting the menu item '**Generate|Generate Data Records**' or selecting the menu item '**Generate Data Records'** from the context menu (right mouse click).


Record Generator

Test Type
[✓ Equivalence Class Method]
Values per Equivalence Class
One representative value
[✓ Boundary values]
All specified values
CECIL Method (Cause-Effect Coverage Incorporating Linear boundaries)

Combination Strategy
[✓ Use each class/effect once (minimum number of test cases)]
Use each pair once (medium number of test cases)
Use all combinations (maximum number of test cases)

[✓ Generate Valid Records] [✓ Generate Invalid Records]

Figure 42 - Record Generator Dialog

Note: Before generating data records you have to define equivalence classes or effects! See User's Guide Part I for more information.

The most important decision is the fundamental generation method:

- The simple '**Equivalence Class Method**' generates data records that cover the classes defined in the **Equivalence Class Editor**.
- The powerful '**CECIL Method**' generates data records based on effects defined in the **Data Effects Editor**.

The CECIL method always tries to cover all relevant (multi-dimensional) boundaries by applying sophisticated linear programming algorithms. For the equivalence class method, you can choose how many values of each equivalence class should be covered:

- **'One representative value'**: Only the first value of each class's representative value list will be taken for the generation process.
- **'Boundary values'**: When boundary values are available, they are both used for record generation. If a class has no boundary value, a representative value will be used instead.
- **'All specified values'**: All boundary and representative values of each equivalence class will be covered.

Next, the user chooses the combination strategy for the equivalence classes or effects. Note that only the first option is currently available for the CECIL method:

- **'Use each class/effect once'**: This strategy generates a **minimum** number of records in which each equivalence class or effect is used at least once.
- **'Use each pair once'**: This strategy generates a **medium** number of records. It guarantees that all possible pair-wise combinations are covered (Exception: 2 invalid classes or effects are never combined).
- **'Use all combinations'**: The last strategy generates a **maximum** number of records in which all possible combinations are covered (Exception: 2 invalid classes or effects are never combined).

Finally, the user fixes the desired record type by selecting **'Generate Valid Records'** and/or **'Generate Invalid Records'**.

- A **valid** record consists only of values that come from valid equivalence classes or effects.
- An **invalid** record includes exactly one value from an invalid equivalence class or effect, all other values come from valid ones. Combining 2 or more invalid values is not allowed.

After confirming the options with the **'OK'** button, the generation will be executed. Afterwards, a message box informs the user about the success of the operation and the number of generated records.

Note: If 1 or more of the generated records already existed before, they are not saved again. This may result in a message that 0 records have been generated.

9 Search Dialogs

9.1 Search Window Dialog

If you are searching for a particular window, switching to the '**Windows**' tab of the **Application Editor** and using the toolbar button '**Search Object**' brings you to the **Search Window Dialog**. If a window was selected before calling the dialog, the scope of the search is limited to this window and its children unless you check '**Whole Application**'.

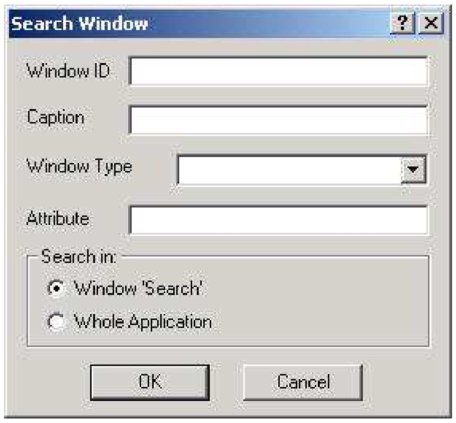


Figure 43 - Search Window Dialog

You can combine different search criteria: window ID, caption, window type and the name of a user-defined attribute that belongs to the window. It suffices to enter part of the ID/Caption/Attribute Name. For example, all radio buttons whose ID contains *ma* can be searched. After pressing '**OK**', all matching windows are highlighted in the **Application Editor**. (e.g., *Ma*le, *Ma*rried, E*ma*il etc.)

9.2 Search Task Dialog

If you are searching for a particular task, switching to the '**Tasks**' tab of the **Application Editor** and using the toolbar button '**Search Object**' brings you to the **Search Task Dialog**. If a task was selected before calling the dialog, the scope of the search is limited to this task and its children unless you check '**Whole Application**'.

