



IBM ILOG Gantt for .NET V4.0

Programming with

IBM ILOG Gantt for .NET Windows

Forms and Web Forms Controls

June 2009

© Copyright International Business Machines Corporation 1987, 2009.

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

Contents

Preface	Programming with IBM ILOG Gantt for .NET Windows Forms and Web Forms	
Controls	7	
Creating and Using Gantt Data Models		9
Introducing Gantt Data Model Interfaces		10
Introducing the Gantt Data Model In-Memory Implementation		11
Populating Gantt Data Models		12
Managing Activities		13
Managing Resources		15
Managing Constraints		17
Managing Reservations		19
Optimizing Gantt Data Model Modifications		20
Displaying Scheduling Data		21
Creating Custom Gantt Data Models		21
Extending the In-Memory Implementation		22
Extending the Model without Coding		22
Creating a Subclass of SimpleGanttModel		24
Creating a Subclass of SimpleActivity		24
Implementing the Gantt Data Model Interfaces		27
Implementing the IGanttModel Interface		27

Implementing the Collection Interfaces of the Gantt Data Model	30
Implementing the Scheduling Entities	32
Listening to Gantt Data Model Events	33
Catching Activity Events	34
Catching Resource Events	36
Catching Constraint Events	37
Catching Reservation Events	39
Displaying Scheduling Data Using Gantt Charts	41
Introducing the Gantt Chart, Schedule Chart, and Reservation Chart Controls	42
Displaying Scheduling Data	45
Accessing the Lower-Level Controls	46
Modifying the Appearance of Gantt Chart Controls	47
Using the Predefined Behavior of the Gantt Chart Controls	50
Expanding or Collapsing Rows	51
Grouping, Filtering and Sorting Rows	54
Controlling the Displayed Time Interval	55
Displaying Scheduling Data Using Tables	57
Introducing the Activity, Resource, and Reservation Tables	59
Connecting a Gantt Table to a Gantt Data Model	61
Modifying the Appearance of a Gantt Table	61
Managing Columns of a Gantt Table	64
Modifying the Appearance of Table Columns	65
Editing Values in the Table	66
Default Columns for Activity, Resource, and Reservation Tables	69
Dialog Box for Editing the Columns of a Gantt Table	70
Using Predefined Behavior to Manipulate Rows and Columns	71
Expanding or Collapsing Rows	73
Scrolling the Gantt Table	74
Getting and Setting the Current Cell	75
Controlling Selection in the Table	75

Hit Testing in the Gantt Table	76
Grouping, Filtering and Sorting Rows	77
Displaying the Load of a Resource	81
Introducing the LoadChart Class	82
Connecting the Load Chart to a Resource	82
Modifying the Appearance of a Load Chart	82
Controlling the Displayed Time Interval	84
Displaying Activities Using a Calendar View	85
Introducing the CalendarView class	86
Displaying Activities in the Calendar View	87
Modifying the Appearance of the Calendar View	87
Representing Activity bars in the Calendar View	89
Controlling the Layout of Activities in the Calendar View	90
Using the Predefined Behavior of the Calendar View	91
Hit Testing in the Calendar View	93
Customizing the Drawing of Gantt Components	95
Using Time Grids and Date Indicators	95
Representing Activity Bars in Gantt Sheets	98
Styling Activity Bars in Gantt Sheets	100
Modifying the Appearance of Activity Bars	101
Defining When a Bar Style Applies	106
Interacting and Styling	107
Example of Styling	108
Dialog Box Control for Styling Activity Bars	109
Modifying Styles	110
Programming the Activity Bar Style Dialog Box	112
Creating Owner-Drawn Gantt Components	113
Providing User Code to Draw Gantt Table Cells	113
Providing User Code to Draw Gantt Sheet Rows	114
Providing User Code to Draw Activity Bar Styles	115

Providing User Code to Draw Time Scale Rows	116
Providing User Code to Draw Constraint Links	117
Creating Custom Gantt Representations.	119
Displaying Scheduling Data using Gantt Sheets	120
Introducing the Activity Sheet, Schedule Sheet, and ReservationSheet	120
Displaying Scheduling Data in Gantt Sheet Controls	123
Modifying the Appearance of Gantt Sheet Controls	124
Using the Predefined Behavior of Gantt Sheet Controls	127
Controlling the Displayed Time Interval	132
Hit Testing in the Gantt Sheet Controls	133
Using Time Scales	134
Introducing the Time Scale Class	134
Modifying the Appearance of the Time Scale	135
Using the Predefined Behavior of Time Scales	136
Controlling the Displayed Time Interval	137
Customizing Time Scale Rows	137
Synchronizing a Time Scale and a Time Grid	138
Displaying Time-based Information	138
Synchronizing the Time of Several Controls.	140
Using Time Lines	141
Using Time Scrollbars	143
Reading and Writing Scheduling Data Using XML	145
Overview of the SDXL Language	146
Serializing Scheduling Data to SDXL	147
Deserializing Scheduling Data from SDXL	149
Customizing XML Serialization or Deserialization	151
Working with ADO.NET	155
Overview of the Architecture	156
Using Gantt Model Adapters	157
Updating a Gantt Data Model from a DataSet	157

