

Telelogic

**Tau<sup>®</sup>**

**C# Tutorial**



**IBM<sup>®</sup>**



*Tau*®

**C# Tutorial**



This edition applies to Telelogic Tau version 4.2 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2008.

US Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

## Copyright Notice

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

Copyright © 2008 by IBM Corporation.

## IBM Patents and Licensing

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to the following:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software|  
IBM Corporation  
1 Rogers Street  
Cambridge, Massachusetts 02142  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

### **Disclaimer of Warranty**

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

## **Confidential Information**

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Additional legal notices are described in the legal\_information.html file that is included in your software installation.

## **Sample Code Copyright**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

## **IBM Trademarks**

For a list of IBM trademarks, visit this Web site [www.ibm.com/legal/copytrade.html](http://www.ibm.com/legal/copytrade.html). This contains a current listing of United States trademarks owned by IBM. Please note that laws concerning use and marking of trademarks or product names vary by country. Always consult a local attorney for additional guidance. Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

Not all common law marks used by IBM are listed on this page. Because of the large number of products marketed by IBM, IBM's practice is to list only the most important of its common law marks. Failure of a mark to appear on this page does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

## **Third-party Trademarks**

Adobe, the Adobe logo, Acrobat, the Acrobat logo, FrameMaker, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

AIX and Informix are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

HP and HP-UX are registered trademarks of Hewlett-Packard Corporation.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Macrovision and FLEXnet are registered trademarks or trademarks of Macrovision Corporation.

Microsoft, Windows, Windows 2003, Windows XP, Windows Vista and/or other Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft Corporation.

Netscape and Netscape Enterprise Server are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Sun, Sun Microsystems, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Pentium is a trademark of Intel Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.





---

# Contents

Copyright license .....	v
IBM trademarks .....	v
Third-party trademarks .....	vi
<b>Getting Started .....</b>	<b>1</b>
Overview .....	1
Tutorial Objectives .....	1
Before You Begin .....	1
Documentation Conventions .....	2
About the Tau Product .....	2
Tool Tips .....	3
Starting Tau .....	3
Saving a Project in Tau .....	3
Closing Tau .....	4
Project Files and Directories .....	4
Model Element Names .....	4
Model Views .....	5
<b>Lesson 1: Starting Your Project .....</b>	<b>7</b>
Goals for this Lesson .....	7
Exercise 1: Using the Tau Developer Wizard .....	7
Task 1a: Creating a New Project and Workspace .....	7
Exercise 2: Working with Packages .....	8
Task 2a: Renaming the Default Package .....	9
Task 2b: Creating a New Package .....	9
Summary .....	9
<b>Lesson 2: Creating a Use Case Diagram .....</b>	<b>11</b>
Goals for this Lesson .....	11
Exercise 1: Creating a Use Case Diagram .....	11
Task 1a: Adding a New Use Case Diagram .....	11
Task 1b: Renaming a Use Case Diagram .....	12
Task 1c: Adding an Actor and a Use Case .....	12

## Table of Contents

---

Task 1d: Adding an Association .....	12
<b>Summary .....</b>	<b>14</b>
<b>Lesson 3: Creating an Activity Diagram .....</b>	<b>15</b>
<b>Goals for this Lesson .....</b>	<b>15</b>
<b>Exercise 1: Creating an Activity Diagram .....</b>	<b>16</b>
Task 1a: Configuring UML Settings. ....	16
Task 1b: Adding a New Activity Diagram .....	16
Task 1c: Drawing Activity Nodes. ....	17
<b>Summary .....</b>	<b>18</b>
<b>Lesson 4: Creating a Class Diagram .....</b>	<b>19</b>
<b>Exercise 1: Creating a Class Diagram .....</b>	<b>19</b>
Task 1a: Adding a New Class Diagram. ....	19
Task 1b: Drawing a Class .....	20
Task 1c: Adding Attributes and Operations. ....	20
<b>Summary .....</b>	<b>20</b>
<b>Lesson 5: Generating C# Code and More .....</b>	<b>21</b>
<b>Goals for this Lesson .....</b>	<b>21</b>
<b>Exercise 1: Generating and Editing C# Code .....</b>	<b>21</b>
Task 1a: Generating and Viewing C# Code .....	21
Task 1b: Creating a Project in Microsoft Visual Studio .....	22
Task 1c: Adding Code for the Compute Operation .....	23
Task 1d: Viewing Model Updates .....	24
Task 1e: Compiling C# Code .....	24
Task 1f: Running your Application .....	25
<b>Optional .....</b>	<b>25</b>
<b>Summary .....</b>	<b>26</b>
<b>Conclusion .....</b>	<b>27</b>
<b>Technical Support and Documentation .....</b>	<b>29</b>
<b>Contacting Technical Support .....</b>	<b>29</b>
Regional Contact Information .....	29
Tau Documentation. ....	29
<b>Index .....</b>	<b>29</b>

# Getting Started

## Overview

This tutorial teaches you the basics of working with the Tau product in a C# coding environment, and introduces the concepts of requirements analysis and project implementation. In the tutorial, you model a simple application that calculates the population growth projections of senior citizens residing in a city. The growth projections will help the city's planner determine the appropriate size of a new senior center. The calculation example used in this tutorial is based on Fibonacci numbers. For more information on Fibonacci numbers, see [http://en.wikipedia.org/wiki/Fibonacci\\_number](http://en.wikipedia.org/wiki/Fibonacci_number).