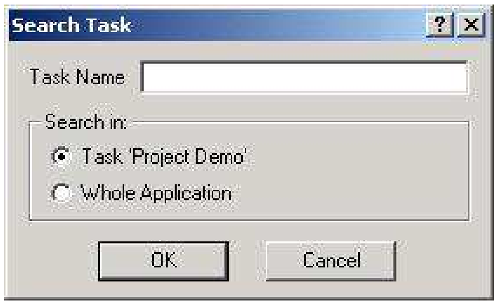


Figure 44 - Search Task Dialog

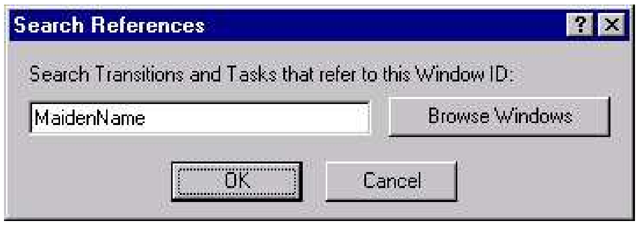
It suffices to enter part of the Task Name. For example, all tasks whose name contains *crea* are searched. After pressing '**OK**', all matching tasks are highlighted in the **Application Editor**. (e.g., *Crea*te_Person, Feed_*Crea*ture etc.)

9.3 Search References Dialog

This function is particularly useful for finding:

- References to a task in task steps (i.e. where has the task been used as sub task)
- References to a requirement in tasks (i.e. to which tasks a certain requirement has been assigned)
- References to a window in conditions, actions or task steps

Just use the menu '**Edit|Search References**'. If you selected a **task**, all of its super tasks are highlighted in the **Application Editor**. If you selected a **requirement**, all tasks to which that requirement has been assigned are highlighted in the **Application Editor**. If you selected a **window** or didn't select anything, the **Search References Dialog** appears:

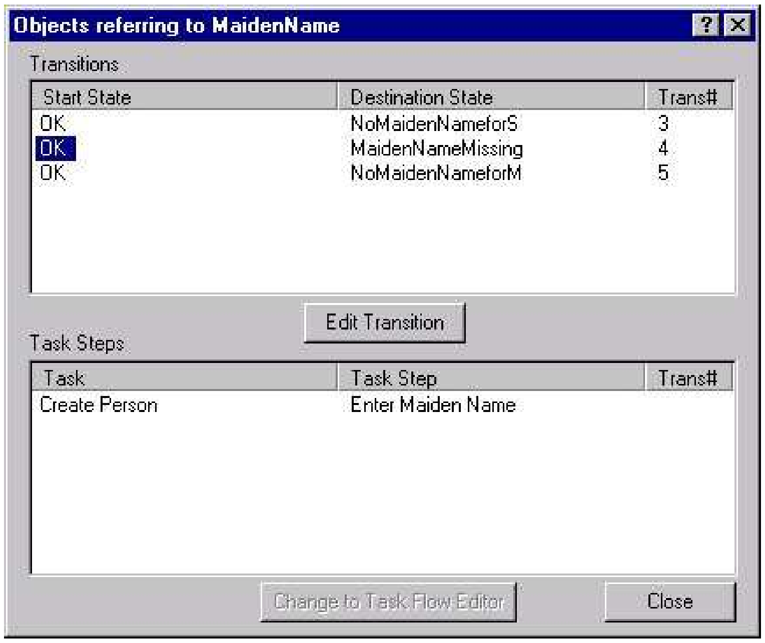

Search References

Search Transitions and Tasks that refer to this Window ID:

MaidenName

Figure 45 - Search References Dialog

If a window was selected before calling the dialog, its ID is used as default value. You may either use the '**Browse Windows**' button to select a window or enter its ID manually (Useful if the window has already been deleted, but is still referenced). Please note that the complete ID has to be entered. After pressing OK, the search results are displayed in the following dialog:


Objects referring to MaidenName

Transitions

Start State	Destination State	Trans#
OK	NoMaidenNameforS	3
OK	MaidenNameMissing	4
OK	NoMaidenNameforM	5