Filling a DataSet from a Gantt Data Model	158
Using the Generic Gantt Model Adapter	160
Developing a Custom Gantt Model Adapter	162
Using the Clipboard to Store Scheduling Data	167
Storing Scheduling Data in the Clipboard.....	168
Retrieving Scheduling Data from the Clipboard.....	169
Managing Undo/Redo in a Gantt Data Model	171
Enabling Undo/Redo	172
Disabling Undo/Redo.....	172
Undoing Modifications	172
Redoing Modifications.....	172
Grouping Modifications.....	173
Using Predefined Dialog Boxes for Editing Scheduling Data	175
Editing an Activity Using the Predefined Dialog Box.....	176
Editing a Resource Using the Predefined Dialog Box	177
Editing a Constraint Using the Dialog Box	178
Printing Gantt Charts.....	181
Introducing the GanttPrintDocument Class	181
Customizing Printing.....	185
Storing and Displaying Working and Nonworking Times	187
Using a WorkCalendar to Store Working and Nonworking Periods	188
Navigating in a WorkCalendar	189
Editing the Content of a WorkCalendar.....	190
Displaying Working and Nonworking Times in Gantt Controls.....	191
Creating Project Scheduling Applications with IBM ILOG Gantt for .NET	193
Programming with the ProjectSchedulingModel	194
The ProjectSchedulingModel Class.....	195
Activities in the ProjectSchedulingModel	196
Controlling When the Project Schedule is Recomputed	198

Displaying and Editing Content of a ProjectSchedulingModel in a Gantt Control	199
How the Project Scheduling Model Computes the Schedule of a Project	200
How Project Start Date Affects the Schedule.	201
How Constraint Links Affect the Schedule	201
How Constraints on Activities Affect the Schedule	203
Calendars in the Project Scheduling Model	205
Resource Leveling in ProjectSchedulingModel	206
Displaying the Critical Path of a Project Scheduling Model	207
Saving and Reading a ProjectSchedulingModel to an XML File	208
Localizing a Gantt Application	211
Creating a Localization Project.	212
Translating the Resource Files.	212
Creating the Satellite Assemblies	213
Displaying a GanttChart in an AJAX Web Application	215
Overview of the IBM ILOG Gantt for .NET AJAX Framework.	216
Adding AJAX Capabilities to IBM ILOG Gantt for .NET Web Controls	216
Customizing the AJAX Extenders	221
Interacting with the Client Control	222
Providing Contextual Information to the Client Control.	223

Programming with IBM ILOG Gantt for .NET Windows Forms and Web Forms Controls

This section provides the essential programming information you need to build applications with IBM® ILOG® Gantt for .NET. This section provides information about key programming concepts, as well as code samples and detailed explanations.

In This Section

Creating and Using Gantt Data Models

Introduces the Gantt data model, that is, the classes that contain the scheduling data you want to display.

Listening to Gantt Data Model Events

Explains how to listen to Gantt data model events.

Displaying Scheduling Data Using Gantt Charts

Describes how to display scheduling data using the main controls of IBM ILOG Gantt for .NET.

Displaying Scheduling Data Using Tables

Describes how to display scheduling data in tables using the IBM ILOG Gantt for .NET controls.

Displaying the Load of a Resource

Describes how to use the Load Chart control for displaying the load of a resource.

Displaying Activities Using a Calendar View

Describes how to use the Calendar View control for displaying activities in a calendar.

Customizing the Drawing of Gantt Components

Describes how to customize the drawing of the main IBM ILOG Gantt for .NET controls.

Creating Custom Gantt Representations

Describes how to develop custom components by assembling and connecting Gantt controls together.

Reading and Writing Scheduling Data Using XML

Describes how to read and write scheduling data using XML.

Working with ADO.NET

Describes how to use Gantt data models with ADO.NET.

Using the Clipboard to Store Scheduling Data

Explains how to store scheduling data into the clipboard.

Managing Undo/Redo in a Gantt Data Model

Describes the class responsible for managing undo/redo in a Gantt data model.

Using Predefined Dialog Boxes for Editing Scheduling Data

Describes how to use predefined dialog boxes for editing scheduling information.

Printing Gantt Charts

Describes how to add printing capabilities to your Gantt application.

Storing and Displaying Working and Nonworking Times

Describes how to use the classes that allow you to store, edit, and display the working and nonworking times.

Creating Project Scheduling Applications with IBM ILOG Gantt for .NET

Explains how to create applications that require project scheduling capabilities through a specific Gantt Data Model class: the ProjectSchedulingModel.

Localizing a Gantt Application

Describes how to create a localized version of IBM ILOG Gantt for .NET.

Displaying a GanttChart in an AJAX Web Application

Describes how to build AJAX-enabled Web applications with IBM ILOG Gantt for .NET.

Creating and Using Gantt Data Models

IBM® ILOG® Gantt for .NET provides an abstract data model that allows you to specify the complete mapping of user scheduling data to scheduling data understandable by the Gantt library.

An in-memory implementation of the Gantt data model is also provided. This ready-to-use implementation can be easily extended.

In This Section

Introducing Gantt Data Model Interfaces

Describes the data model interfaces of IBM ILOG Gantt for .NET.

Introducing the Gantt Data Model In-Memory Implementation

Describes the concrete data model in-memory implementation classes.

Populating Gantt Data Models

Explains how to populate a Gantt data model using the Gantt Data Model API.

Displaying Scheduling Data

Describes how to connect a Gantt data model to a Gantt control.

Creating Custom Gantt Data Models

Explains how to create a customized Gantt data model.