## Tutorial Objectives

When you have completed this tutorial, you will have had experience with:


- ◆ Using the Tau interface
- ◆ Creating a C# project
- ◆ Creating a use case diagram
- ◆ Creating an activity diagram
- ◆ Creating a class diagram
- ◆ Generating and editing C# code
- ◆ Integrating Tau with Visual Studio 2005
- ◆ Running and compiling a C# application

## Before You Begin

To complete the lessons in this tutorial, you must have Microsoft Visual Studio 2005 environment configured. This is because Tau integrates with the IDE for building, running and debugging C# applications. The integration must also be installed from **Program Files > Telelogic > Telelogic Lifecycle Solution Tools > Tau 4.0 > Install Microsoft Visual Studio 5005 integration**.

## Documentation Conventions

This document uses the following conventions:

- ◆ **Boldface** for names of GUI objects and controls, including selection choices; and emphasis. Examples:
  - From the **Default model view** drop-down list box, select **Standard View**.
  - Click the **Activity flow final** symbol  on the **Drawing** toolbar.
  - If the Tau browser does not display, select **View > Browser**.
  - A project file, called **<project\_name>.ttp**.
- ◆ Courier font in 10 point for pathnames, system messages, and items that you have to type. Examples:
  - The Output window displays the message `Animation session terminated`.
  - In the **Project name** box, replace the default project name with `<project name>`.
  - Type `show` for the function name, and press **Enter**.
- ◆ *Italics* for the first mention of a concept with an explanation.

## About the Tau Product

*Telelogic Tau* provides standards-based Model Driven Development™ (MDD™) of complex systems and robust software for enterprise IT applications, including those utilizing Service Oriented Architectures. Tau's iterative requirements-based approach, comprehensive error-checking, and automated simulation increase developer productivity from initial requirements through final documentation and deployment.

Tau offers a large feature set for developers to employ key enabling technologies in a natural, easy-to-use tool environment. Tau makes a seamless and efficient environment for systems, software, and testability. It enables you to perform these tasks:

- ◆ **Analyze**, during which you can define, analyze, and validate the system requirements.
- ◆ **Design**, during which you can specify and design the architecture.
- ◆ **Implement**, during which you can automatically generate code, then build and run within the Tau product.

## Tool Tips

This section describes time saving features in Tau that enable you to work more efficiently on a project. Wherever possible, the task-oriented procedures in this tutorial demonstrate the use of these features.

### Using Shortcuts

Keyboard shortcuts (**CTRL + Arrow Key**, for example) allow you to quickly navigate through the Tau interface. For a complete list of available shortcuts, see the Help topic “Editor Shortcuts.”

### Using the Auto Placement Feature

You can use Tau’s auto placement feature to quickly add a series of elements in the drawing area (**CTRL + Space Bar**). This feature is especially useful when drawing activity and state machine diagrams. For more information on this feature, see the Help topic “Add Symbols.”

## Starting Tau

### Windows

To start the Tau product in Windows, for a typical installation by selecting **Start > Programs > Telelogic > Telelogic Tau *version number***.

### Linux and Solaris

To start the Tau product on Linux and Solaris, type the following command:

```
<installation path>/bin/tau
```



## Saving a Project in Tau

To save a project in Tau, on the main menu bar, select **File > Save All**. To configure Tau to automatically save changes to your project, follow these steps.

1. On the main menu bar, select **Tools > Options**.
2. On the **Save** tab, in the Auto-backup panel, select the **Activate** checkbox and specify an interval (every 5 minutes, for example.)
3. Click **OK**.

## Closing Tau

To exit the Tau product, follow these steps.

1. Save your work. Do one of the following:
  - ◆ Press **CTRL+S**.
  - ◆ Click the Save button  to save your work.
  - ◆ **File > Save All** especially if many files were edited.
2. Choose **File > Exit** or click the **Close** button .

### Note

---

A red bar on the lefthand side of any file in your workspace indicates that you need to save your work before you exit Tau.

## Project Files and Directories

The Tau product creates the following files and subdirectories in the project directory:

- ◆ A project file, called **<project\_name>.ttp** contains references to all files, model files (\*.u2) and addins used in the project.
- ◆ A workspace, called **<project\_name>.ttw** in which enables users to reference many projects.
- ◆ Model files, called **<file\_name.u2>** that contain the unit files for the project, including UML diagrams, packages, use cases, code generation configurations and other granularity of UML elements.

## Model Element Names

It is recommended that the names of model elements should follow some conventions, such as these:

- ◆ Class names begin with an upper case letter, such as “System.”
- ◆ Operations and methods begin with lower case letters, such as “restartSystem.”
- ◆ Upper case letters to separate concatenated words, such as “checkStatus.”