Task Steps

Task	Task Step	Trans#
Create Person	Enter Maiden Name	

Figure 46 - Found References Dialog

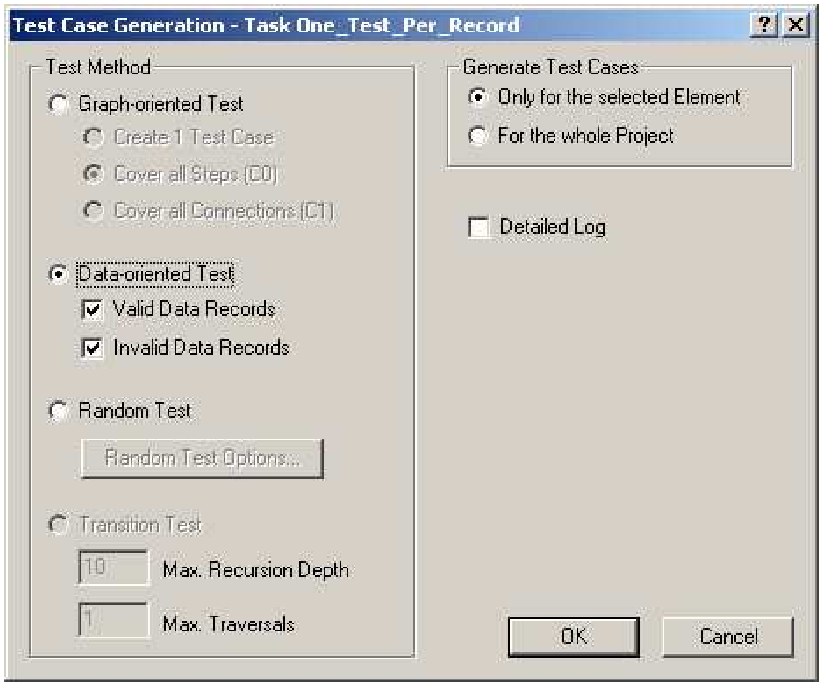
The first list contains all transitions with conditions or actions referring to the window ID. If you want to change a transition immediately, select it from the list and press '**Edit Transition**'. The **Step Editor** appears where you may perform the desired changes.

The second list contains all task steps with events, conditions or actions referring to the window ID. After selecting an item from the list, you can open the corresponding **Task Flow Editor** by clicking on '**Change to Task Flow Editor**'. Please note that in this case the **Found References Dialog** has to be closed automatically.

10 Test Case Administration

10.1 Test Case Generator

After the toolbar button '**Generate Test Cases**' is pressed, the **Test Case Generation Dialog** appears. You may either generate test cases for the currently selected element (a use case task, window, or transition) or for the whole application. If you select a task folder or window, test cases are also generated for its children (recursively). Graph- and data-oriented test cases are always named after the use case task they refer to, transition test cases after the transition.


Test Case Generation - Task One_Test_Per_Record

Test Method
- Graph-oriented Test
  - Create 1 Test Case
  - [✓ Cover all Steps (C0)]
  - Cover all Connections (C1)
- [✓ Data-oriented Test]
  - [✓ Valid Data Records]
  - [✓ Invalid Data Records]
- Random Test
  - Random Test Options...
- Transition Test
  - 10 Max. Recursion Depth
  - 1 Max. Traversals

Generate Test Cases
- [✓ Only for the selected Element]
- For the whole Project

Detailed Log

OK Cancel

Figure 47 - Test Case Generation Dialog

Graph-oriented Test

- **Create 1 Test Case** lets IDATG stop the generation after finding the first valid path. This option is useful for tasks that have many unreachable branches.
- **Cover all Steps** causes IDATG to generate test cases for the selected use case task. The aim is to cover all steps of the task flow (C0 coverage).
- **Cover all Connections** causes IDATG to generate test cases for the selected use case task. The aim is to cover all connections between the steps of the task flow (C1 coverage).

Data-oriented Test

This method is only available, if the selected task has been assigned to a data set (see User's Guide Part I). It should only be used if you want to generate a separate test case for each data record in a set. If you want to use more than one record in the same test case, check '**Graph-oriented Test**' instead. You can select one or both of the following options:

- **Valid Data Records** causes IDATG to generate a test case for each valid record of the data set.
- **Invalid Data Records** causes IDATG to generate a test case for each invalid record of the data set.