Related Sections

The Data Model

Presents the scheduling information model whose content is displayed as Gantt charts.

Working with ADO.NET

Describes how to use Gantt data models with ADO.NET.

Reading and Writing Scheduling Data Using XML

Describes how to read and write scheduling data using XML.

Creating Project Scheduling Applications with IBM ILOG Gantt for .NET

Explains how to create applications that require project scheduling capabilities.

Introducing Gantt Data Model Interfaces

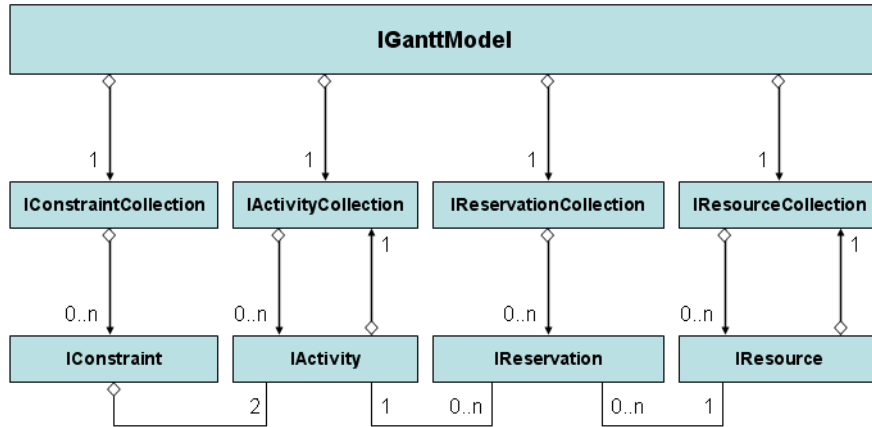
The data model is completely abstract and is defined by the following interfaces:

Interface	Description
IGanttModel	The main interface of the model. It acts as an intelligent container for the other abstract interfaces that represent the scheduling data. The interface contains factory methods to create the scheduling entities: activities, resources, constraints, and reservations. It is responsible for triggering events when parts of the model are modified. Finally, it maintains the unique identifiers of activities and resources.
IActivity	Represents an activity or task that must be completed in the schedule. Activities are hierarchical and therefore an activity can contain child activities. An activity with at least one child is called a <i>parent activity</i> . Conversely, an activity with no child activities is called a <i>leaf activity</i> . Each activity has an identifier that is unique in the model.
IActivityCollection	An indexed collection of activities.
IResource	Represents a resource that can be allocated to an activity to make its completion possible. Resources are hierarchical and therefore a resource can contain child resources. A resource with at least one child is called a <i>parent resource</i> . Conversely, a resource with no child resources is called a <i>leaf resource</i> . Each resource has an identifier that is unique in the model.
IResourceCollection	An indexed collection of resources.

Interface	Description
IConstraint	Represents an activity-to-activity scheduling constraint.
IConstraintCollection	A set of constraints.
IReservation	Represents the allocation of a resource to an activity.
IReservationCollection	A set of reservations.

All these classes are located in the ILOG.Views.Gantt.Data namespace.

The following illustration shows the relationships between the interfaces that compose the Gantt data model:



See Also

Populating Gantt Data Models | Working with ADO.NET | Reading and Writing Scheduling Data Using XML | Introducing the Gantt Data Model In-Memory Implementation

Introducing the Gantt Data Model In-Memory Implementation

IBM® ILOG® Gantt for .NET provides a ready-to-use implementation of the Gantt data model. The classes listed here are basic implementations of the Gantt data model interfaces. They can be used to manipulate standard scheduling data directly or they can be extended to meet specific requirements.

The in-memory data model implementation is defined by the following classes:

Interface	Description
SimpleGanttModel	A basic implementation of the IGanttModel interface. It uses the classes listed below.
SimpleActivity	A basic implementation of the IActivity interface.
SimpleResource	A basic implementation of the IResource interface.
SimpleConstraint	A basic implementation of the IConstraint interface.
SimpleReservation	A basic implementation of the IReservation interface.

All these classes are located in the ILOG.Views.Gantt.Data namespace.

See Also

Populating Gantt Data Models | Working with ADO.NET | Reading and Writing Scheduling Data Using XML | Creating Project Scheduling Applications with IBM ILOG Gantt for .NET

Populating Gantt Data Models

The API of the Gantt data model allows to modify the scheduling data represented by the model.

The Gantt data model can be populated before or after a chart has been bound to it. Initial data is immediately displayed by the chart when it binds to the data model. Data populated after the chart has been bound causes the chart to be updated dynamically to reflect the new data in the data model.

Although a Gantt data model can be populated through the abstract API, the SimpleGanttModel class is used for convenience to demonstrate code fragments in the following sections.

In This Section

Managing Activities

Describes how to manage activities inside the Gantt data model.

Managing Resources

Describes how to manage resources inside the Gantt data model.

Managing Constraints

Describes how to manage constraints inside the Gantt data model.

Managing Reservations

Describes how to manage reservations inside the Gantt data model.

Optimizing Gantt Data Model Modifications

Describes how to optimize Gantt data model modifications.

Related Sections

The Data Model

Describes the scheduling data contained in the data model.

Working with ADO.NET

Describes how to use Gantt data models with ADO.NET.

Reading and Writing Scheduling Data Using XML

Explains how to serialize and deserialize scheduling data to and from an SDXL stream.