## Model Views

Tau enables you to construct a model in several views, each representing different abstract characteristics of your model. For detailed information on model views, see the Help topic “Views.”





# Lesson 1: Starting Your Project

When you create a C# project in Tau, it contains UML diagrams as well as libraries, and add-ins. Tau creates a directory containing the *project files* in a specified location. The name you choose for your new project is used to name project files and directories, and it appears at the top level of the project hierarchy in the Tau browser.

## Goals for this Lesson

In this lesson, you create a new C# project in Tau, configure a project workspace, and add the necessary packages for the exercises in this tutorial. You will learn about the following concepts:

- ◆ Project configuration settings
- ◆ Project workspaces
- ◆ Project directories

## Exercise 1: Using the Tau Developer Wizard

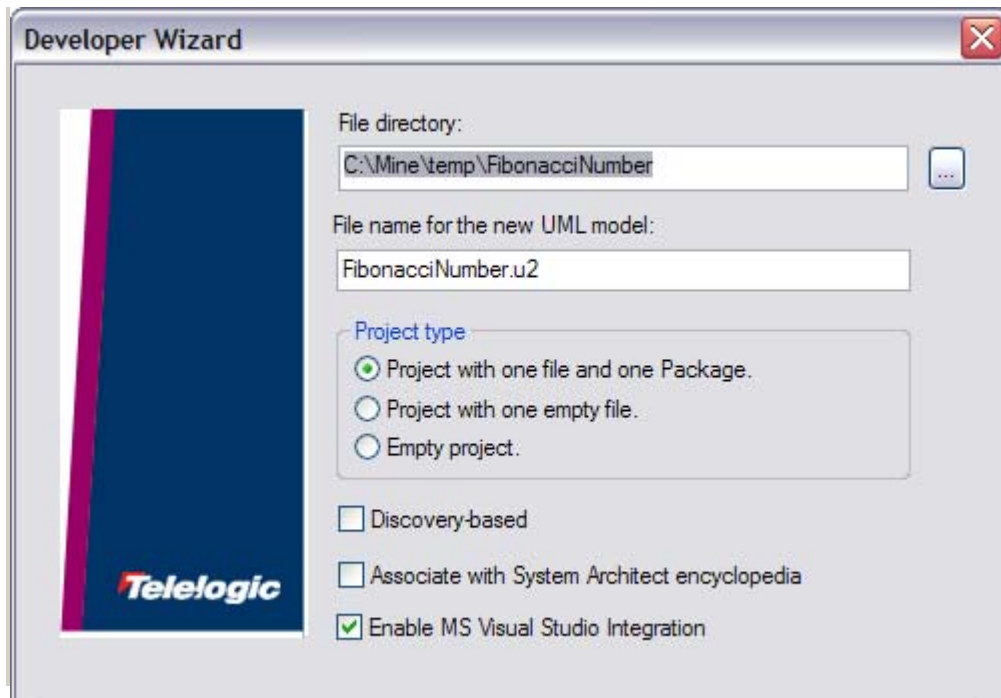
In this exercise, you create a new C# project and workspace in Tau. You also create a package structure that you reference in completing the exercises in this tutorial.

### Task 1a: Creating a New Project and Workspace

To create a C# project, follow these steps:

1. Click **Start > Programs > Telelogic > Telelogic Tau *version number***.
2. Do one of the following:
  - ◆ Press **CTRL+N**, or
  - ◆ On the main toolbar, select **File > New**, or
  - ◆ On the Telelogic Tau Welcome page, in the **New Project** panel, click **Proceed**.
3. In the **New** dialog box, in the **Project** tab, choose **UML for C# Code Generation**.
4. In the **Project name** box, type `FibonacciNumber`.
5. In the **Location** box, enter a new directory name or browse to an existing directory.

6. Accept the default option **Create new workspace**.
7. Click **OK**.



8. In the Developer Wizard, select **Enable MS Visual Studio Integration**.
9. Accept the remaining Developer Wizard defaults and click **Next**, then click **Finish**. Tau creates a new project and workspace. In the Output window Messages tab, the following message displays:

Add-in module CSharpApplication activated.

Add-in module MSVS8Integration activated.

## Exercise 2: Working with Packages

In this exercise, you create two C# packages in your workspace. The first package is for analysis and requirements diagrams that provide a high level overview of your application. The second package is for a class diagram. In subsequent lessons, you create a use case diagram and an activity diagram in the analysis package, and a class diagram in the class package.

By default, when you create your project, Tau adds a single package in the project directory. The package has the same name as the project. In the first task of this lesson, you rename the package

included with the project. In the second task, you create a second C# package for your class diagram.

## Task 2a: Renaming the Default Package

To rename the C# package, follow these steps.