Random Test
If this option is chosen, IDATG will try to create random sequences of building blocks. In order to avoid that invalid sequences are generated, you have to specify conditions and actions very thoroughly in the model.
The button '**Random Test Options**' allows you to specify coverage criteria and other options. (See User Guide part I for more info)

Transition Test
This option causes IDATG to generate one test case for each specified transition. The advanced options affect the test case generation algorithm. They should only be changed by experienced users.

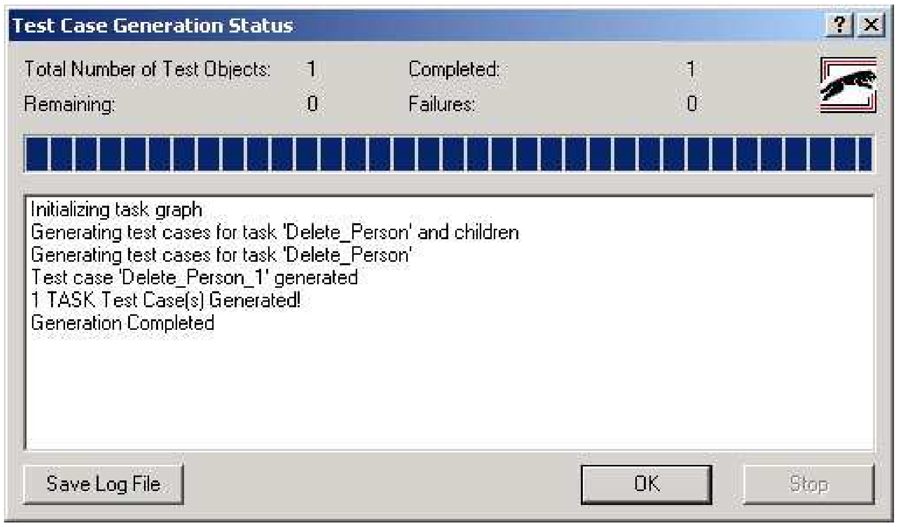
Max. Recursion Depth: This value fixes the maximum number of test steps in a test case. A lower number means less execution time but also a lower chance of finding a solution. The value may range from 1 to 30. The default value is 10.

Max. Traversals: This value fixes the maximum number of generation trials for control values (e.g. input field values). Again, a lower number means less execution time but also a lower chance of finding a solution. The value may range from 1 to 10.

Detailed Log: If checked, detailed information about the generation process is displayed in the **Generation Status Dialog**. This option can help to determine the cause for generation failures. However, the amount of displayed information can become very large.

10.2 Generation Status Dialog

During the generation a dialog informs about the progress of the procedure. Additional info is displayed if ‘**Detailed Log**’ has been checked in the previous dialog. This is particularly useful for determining the cause of a generation failure, but may result in a lot of technical information that may be confusing for the common user.



Total Number of Test Objects: 1   Completed: 1
Remaining: 0   Failures: 0

```
Initializing task graph
Generating test cases for task 'Delete_Person' and children
Generating test cases for task 'Delete_Person'
Test case 'Delete_Person_1' generated
1 TASK Test Case(s) Generated!
Generation Completed
```

Figure 48 - Generation Status Dialog

Stopping the Generation
If you are working with a very complex specification, the generation process may take some time. By pressing ‘**Stop**’ you can quit the generation without having to kill the IDATG

process. As soon as the generator detects that the user has pressed the button, it performs a “clean” exit and saves all test cases generated up to this point.
Note: It is VERY unlikely that the generation algorithm crashes or reaches an endless loop. If the generation process does not respond for a long time it is usually a sign that you specified a lot of conditions that cannot always be fulfilled and thus require a lot of backtracking.

Saving the Logging Information

The displayed text can be saved to a file by pressing ‘**Save Log File**’. The file is named “*Logfile.txt*” and is placed in the *My Documents\Idatg\Logs* directory. Often the cause of a generation failure can be determined by analyzing this file. If you are not sure how to interpret the data yourself, you may find help by sending the file to the IDATG support team.