Managing Activities

The Gantt data model manages root activities in a collection. As activities are hierarchical in nature, each activity also manages a collection of its subactivities. The `IActivityCollection` interface contains methods for accessing, adding, removing, or moving activities.

Accessing Activities

The `IGanttModel.Activities` property returns the collection of the root activities of the model.

To get a collection containing the subactivities of a specified activity, use the `IActivity.ChildActivities` property. If the activity is a leaf activity, then the returned collection should be empty.

The `IGanttModel.FindActivity` method can be used to find an activity using its unique identifier.

Adding Activities

A collection of activities can be obtained from the Gantt data model with the `IGanttModel.Activities` property of the model. The collection that is returned allows you to modify the root activities of the model:

```
IGanttModel model = new SimpleGanttModel();  
IActivity activity = model.NewActivity();  
model.Activities.Add(activity);
```

To add child activities to an existing activity, retrieve the collection of activities of the parent activity with the `IActivity.ChildActivities` property of the activity. The collection that is returned allows you to modify the child activities of the parent activity:

```
IGanttModel model = new simpleGanttModel();
IActivity activity = model.NewActivity();
IActivity childActivity = model.NewActivity();
activity.ChildActivities.Add(childActivity);
model.Activities.Add(activity);
```

Note: *The child activity is added to its parent before the parent is added to the model. This gives better performance than if the parent activity is added to the model first. As a general rule, if you need to add a complete hierarchy of activities, you should connect the root of the new activities after setting up the whole hierarchy of activities.*

The `AddRange` method is for adding several activities at a time. This is a more efficient way of adding several activities than calling **Add** several times.

Removing Activities

Removing an activity from the model means removing this activity from its collection. This can be done by using the `IActivityCollection.Remove` method. The following C# code sample shows how to remove an activity whose unique identifier is "A1":

```
IGanttModel model = ...;
IActivity activity = model.FindActivity("A1");
if (activity != null)
{
    IActivityCollection activities = (activity.Parent == null)
        ? model.Activities
        : activity.Parent.ChildActivities;
    activities.Remove(activity);
}
```

Note: *If the removed activity contains child activities, then the child activities are also considered as being removed from the model. When activities are removed from the model, their associated constraints and reservations are automatically removed first.*

The `RemoveRange` method is for removing several activities at a time. This is a more efficient way of removing several activities than calling **Remove** several times.

Ordering Activities

The order of the activities in an **IActivityCollection** may be important: It can be used by controls to display the list of activities, or by the XML serializer to know in which order activities will be serialized.

To move an activity inside its collection, use the `IGanttModel.MoveActivity` method.

The `GanttModelUtil.SortActivities` methods are used for sorting an `IActivityCollection`. Although you can provide your own comparison mechanism to compare activities, it is easier to use a method that uses a property name to do the sort. For example, if you want to sort all the activities of a Gantt data model based on the `StartTime` property of the activities, use the following C# code:

```
IGanttModel model = new SimpleGanttModel();  
// Populate the model here ...  
GanttModelUtil.SortActivities(model.Activities, "StartTime", true, -1);
```

The parameter **true** indicates that the sort order is ascending. The parameter **-1** specifies that all the activities of the model, including child activities, are to be sorted.

***Note:** Sorting can also be done at the view level, leaving the model unchanged. See [Sorting Rows](#) for details.*

Moving Activities

Moving an activity implies changing its parent activity, its index in its parent-child collection, or both. To do this, use the `IGanttModel.MoveActivity` method.

See Also

[Managing Resources](#) / [Managing Constraints](#) / [Managing Reservations](#)

Managing Resources

The Gantt data model manages root resources in a collection. As resources are hierarchical in nature, each resource also manages a collection of its subresources. The `IResourceCollection` interface contains methods for accessing, adding, removing, or moving resources.

Accessing Resources

The `IGanttModel.Resources` property returns the collection of the root resources of the model.

To get a collection containing the subresources of a specified resource, use the `IResource.ChildResources` property. If the resource is a leaf resource, then the returned collection should be empty.

The `IGanttModel.FindResource` method can be used to find a resource using its unique identifier.

Adding Resources

A collection of resources can be obtained from the Gantt data model with the `IGanttModel.Resources` property of the model. The collection that is returned allows you to modify the root resources of the model:

```
IGanttModel model = new SimpleGanttModel();
IResource resource = model.NewResource();
model.Resources.Add(resource);
```

To add child resources to an existing resource, retrieve the collection of resources of the parent resource with the `IResource.ChildResources` property of the resource. The collection that is returned allows you to modify the child resources of the parent resource:

```
IGanttModel model = new SimpleGanttModel();
IResource resource = model.NewResource();
IResource childResource = model.NewResource();
resource.ChildResources.Add(childResource);
model.Resources.Add(resource);
```

Note: *The child resource is added to its parent before the parent is added to the model. This gives better performance than if the parent resource is added to the model first. As a general rule, if you need to add a complete hierarchy of resources, you should connect the root of the new resources after setting up the whole hierarchy of resources.*

The `AddRange` method is for adding several resources at a time. This is a more efficient way of adding several resources than calling **Add** several times.