1. In the Model view, select the **FibonacciNumber** package.
2. Press **F2**.
3. Type `Analysis`.

## Task 2b: Creating a New Package

To create a new C# package, follow these steps:

1. In the Model view, right-click **Model**, then Select **New Model Element > Package**.
2. In the Create Model Root Element dialog box, in the **Element Name** field, type `Implementation`.
3. On the main menu bar, select **File > Save All**. Notice all the red change bars disappear.

## Summary

In this lesson, you created a project that will serve as the basis and structure for storing the models in the rest of the tutorial. You are ready to proceed to the next lesson, in which you begin your project by creating a use case diagram.



# Lesson 2: Creating a Use Case Diagram

*Use case diagrams* show the behavior and capabilities of a system as it interacts with an external user or actor. A use case diagram also shows what a system will do and who will use it.

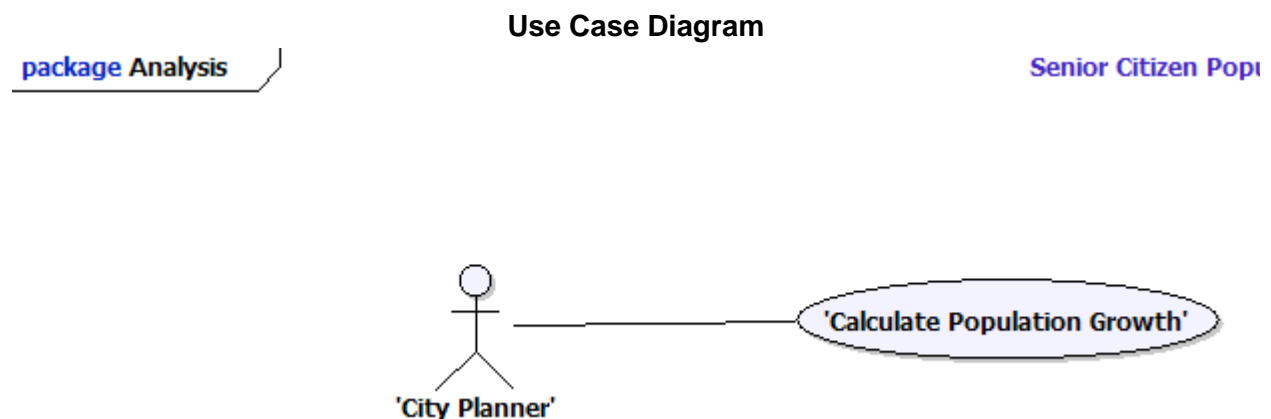
## Goals for this Lesson

In this lesson, you create a simple use case diagram for the application you are modeling. As you work through this lesson, you will learn about the following elements in use case diagrams, and how to draw them:

- ◆ Actors
- ◆ Use Cases
- ◆ Associations

## Exercise 1: Creating a Use Case Diagram

The following figure shows the use case diagram that you create in this exercise.




## Task 1a: Adding a New Use Case Diagram

To create a use case diagram, follow these steps.

1. Right-click **Analysis**.
2. Select **New Diagram > Use Case diagram**. Tau creates the use case diagram in your workspace and opens it in the drawing area.

## Task 1b: Renaming a Use Case Diagram

In this task, you use the properties editor to rename the use case diagram. Follow these steps.



1. Right-click **Use case diagram1** and select **Properties**.
2. In the Edit Properties dialog box, in the **Name** field, replace the default name with `Senior Citizen Population`.
3. Click the **Close** button  to exit the Properties dialog box. Optionally you can leave this open and it will update to reflect the properties of the selected element .

## Task 1c: Adding an Actor and a Use Case

In this task, you add an actor and a use case to the diagram. An *actor* is an external element outside of the system that interacts with the system. A *use case* illustrates the capabilities of a system and shows why a user interacts with the system.

The actor in your diagram is a city planner who is using a system to gather data on the expected population growth of senior citizens in the city. The use case shows the system using a Fibonacci algorithm to compute the growth projections the planner needs to help determine the appropriate size of the new senior center.

To add an actor and a use case to your diagram, follow these steps.

1. Click the actor symbol  on the **Drawing** toolbar, then click in the drawing area. Tau adds an actor element in the drawing area.
2. Replace the default name with `CityPlanner`.
3. Click the use case symbol  on the **Drawing** toolbar, then click in the drawing area. Tau adds a use case element in the drawing area.
4. Replace the default name with `CalculatePopulationGrowth`.

## Task 1d: Adding an Association

An *association line* shows a relationship between two elements in a use case diagram. In this task, you draw an association line that shows the interacting relationship between the city planner and

the application use case. You can add an association line using the association symbol on the Drawing Tool menu, or by selecting the association “handle” on the **CityPlanner** element.


**To draw an association line using the handle:**

1. In the drawing area, select **CityPlanner**.
2. Click on the Association “handle” at the bottom of the **CityPlanner** element as shown in the following figure.



3. Click anywhere inside of the **CalculatePopulationGrowth** use case element. Tau adds an association line that connects the two elements.