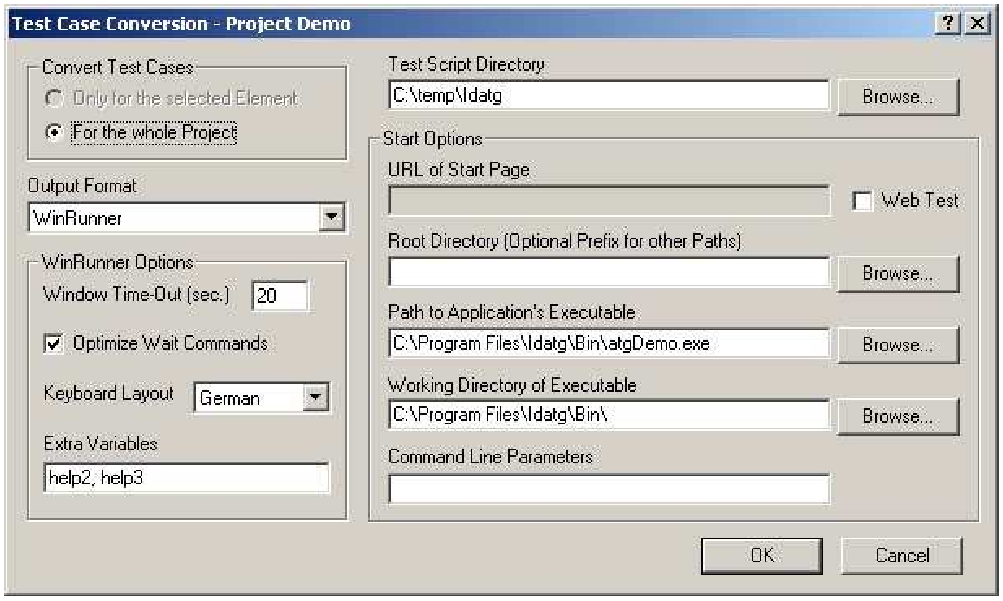
Generation of Verify Commands

During the generation, IDATG adds special verify commands to the generated test script:

- For transition test cases, it is checked, whether the input focus is really on the expected destination state after the tested transition.
- After certain actions the expected effects are verified. This refers to *SetAttribute (#WindowID:$Enabled#, Value), SetAttribute (#WindowID:$Visible#, Value), SetAttribute (#WindowID:$Caption#, Value)* and all actions that change the contents of a window (e.g. *SetInputfield(#WindowID#, Value))*.

10.3 Test Case Converter

After the toolbar button ‘**Convert Test Cases**’ is pressed, the **Test Case Conversion Dialog** appears. You may either convert test cases for the currently selected window/use case task or for the whole application. The converted test cases are saved as scripts that can be executed by the selected test tool.


Test Case Conversion - Project Demo

Convert Test Cases: Only for the selected Element [✓ For the whole Project]

Output Format: WinRunner

WinRunner Options
Window Time-Out (sec.): 20
Optimize Wait Commands: ✓
Keyboard Layout: German
Extra Variables: help2, help3

Test Script Directory: C:\temp\Idatg

Start Options
URL of Start Page:
Web Test: ☐
Root Directory (Optional Prefix for other Paths):
Path to Application's Executable: C:\Program Files\Idatg\Bin\atgDemo.exe
Working Directory of Executable: C:\Program Files\Idatg\Bin\
Command Line Parameters:

Figure 49 - Test Case Conversion Dialog

Since it would be unwise to generate a separate script for each GUI object, scripts are only generated for windows that have a titlebar of their own. These scripts also contain the test cases for all children of the window (unless a child has its own titlebar). A separate script is also generated if a window does not have a parent.

The following options can be set:

Output Format: Test cases can be exported in various formats (for details, see User's Guide Part I). Initially, the default output format that you specified in the **Project Administration Dialog** is selected. If you want to convert test cases for a different tool, you have to generate/enter proper window tags first! Otherwise the windows will not be recognized by the test tool.

Tool-specific Options: Depending on the **Output Format**, various options specific to the selected tool may appear. For a detailed description of these options, please refer to the relevant tool section in User's Guide Part I.

Test Script Directory: The directory, where the converted test scripts should be placed. Use the '**Browse...**' button to select a directory. The default directory is *My Documents\Idatg\Scripts* or the TEMPPO Test Manager installation directory in case you selected the output format for TEMPPO Test Manager.