Removing Resources

Removing a resource from the model means removing this resource from its collection. This can be done by using the `IResourceCollection.Remove` method. The following C# code sample shows how to remove a resource whose unique identifier is `R1`:

```
IGanttModel model = ...;
IResource resource = model.FindResource("R1");
if (resource != null)
{
    IResourceCollection resources = (resource.Parent == null)
        ? model.Resources
        : resource.Parent.ChildResources;
    resources.Remove(resource);
}
```

Note: *If the removed resource contains child resources, then the child resources are also considered as being removed from the model. When resources are removed from the model, their associated reservations are automatically removed first.*

The `RemoveRange` method is for removing several resources at a time. This is a more efficient way of removing several resources than calling **Remove** several times.

Ordering Resources

The order of the resources in an **IResourceCollection** may be important. It can be used by controls to display the list of resources, or by the XML serializer to know in which order resources will be serialized.

To move a resource inside its collection, use the `IGanttModel.MoveResource` method.

The `GanttModelUtil.SortResources` methods are used for sorting an **IResourceCollection**. Although you can provide your own comparison mechanism to compare resources, it is easier to use a method that uses a property name to do the sort. For example, if you want to sort all the resources of a Gantt data model based on the **MaxUnits** property of the resources, use the following C# code:

```
IGanttModel model = new SimpleGanttModel();
// Populate the model here ...
GanttModelUtil.SortResources(model.Resources, "MaxUnits", true, -1);
```

The parameter **true** indicates that the sort order is ascending. The parameter **-1** specifies that all the activities of the model, including child activities, are to be sorted.

Note: *Sorting can also be done at the view level, leaving the model unchanged. See [Sorting Rows](#) for details.*

Moving Resources

Moving a resource implies changing its parent resource, its index in its parent-child collection, or both. To do this, use `IGanttModel.MoveResource` method.

See Also

[Managing Activities](#) / [Managing Constraints](#) / [Managing Reservations](#)

Managing Constraints

The Gantt data model manages constraints in a collection. The **IConstraintCollection** interface contains methods for adding or removing constraints. The order of the constraints in an **IConstraintCollection** is not important and therefore the API does not allow you to manage the order of constraints in the model.

Accessing Constraints

A set of constraints can be obtained from the Gantt data model with the `IGanttModel.Constraints` property of the model.

To retrieve the constraints associated with a specified activity, use the appropriate property of the **IActivity** interface. The `IActivity.FromConstraints` property returns the collection of the source constraints of the activity. The `IActivity.ToConstraints` property returns the collection of the destination constraints of the activity.

The `IGanttModel.FindConstraint` method can be used to find a constraint between two activities.

Adding Constraints

A set of constraints can be obtained from the Gantt data model with the `IGanttModel.Constraints` property of the model. The set that is returned allows you to add new constraints on the model using the **Add** method:

```
IGanttModel model = new SimpleGanttModel();
IActivity from = model.NewActivity();
IActivity to = model.NewActivity();
model.Activities.AddRange(new IActivity[] { from, to });
IConstraint constraint = model.NewConstraint(from, to,
ConstraintType.EndToStart);
model.Constraints.Add(constraint);
```

Note: The *from* and *to* activities must be added to the model before the constraint.

The `AddRange` method is for adding several constraints at a time. This is a more efficient way of adding several constraints than calling **Add** several times.

Removing Constraints

A set of constraints can be obtained from the Gantt data model with the `IGanttModel.Constraints` property of the model. The set that is returned allows you to remove existing constraints from the model using the **Remove** method. The following C# code sample shows how to remove all the constraints whose source activity unique identifier is A1:

```
IGanttModel model = ...;
IActivity activity = model.FindActivity("A1");
IConstraint[] constraints = new IConstraint[activity.FromConstraints.Count];
activity.FromConstraints.CopyTo(constraints, 0);
model.Constraints.RemoveRange(constraints);
```

The `RemoveRange` methods is for removing several constraints at a time. This is a more efficient way of removing several constraints than calling **Remove** several times.

See Also

Managing Activities / Managing Resources / Managing Reservations

Managing Reservations

The Gantt data model manages reservations in a collection. The `IReservationCollection` interface contains methods for adding or removing reservations. The order of the reservations in an **IReservationCollection** is not important and therefore the API does not allow for managing the order of reservations in the model.

Accessing Reservations

A set of reservations can be obtained from the Gantt data model with the `IGanttModel.Reservations` property of the model.

To retrieve the reservations associated with a specified activity, use the `IActivity.Reservations` property of the `IActivity` interface.

To retrieve the reservations associated with a specified resource, use the `IResource.Reservations` property of the `IResource` interface.

The `IGanttModel.FindReservation` method can be used to find a reservation between an activity and a resource.

Adding Reservations

A set of reservations can be obtained from the Gantt data model with the `IGanttModel.Reservations` property of the model. The set returned allows you to add new reservations to the model using the **Add** method:

```
IGanttModel model = new SimpleGanttModel();
IActivity activity = model.NewActivity();
model.Activities.Add(activity);
IResource resource = model.NewResource();
model.Resources.Add(resource);
IReservation reservation = model.NewReservation(activity, resource);
model.Reservations.Add(reservation);
```

Note: *The activity and the resource must be added to the model before the reservation is added.*

The `AddRange` method is for adding several reservations at a time. This is a more efficient way of adding several reservations than calling **Add** several times.