**To draw an association line using the symbol:**

1. Click the **Association** symbol  on the **Drawing** Toolbar.
2. Click the right edge of **CityPlanner** and the left edge of **ComputePopulationGrowth**. Tau adds an association line that connects the two elements.
3. On the main menu bar, select **File > Save All**.

Your drawing should resemble the [Use Case Diagram](#) figure.

## Summary

In this lesson, you created a use case diagram. You became familiar with the following elements of use case diagrams:

- ◆ Actors
- ◆ Use cases
- ◆ Associations

You are now ready to proceed to the next lesson, in which you create an activity diagram.



# Lesson 3: Creating an Activity Diagram

An activity diagram shows behavior based on sequences of activities. The activity diagram consists of various activities, data, and messages connected to each other using arrows. The arrows are used to show the direction of activity flow in the diagram.

## Goals for this Lesson

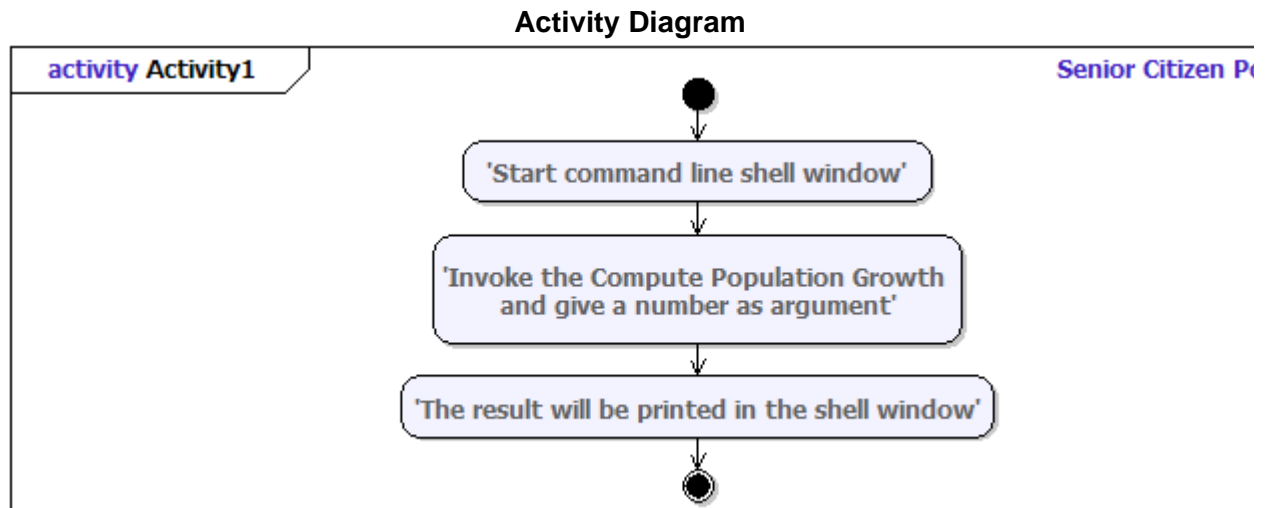
In this lesson, you create an activity diagram. Your activity diagram will show the sequence of activities that occur when the city planner uses the application to estimate the future population growth of senior citizens. The city planner starts the application, then enters a value that the application uses to calculate the population growth. In your activity diagram, the value supplied by the planner represents a number of years in the future. The response from the application is a population growth projection based on the number of years.

When you create this activity diagram using Tau, you will learn how to draw

- ◆ An initial node
- ◆ An action node
- ◆ An activity line
- ◆ A final node

## Exercise 1: Creating an Activity Diagram

The following figure shows the Activity Diagram that you create in this exercise.



### Task 1a: Configuring UML Settings

In this task, you configure UML editing settings so you can draw the elements of your activity diagram as shown in the [Activity Diagram](#) figure. By default, Tau is configured to draw the elements of an activity diagram horizontally. Follow these steps to change the setting to vertical so you can draw the elements as shown above.

1. On the menu bar, select **Tools > Options**, then click the **UML Advanced Editing** tab.
2. On the **UML Advanced Editing** tab, in the Activity diagrams panel, select **Vertical** from the **Autocreate orientation** drop down list box.
3. Click **OK**.

### Task 1b: Adding a New Activity Diagram

To create an Activity diagram, follow these steps.




1. In the Tau browser, expand **Model**, then right on the **Analysis** package.
2. Select **New Diagram > Activity Diagram**. Tau creates the activity diagram in your workspace and opens it in the drawing area.

3. In the Edit Properties dialog box, in the **Name** field, replace the default name with Senior Citizen Population.
4. Click outside of the **Name** field to commit your name change.

## Task 1c: Drawing Activity Nodes

Nodes show a specific unit of behavior within an activity flow. In this task, you draw an initial node, three action nodes and a final activity node in your diagram. The nodes show the units of behavior that occur when the user starts the application and enters a value that the application uses to produce a population projection figure.