WebTest: Check this if you wish that the Internet Explorer should be started automatically when running the scripts.

URL of Start Page: The internet address of the webpage you want to test (e.g., http://www.google.com). Only active if you checked '**WebTest**'.

Root Directory (optional): Here you can enter a prefix for the following two fields (exe path and working directory). This may be useful if the same project is tested on various machines with differing directory structures.

Path to Application's Executable: The path to the executable file of the tested application. Use the '**Browse...**' button to select an executable file.

Working Directory: The default directory of the tested application. IDATG automatically suggests the path to the executable.

Command Line Parameters: Here you can enter parameters that you want to pass to the tested application (like in a DOS command line).

10.3.1 Missing Tags

Before converting test cases it is important that all concerned windows have tags that are recognized by the selected test tool. You can use the **Tag Generator** to let IDATG find these tags for you. However, it is possible that some tags remain empty. In this case an information dialog is shown that contains the IDs of all windows with empty tags.

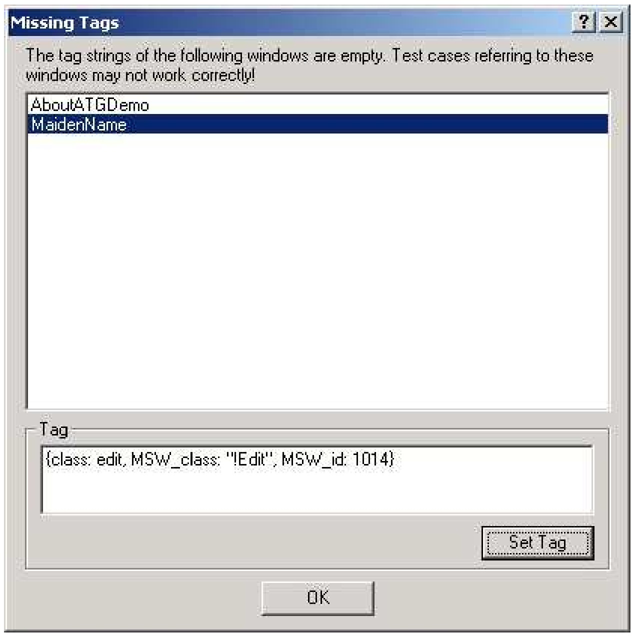

Missing Tags
The tag strings of the following windows are empty. Test cases referring to these windows may not work correctly!
AboutATGDemo
MaidenName
Tag
{class: edit, MSW_class: "!Edit", MSW_id: 1014}
Set Tag
OK

Figure 50 - Missing Tags Dialog

For your convenience, it is not necessary to call the **Window Properties Editor** for each listed window; you can set the tag immediately in the information dialog. Just select the window in the list, enter its tag and press '**Set Tag**'. After that the window ID disappears from the list. Press '**OK**' if you have entered all relevant tags to get to the **Test Case Converter**.

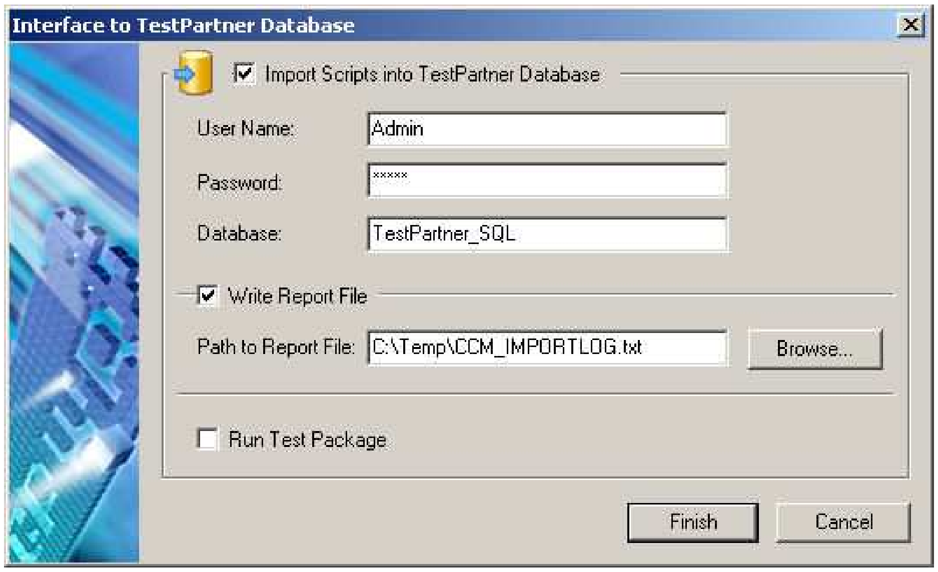
Note: You only have to set the tags of windows that appear in test cases, all others may remain empty.