Removing Reservations

A set of reservations can be obtained from the Gantt data model with the `IGanttModel.Reservations` property of the model. The set that is returned allows you to remove existing reservations from the model using the **Remove** method. The following C# code sample shows how to remove all the reservations whose activity unique identifier is A1:

```
IGanttModel model = ...;
IActivity activity = model.FindActivity("A1");
IReservation[] reservations = new IReservation[activity.Reservations.Count];
activity.Reservations.CopyTo(reservations, 0);
model.Reservations.RemoveRange(reservations);
```

The `RemoveRange` method is for removing several constraints at a time. This is a more efficient way of removing several constraints than calling **Remove** several times.

See Also

Managing Activities / Managing Resources / Managing Constraints

Optimizing Gantt Data Model Modifications

Each time a Gantt data model is modified, the controls bound to it are notified of the changes so they can reflect these changes. This notification can be time consuming if many controls are connected to the Gantt data model, or if the modification of the model affects many scheduling entities.

Binding Controls

If you have a large dataset, you are recommended to populate your data model before the chart is bound. This will reduce the time spent in notifications.

When several controls are bound to the same data model, but only a few of them are visible at the same time, you can disconnect controls that are not visible to decrease the time spent in notifications. Later, when these controls will become visible, you will have to connect them to the Gantt data model.

Using the BeginUpdate / EndUpdate mechanism

You can also use the update mechanism provided by the Gantt data model to disable events fired by the model while it is being populated. See the `IGanttModel.BeginUpdate` and `IGanttModel.EndUpdate` methods for details.

Displaying Scheduling Data

IBM® ILOG® Gantt for .NET provides several controls to display a Gantt data model. The table below lists the controls that can be bound to a Gantt data model, as well as the property that must be used to do the binding:

Control	Property	Description
GanttChart	GanttModel	A Gantt chart is a chart that displays activities.
ScheduleChart	GanttModel	A Schedule chart is a chart that displays the schedule of resources.
ReservationChart	GanttModel	A Reservation chart is a chart that displays one reservation per row.
ActivityTable	GanttModel	A table that displays the hierarchy of activities.
ResourceTable	GanttModel	A table that displays the hierarchy of resources.
ReservationTable	GanttModel	A table that displays the reservations of a Gantt model.
LoadChart	GanttModel	A chart that displays the load of a resource.
CalendarView	GanttModel	A calendar control displaying activities of a Gantt model.

The following C# code shows how to display a Gantt data model using a **GanttChart** control:

```
IGanttModel model = new SimpleGanttModel();
GanttChart chart = new GanttChart();
chart.GanttModel = model;
```

See Also

Displaying Scheduling Data Using Tables | Displaying Scheduling Data Using Gantt Charts | Displaying Activities in the Calendar View | Displaying the Load of a Resource | Creating Custom Gantt Representations

Creating Custom Gantt Data Models

IBM® ILOG® Gantt for .NET provides a ready-to-use implementation of its abstract data model. If this implementation does not cover your needs, it can be extended or a custom model can be created by implementing the abstract data model directly.

In This Section

Extending the In-Memory Implementation

Explains how to extend the ready-to-use implementation of the Gantt data model provided with the library.

Implementing the Gantt Data Model Interfaces

Describes how to implement the interfaces of the Gantt data model.

Related Sections

Introducing the Gantt Data Model In-Memory Implementation

Describes the in-memory implementation of the data model classes.

Customizing XML Serialization or Deserialization

Explains how to customize the serialization and deserialization of scheduling data.

Extending the In-Memory Implementation

The ready-to-use implementation of the Gantt data model is a straightforward implementation of the abstract data model. This implementation can be used to represent standard scheduling data. You may need to extend it to meet specific requirements.

In This Section

Extending the Model without Coding

Describes how to add properties to scheduling entities without coding.

Creating a Subclass of SimpleGanttModel

Describes how to create a subclass of SimpleGanttModel to manage custom activities.

Creating a Subclass of SimpleActivity

Describes how to create a subclass of SimpleActivity to add custom properties.

Related Sections

Introducing Gantt Data Model Interfaces

Describes the data model interfaces of ILOG Gantt for .NET.

Introducing the Gantt Data Model In-Memory Implementation

Describes the in-memory implementation of the data model classes.

Extending the Model without Coding

The ready-to-use implementation of the Gantt data model can be customized without coding by dynamically adding properties to scheduling entities. Those custom properties will be seen by the library as regular .NET properties.

The custom properties can be accessed from the Gantt data model using the following properties:

Property	Description
CustomActivityProperties	Gets the collection of custom property descriptors for activities.
CustomResourceProperties	Gets the collection of custom property descriptors for resources.
CustomConstraintProperties	Gets the collection of custom property descriptors for constraints.
CustomReservationProperties	Gets the collection of custom property descriptors for reservations.

Note: All these properties are available in the property window of Visual Studio .NET and it is not necessary to code to extend the data model.