To draw activity nodes, follow these steps:

1. Click the **initial node**  symbol on the **Drawing** toolbar, then click in the drawing area. Tau adds an initial node element in the drawing area.
2. In the drawing area, select the **initial node**.
3. Press **Shift-Spacebar** and on the pop-up menu, click the **Activity/action**  symbol. repeat twice to result in three action nodes. Notice that each time you add an action node, Tau automatically includes an *activity flow arrow* between each node. The arrows show the direction of flow in the diagram.
4. Continue to press **Shift-Spacebar** and click the **Activity final**  symbol on the **Drawing** toolbar.
5. Click each activity node and type the names as shown in the [Activity Diagram](#) figure.
6. On the main menu bar, select **File > Save All**.

You have finished drawing the activity diagram. Your diagram should resemble the [Activity Diagram](#) figure.

## Summary

In this lesson, you created an activity diagram. You learned about the following elements in an activity diagram:

- ◆ Initial Nodes
- ◆ Action Nodes
- ◆ Final Nodes
- ◆ Associations

You are now ready to proceed to the next lesson, in which you create a class diagram.

# Lesson 4: Creating a Class Diagram

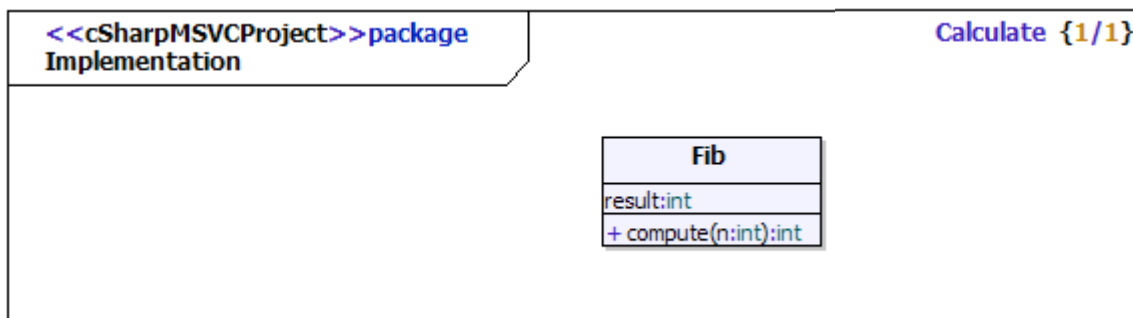
A class diagram shows the types of elements in a system and how they interact and relate to each other. Class relationships are typically shown with dependency, generalization and association lines.

## Exercise 1: Creating a Class Diagram

In this exercise, you create a class diagram and draw a class in the diagram. The class contains an operation and an attribute for the application you are modeling.

### Task 1a: Adding a New Class Diagram

The following figure shows the Class Diagram that you create in this exercise.




To create a class diagram, follow these steps:

1. In the browser, expand **Model**.
2. Right-click the **Implementation** package and select **New Diagram > Class Diagram**. Tau creates the class diagram in the scope of the **Implementation** package.
3. Change the name of the **Class Diagram 1** by using the Edit Properties dialog box, or by selecting the diagram in the model view and pressing **F2**. The new name should be **Calculate**.

## Task 1b: Drawing a Class

To draw a class, follow these steps:

1. Click the Class button  on the **Drawing** toolbar, then click anywhere in the drawing area to add the class to the diagram.
2. Name the class `Fib` by selecting in the class on the diagram and typing the text.

## Task 1c: Adding Attributes and Operations

In this task, you add an attribute and an operation to the class you created in the previous task. The *attribute* you add will be the resulting number that is generated when the application performs a computation. The *operation* is the act of computing the number.

To add attributes to the **Fib** class, follow these steps:

1. Select the **Fib** class.
2. Place the cursor in the middle compartment of the class box and type `result:int` in the attribute text box.
3. Place the cursor in the bottom compartment of the class box and type `compute(n:int):int` in the operation text box. You can try using name completion here by using **CTRL + Space bar** after you started typing `int` to see a list of possible candidates.
4. If your types have a red underline perform a **Check All**
5. On the main menu bar, select **File > Save All**.

## Summary



In this lesson, you created a class diagram. You learned about the following elements in a class diagram:

- ◆ Attributes
- ◆ Operations

You are now ready to proceed to the next lesson, in which you will generate code from the **Fib** class.

# Lesson 5: Generating C# Code and More

## Goals for this Lesson

In this lesson, you

- ◆ Generate source code from your model.
- ◆ Add external code, and view updates to your model.
- ◆ Build and run your application.

## Exercise 1: Generating and Editing C# Code

In this exercise, you generate C# code from the **Fib** element you created in the previous lesson. After you generate the code, you use Microsoft Visual Studio to add external code to the `compute` operation contained in that element.

### Task 1a: Generating and Viewing C# Code

To generate the C# code from your source files, perform the following steps.

1. Right-click on **Implementation** package and select **Update C# source code**. Tau generates C# source code files for the classes and interfaces contained in your.

**Note:** Default file mapping is used to generate the source code, more on configuring this can be found in the online help.

2. To examine the generated code, right-click the **Fib** class and select **Goto source**.
3. In the **Visual Studio Selection** dialog box, select **New Instance of Visual Studio**, then click **OK**.