10.3.2 Interface to TestPartner Database

When choosing ”TestPartner” as output format in the **Test Case Converter** dialog, the conversion is done in two steps:

1. The converted test scripts are saved as TestPartner-compatible XML files in the specified test script directory.
2. The scripts can be imported automatically into the TestPartner database.

After step 1, the TestPartner Database Dialog appears. If you have a working installation of MicroFocus TestPartner on your machine, you can define in this dialog whether to import the converted test cases automatically into TestPartner's database. Furthermore, IDATG can force the converted test cases to be executed by TestPartner after the import process.



☑ Import Scripts into TestPartner Database

User Name: Admin

Password: *****

Database: TestPartner_SQL

☑ Write Report File

Path to Report File: C:\Temp\CCM_IMPORTLOG.txt

☐ Run Test Package

Figure 51 – TestPartner Database Dialog

Import Scripts into TestPartner Database: Enables automatic test script import. If this option is checked, the following fields have to be specified:

User Name: User name of your TestPartner login
Password: Password of your TestPartner login
Database: The DSN of the database in which your project assets are to be stored

Note: Usually, these three fields contain the same data that you use in the login dialog of TestPartner.

Write Report File: If checked, a detailed report of the import process is created

Path to Report File: Specifies the location of the report file.

Run Test Package: If checked, the test package (i.e., every single test case currently converted) is executed by TestPartner automatically after the import.

Click on the **Finish** button to complete this step and start the import process of the converted test package(s). A shell window pops up and a TestPartner-command line tool is executed in order to start the import process. Depending on your system settings, the shell window immediately closes after execution of the import tool, making it difficult to read possible error messages. Therefore it is helpful to use the option **Write Report File** that saves all shell output to a log file and lets you identify the reason for possible errors during the import process.

Important Notes:

- In order to import and/or automatically run test scripts, it is not necessary to have MicroFocus TestPartner currently running on your machine.
- If TestPartner is running during the import process, make sure to **close all open test script windows** in TestPartner before importing. The import will fail for test scripts for which older versions with the same name are currently opened in TestPartner.
- Depending on your TestPartner settings, TestPartner saves and maybe automatically shows test results after a test has been finished or interrupted because of an error. If TestPartner is not running during the Auto-Run of your imported test suite, you will have to start it on your own to show the recent results of your test run.

10.4 Test Case Editor

After generating test cases for a certain object you can view the finished test cases by pressing the toolbar button '**View Test Cases**'. If there is only one test case for the selected object, the **Test Case Editor** opens immediately. Otherwise a list of all available test cases opens first from which you can choose the test case you want to view.

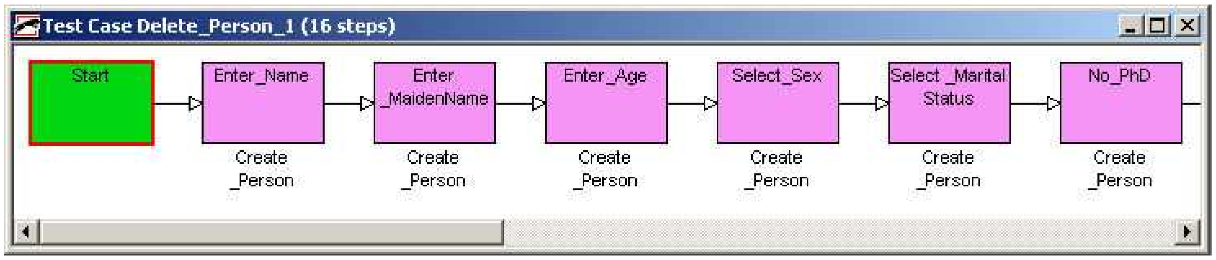


Figure 52 - Test Case Editor

Test cases are displayed very similar to tasks in the **Task Flow Editor**. However, there are some differences:

- A test case may not contain branches, therefore it is not possible to change the connections between the steps.
- For task-based test cases, it is helpful to know which task steps are concerned. Since the step names are not always unique, the task names are written below the steps, too. Note that a test case usually contains steps from different tasks.
- If a test step refers to a transition, this means that it has the same properties as the transition. However, the link is much weaker as between task steps and transitions. Remember that changing a transition also means changing all **task** steps that refer to it. However, changing a transition does not change any **test** steps and vice versa.
- The event of a test step must only contain concrete values. Designators are not allowed.
- Test steps do not require conditions or actions.
- Editing a test case does not make much sense, since it will probably be overwritten the next time you generate test cases. Nevertheless, it is possible to insert and delete steps and also to edit them in the **Step Editor**.

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 border.

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 test case. 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 at the end of the Test Case

- Select the last step.
- Press '**Insert Step**'
- An undefined step is created at the end of the test case

Adding a new Step between two Steps

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

Editing a Step

- Double click on the step. The **Step Editor** appears and allows you to change the step properties.

Assigning an Object to a Step

Remember that all steps have to be assigned to a window or transition before you can generate a test case.

- Select the step
- Press the toolbar button '**Assign Object to Step**'
- All other editors are now in a special waiting state. Instead of performing their usual functions, the next selection is sent directly to the **Test Case Editor**. You may either select a window in the **Application Editor** or a window or transition in a **Window** or **Menu Editor**.
- If the assignment has been successful, the color of the step changes accordingly.

Viewing the Object assigned to a Step

If you want to view the window or transition that has been assigned to a certain step, just select the step and press the toolbar button '**Edit Window**' or '**Edit Menu**'.

Finding a Task in the Task Hierarchy

If you want to find the task assigned to a test step in the **Task Hierarchy**, simply right-click on the test step and select '**Show in Task Tree**'.

Deleting Steps

The selected steps may be deleted by choosing the menu '**Delete**' or by pressing the corresponding toolbar button. The successor of the steps will be connected to their former predecessor to avoid that it becomes isolated. The start step cannot be deleted.

10.5 Window Tag Generator

Capture-replay tools need a tag string for each window in order to be able to address it. Unfortunately these tools use very differing algorithms that are besides often not easy to comprehend. For example, if the user has finally managed to write all tags in SilkTest style and later decides to use WinRunner, he would have to change hundreds of window tags manually. However, IDATG provides a solution for this great problem. The menu item '**Generate|Generate Window Tags**' opens the **Tag Generator** that is able to do most of the work for you.
Note that tags are generated automatically by IDATG after recording windows with the GUI Spy or importing .idj files so you do not have to call the **Tag Generator** explicitly in these cases. An explicit call of the **Tag Generator** is only necessary if you have inserted windows manually or if you have decided to change your test execution tool.

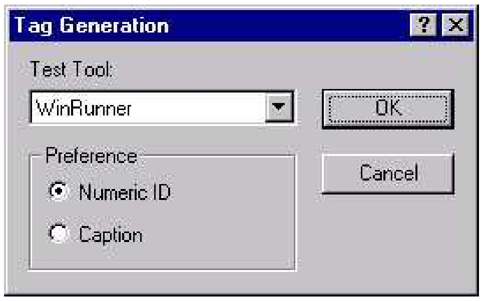


Figure 53 - Tag Generator

Note: Tags are always generated for the currently selected window and its children.

Preference:
The default setting is that IDATG prefers the **numeric ID** when generating a tag. This has the advantage that the window will also be recognized if the caption changes or a version for a different language should be tested.
If you wish that the **caption** should be used instead, check the corresponding button.

After selecting a test tool from the list and pressing '**OK**', IDATG launches a sequence of refined algorithms that try anything possible to generate correct tags for this tool.

Despite of all these efforts it is unavoidable in some cases that the user has to edit the tags manually. Especially user-defined windows and windows without a caption may be a problem. In this case, please refer to User's Guide Part I.