Code Sample

In the sample, a property named `Priority` is added to an activity. The following enumeration defines the property type:

```
public enum ActivityPriority
{
    Undefined,
    Low,
    Medium,
    High
}
```

The C# code sample below adds the property to the model:

```
SimpleGanttModel model = new SimpleGanttModel();
model.CustomActivityProperties.Add("Priority",
                                  typeof(ActivityPriority),
                                  ActivityPriority.Undefined);
```

Accessing Custom Properties by Code

To access a custom property defined on a schedule entity, use the indexer of the entity. The following C# code sample shows how to change the value of the `Priority` property defined above:

```
SimpleGanttModel model = new SimpleGanttModel();
model.CustomActivityProperties.Add("Priority",
                                  typeof(ActivityPriority),
                                  ActivityPriority.Undefined);
SimpleActivity activity = (SimpleActivity)model.NewActivity();
activity["Priority"] = ActivityPriority.Low;
```

```
model.Activities.Add(activity);
```

See Also

Implementing the Gantt Data Model Interfaces / Creating a Subclass of SimpleGanttModel / Creating a Subclass of SimpleActivity

Creating a Subclass of SimpleGanttModel

This section explains how to create a subclass of the SimpleGanttModel class that manages custom scheduling entities. The C# code fragments are taken from the **CustomGantt** sample located in the directory:

```
<install-dir>\Samples\QuickStart\CustomGantt
```

In particular, this section shows how to customize activities by adding .NET properties.

When you customize the ready-to-use implementation of the Gantt data model, you must create a subclass of SimpleGanttModel, because this class acts as a factory for the scheduling entities (activities, resources, constraints, and reservations).

The factory methods that create the scheduling entities are as follows:

Method	Description
NewActivity	Creates and returns a new activity.
NewResource	Creates and returns a new resource.
NewConstraint	Creates and returns a new constraint.
NewReservation	Creates and returns a new reservation.

In this sample, only activities are customized. The CustomGanttModel can be coded as follows:

```
public class CustomGanttModel : SimpleGanttModel
{
    ...
    public override IActivity NewActivity() {
        return new CustomActivity(this);
    }
    ...
}
```

Whereas the **SimpleGanttModel** class is used to manage activities that use the SimpleActivity class, the CustomGanttModel is used with a subclass of **SimpleActivity** called CustomActivity. The CustomActivity class is described in *Creating a Subclass of SimpleActivity*.

Creating a Subclass of SimpleActivity

In the sample, a property named Priority is added to an activity.

Adding a Gantt .NET Property

The priority of an activity is defined by the following enumeration:

```
public enum ActivityPriority
{
    Undefined,
    Low,
    Medium,
    High
}
```

The CustomActivity class adds a .NET property called Priority, which has as type the enumeration defined above.

The code of the setter for this property is typical:

1. Checks the new value against the current value to avoid unnecessary notification.
2. Creates the event data, specifying the activity that is to be modified, the property name, the new value of the property, and the old value of the property.
3. Notifies the model that the property is going to change.
4. Cancels the modification, if it is vetoed.
5. Sets the priority from the event data, so that a listener has a chance to modify the value.
6. Notifies the model that the property has changed.

Here is the C# code of the Priority property:

```
public class CustomActivity : SimpleActivity
{
    private ActivityPriority _priority = ActivityPriority.Undefined;
    ...
    [DefaultValue(ActivityPriority.Undefined)]
    [XmlAttribute("priority")]
    public ActivityPriority Priority
    {
        get
        {
            return _priority;
        }
        set
        {
            if (Priority == value)
                return;
            ActivitiesChangeEventArgs args =
                new ActivitiesChangeEventArgs(this, "Priority", value,
                _priority);
            OnActivityChanging(args);
            if (args.Cancel)
                return;
            _priority = (ActivityPriority)args.Value;
            OnActivityChanged(args);
        }
    }
}
```

```
    ...  
}
```

Notes: The *DefaultValue* attribute is used by the XML serializer to avoid serializing default values. The *XmlAttribute* attribute is used to control the name of the property that will be used during serialization. Omitting this attribute will cause the serializer to use the name of the property (**Priority**, first letter in uppercase) instead of the name specified by the attribute ("**priority**", first letter in lowercase).

The `Priority` property is now seen by the library as a standard activity property. It can be used anywhere that a property name is required, such as during XML serialization, in an expression, in a column-mapping name, and so on.

Adding a Standard .NET Property

You may want to add public properties to your subclass without considering them as properties specific to Gantt entities, that is, as indicating a property that will be seen by the library as part of a schedule entity.

The code of the setter is straightforward: no notification is needed here, since this property does not represent a Gantt property. The sample below adds a dummy property to show the use of the **GanttProperty** attribute:

```
public class CustomActivity : SimpleActivity  
{  
    private int _dummyProperty = -1;  
    ...  
    [GanttProperty(false)]  
    public int DummyProperty  
    {  
        get  
        {  
            return _dummyProperty;  
        }  
        set  
        {  
            _dummyProperty = value;  
        }  
    }  
    ...  
}
```

Note: The value of the **GanttProperty** attribute indicates to the library that this **.NET** property is not a Gantt property. It will not be serialized, nor used as a mapping name for table columns, and so on.

See Also

Extending the Model without Coding

Implementing the Gantt Data Model Interfaces

If the in-memory implementation of the Gantt data model provided through the `SimpleGanttModel` class and its related classes does not fit your needs, you can create your own Gantt data model by implementing the interfaces of the abstract model directly. This could be the case, for example, if your scheduling data is located in a database and you do not want to duplicate the data for the Gantt.

In This Section

Implementing the `IGanttModel` Interface

Explains how to implement the interface that describes the Gantt data model.

Implementing the Collection Interfaces of the Gantt Data Model

Explains how to implement the collection interfaces that describe the Gantt data model.

Implementing the Scheduling Entities

Explains how to implement the interfaces that describe activities, resources

Related Sections

Introducing Gantt Data Model Interfaces

Describes the data model interfaces of IBM ILOG Gantt for .NET.