Microsoft Visual Studio displays the source code in a file named **Implementation.Fib.cs**.

4. Close Microsoft Visual Studio.

## Task 1b: Creating a Project in Microsoft Visual Studio

In this task, you create a project in Microsoft Visual Studio. For the remainder of this tutorial, you work with your model in both Visual Studio and in Tau. The Visual Studio project you create in this task allows you to edit the source code that you generated in [Task 1a: Generating and Viewing C# Code](#).

Before you create the project, verify that the MSVS8 add-in is enabled in Tau. To do this, perform the following steps:

1. From the main menu, select **Tools > Customize**.
2. Click the Add-ins tab.
3. In the list of Customization modules, verify that **MSVS8Integration** is checked.
4. Click **Close**.

Tau is also dependent on Visual Studio saving the projects when they are created. If your generated files not not show up in your project this may be due to your Visual Studio settings. To correct this set the following option:

1. From the main menu, select **Tools > Options**.
2. In **Project Solutions** there is an option **Save new projects when created**. Make sure the check box is selected.
3. Close Visual Studioe a repeat the following steps.

To create a Visual Studio project, follow these steps:

1. In the Model view select the **Implementation** package.
2. From the Visual Studio .NET menu, select **Create/Update Visual Studio .NET C# Project**.
3. Select **New instance of Visual Studio**.
4. Click **OK**.
5. In the New Project dialog box, select the **Console Application** template and give your project a name.
6. Click **OK**.

The Console Application main menu is displayed. The Solutions Explorer panel contains several files used by MSVS. In the next task, you edit the **Implementation.Fib.cs** and **Program.cs** source files. **Implementation.Fib.cs** contains the code created from your UML model in Tau.



**Program.cs** is a file generated by Visual Studio containing a class program that serves as the main function in your application. This class has automatically been added to your model view in Tau.

## Task 1c: Adding Code for the Compute Operation

In this task you add code that is the body of the `compute` operation that you created in [Lesson 4: Creating a Class Diagram](#). To add code manually for the `compute` operation, follow these steps:

1. In the Model browser, open the **Calculate** diagram.
2. In the drawing area, right-click the **Fib** class and select **Goto source**. Select the submenu choice corresponding to the generated class.
3. Delete the default entry `class Fib` and replace it with the following text:

```
class Fib
{
    int result;
    public int compute(int n)
    {
        if (n == 0)
            result = 1;
        else if (n == 1)
            result = 2;
        else
            result = n + compute(n - 1);
        return result;
    }
}
```

4. Open the **Program.cs** file and change the Program class to

```
class Program
{
    static Implementation.Fib myFib;
    static void Main(string[] args)
    {
        myFib = new Implementation.Fib();
        int res = myFib.compute(Convert.ToInt32(args[0]));
        Console.WriteLine("The result was " +
            res.ToString());
    }
}
```

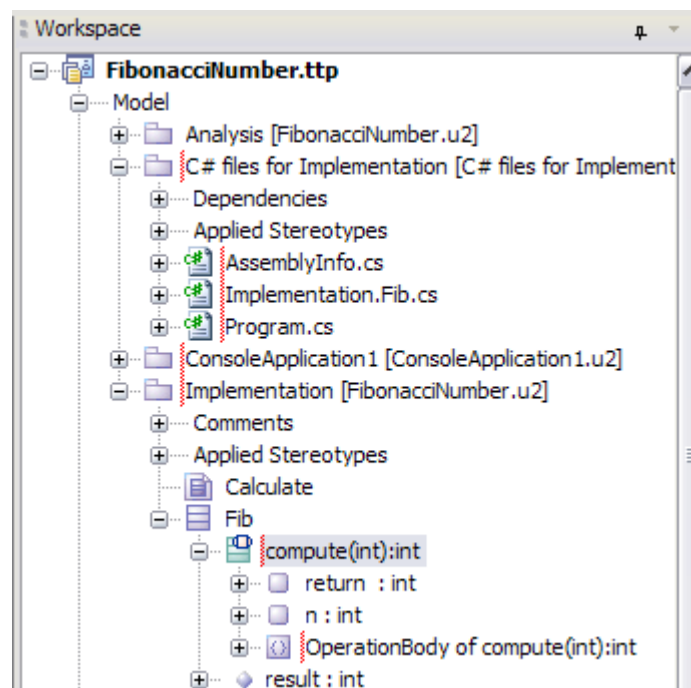
5. Select **File > Save All**.

## Task 1d: Viewing Model Updates

In this task, you view the updates that were made to the model when you added the code in the previous task. To view the updates in Tau that you made in the previous task, follow these steps.

1. On the Tau main menu, select **C# > Update Model**. Optionally you can perform this from Visual Studio using **Tau > Create/Update Project**.
2. In the Tau Model view, expand the **Implementation** package, then expand the **Fib** class.

Notice that Tau has added the `operationBody` element to the `compute(int):int` operation, as shown in the following figure. This is a container in the model that stores the code implementing the behavior of `compute`.



## Task 1e: Compiling C# Code

In this task, you compile the source code in Visual Studio. To compile your code, follow these steps.

1. On the Console Application main menu, select **Build > Build Console Application1**.

2. Verify that a “Build succeeded” message is in the status bar in Visual Studio.

## Task 1f: Running your Application

The application you have modeled in this tutorial runs an algorithm that computes Fibonacci numbers. To run the application, follow these steps.

1. In a command window, navigate to your Visual Studio solutions directory.
2. In the Release directory, enter the following command:

```
ConsoleApplication1 "n"
```

Where `ConsoleApplication1` is the name of the executable and “n” is an integer representing a number of years in the future.

3. Press **Enter**. The result is total number of seniors expected to be residing in the city in the number of years you specify. For example, if you entered 22, the total number of senior expected to be residing in the city in 22 years would be approximately 256.

## Optional

Depending on whether you prefer to work primarily in the model or the code Tau has some options to help automate the updated. Tau enables you automatically update your model, or your code based on settings specified after applying the **TTDCSharp::CSharpSettings** stereotype. This stereotype enables the following functionality:

- ◆ Automatic model Update - This will update the model whenever the source code is saved.
- ◆ Automatic source generation - Whenever something is changed in the model the code will be updated.
- ◆ Support for roundtripping - Enables one to add new code and keep the model current.

To apply the C# stereotype, follow these steps.

1. In the Model view, right-click **Model**, then select **Properties**.
2. Click **Stereotypes**.

In the Stereotypes dialog box, select the **TTDCSharp::CSharpSettings** checkbox.

## Summary

In this lesson, you generated C# code, manually added code, and ran your application. You learned how to

- ◆ Export a C# package
- ◆ Edit source code
- ◆ Run your generated application

---

# Conclusion

This tutorial has introduced you to UML modeling with Telelogic Tau in a C# environment. By completing the exercises in this tutorial, you have become familiar with the Tau product. You have learned how to:

- ◆ Create a project
- ◆ Use drawing tools and shortcut keys
- ◆ Draw diagrams
- ◆ Compile code
- ◆ Add external code to a model

Your knowledge of how to perform these tasks gives you a basic understanding of Telelogic Tau. You will enhance your skills and product knowledge as you continue to work on UML modeling projects using this product.



# Technical Support and Documentation

## Contacting Technical Support

The Technical Support staff answers questions about installation issues, application issues, product defect reporting, and documentation. Technical support engineers, in conjunction with sales application engineers, assist prospective customers with product evaluations and provide timely responses to user issues to ensure maximum productivity.

## Regional Contact Information

**The Americas:**

[tausupport.us@telelogic.com](mailto:tausupport.us@telelogic.com)

**Middle East/Africa:**

[tausupport.eu@telelogic.com](mailto:tausupport.eu@telelogic.com)

**Asia Pacific:**

[support.apac@telelogic.com](mailto:support.apac@telelogic.com)

## Tau Documentation

The Telelogic Tau documentation provides information on most of the topics covered in this tutorial. documentation is available from the following locations:

- ◆ From the Start menu, click **Programs > Telelogic > Telelogic Lifecycle Solutions Documentation > Telelogic Tau *version number* > Tau Help**.
- ◆ From the Help menu in the Tau interface.

The following table lists the Help topics that provide additional information on key concepts covered in this tutorial.

<b>Help Topic</b>	<b>Reference Information</b>
“UML and C#”	General information on UML modeling using Tau in a C# environment.
“Working with Diagrams”	Provides information on creating, saving, and printing diagrams, as well as other common diagram operations.
“UML Language Guide”	Provides a complete list of UML language constructs and model elements



---

# Index

## A

- Activity diagrams
  - creating 16
- Adding
  - actors 12
  - association lines 12
  - attributes 20
  - operations 20

## C

- C# language
  - editing code 21
  - generating code 21
  - packages 8
- Class diagrams
  - creating 19
- Closing
  - Tau 4
- Customer support 29

## D

- Diagrams
  - activity 16
  - class 19
  - usecase 11
- Directories 4
- Documentation 29
  - conventions 2

## E

- Elements
  - names 4
- Exiting
  - Tau 4

## F

- Files 4

## L

- Launching
  - Linux Tau 3
  - Tau 3
  - Windows Tau 3
- Linux 3

## M

- Model
  - views 5
- Models
  - element names 4

## N

- Names
  - for elements 4

## O

- Opening
  - Tau 3
- Operations 20

## P

- Packages
  - creating 8
  - renaming 9
- Projects 4
  - creating 7
  - directories 7
  - files 7
  - saving 3

## R

- Renaming 9
- renaming 11

### S

Solaris  
starting Tau on 3  
Stereotypes  
applying 25

### T

Tau 2  
closing 4  
documentation 29  
exiting 4  
projects 4  
starting 3

technical support 29  
Technical support 29  
Tool Tips 3

### U

Use case diagrams 11  
creating 11

### W

Windows  
starting Tau 3