Introducing the Gantt Data Model In-Memory Implementation

Describes the in-memory implementation of the data model classes.

Implementing the `IGanttModel` Interface

The `IGanttModel` interface acts as a container for its scheduling entities. It is responsible for:

- ◆ Creating scheduling entities.
- ◆ Accessing scheduling entities.
- ◆ Finding activities and resources through their unique identifiers.
- ◆ Finding constraints and reservations.
- ◆ Maintaining unique identifiers for activities and resources.
- ◆ Maintaining up-to-date relations between scheduling entities.
- ◆ Triggering events when scheduling entities are modified.
- ◆ Moving activities and resources.

The scheduling entities created by this model are valid for this model only: each schedule entity has a reference to the data model it belongs to. When a schedule entity is modified, it must notify its model of the modification to allow the model to trigger the event.

Creating Scheduling Entities

Implementing the factory methods of the **IGanttModel** interface is straightforward. Here is the implementation of these methods in the **SimpleGanttModel** class:

```
public class SimpleGanttModel : IGanttModel
{
    ...
    public override IActivity NewActivity()
    {
        return new SimpleActivity(this);
    }

    public override IResource NewResource()
    {
        return new SimpleResource(this);
    }

    public override IConstraint NewConstraint(
        IActivity fromActivity,
        IActivity toActivity,
        ConstraintType type)
    {
        return new SimpleConstraint(
            this,
            fromActivity as SimpleActivity,
            toActivity as SimpleActivity,
            type);
    }

    public override IReservation NewReservation(
        IActivity activity,
        IResource resource)
    {
        return new SimpleReservation(
            this,
            activity as SimpleActivity,
            resource as SimpleResource);
    }
    ...
}
```

Each method creates and returns a new instance of its corresponding schedule entity.

Accessing Scheduling Entities

To access scheduling entities, the **IGanttModel** interface defines the following properties:

```
interface IGanttModel
{
    ...
    IActivityCollection Activities { get; }
    IResourceCollection Resources { get; }
    IConstraintCollection Constraints { get; }
    IReservationCollection Reservations { get; }
    ...
}
```

You need to implement the collection interfaces `IActivityCollection`, `IResourceCollection`, `IConstraintCollection`, and `IReservationCollection`. Each collection must notify the Gantt model when it is modified. See *Implementing the Collection Interfaces of the Gantt Data Model*.

Finding Activities and Resources by Their Unique Identifiers

The model must maintain a structure that allows an activity or a resource to be found by its identifier. The search must be fast. A hash table is the easiest way to implement it. Here is the implementation of the `FindActivity` and `FindResource` methods in the `BaseGanttModel` class:

```
public IActivity FindActivity(string id)
{
    return (IActivity)_activityIDs[id];
}

public IResource FindResource(string id)
{
    return (IResource)_resourceIDs[id];
}
```

The hash table `_activityIDs` has keys that are the identifiers of activities. The values in the hash table are the activities themselves.

The hash table `_resourceIDs` has keys that are the identifiers of resources. The values in the hash table are the resources themselves.

These hash tables must be updated each time activities or resources are added or removed from the model or when they change their identifiers.

Finding Constraints and Reservations

The `FindConstraint` and `FindReservation` methods must be coded to allow a fast search of constraints and reservations into the Gantt model.

The `FindConstraint` method is used to locate a constraint between two activities.

The `FindReservation` method is used to locate a reservation between an activity and a resource.

Maintaining Unique Identifiers for Activities and Resources

Each time activities or resources are added to the model, their identifiers must be checked to make sure that they are unique.

Each time activities or resources are removed from the model, their identifiers must be removed from the structure that maintains the correspondence between an object and its identifier.

Each time an activity or a resource changes its identifier, the structure that maintains the correspondence between an object and its identifier must be updated.

Maintaining Up-to-Date Relationships Between Scheduling Entities

Each constraint references two activities. When activities are removed from the model, constraints that reference those activities must be removed from the model to leave the model in a consistent state.

Each reservation references an activity and a resource. When activities or resources are removed from the model, reservations that reference those activities or resources must be removed from the model to leave the model in a consistent state.

Triggering Events When Scheduling Entities are Modified

Each time a schedule entity is modified, its data model must be notified of the modification to trigger an event. Notification is done in two phases. First, an event is sent to notify that a change is going to occur. Then, another event is sent after the change has been completed. The following methods are responsible for triggering the events related to activities in the SimpleGanttModel class:

Method	Description
OnActivitiesChanging	Must be called when an activity is being modified. The method raises the IGanttModel.ActivitiesChanging event.
OnActivitiesChanged	Must be called after an activity has been modified. The method raises the IGanttModel.ActivitiesChanged event.

The scope of these methods is internal, because they are called from the SimpleActivity class each time an activity is modified.

Moving Activities and Resources

Moving an activity or a resource means changing its parent or its index in its parent-child collection. This operation is not equivalent to removing an activity or a resource and then adding it at a different location. When an activity or a resource is removed from the model, associated entities such as constraints and reservations are also removed from the model. This is not the case when an activity or a resource is moved: one event only is triggered when an activity or a resource is moved in the model.

Implementing the Collection Interfaces of the Gantt Data Model

The Gantt data model uses four collection interfaces that need to be implemented. Each collection is dedicated to a specific Gantt entity:

Class	Description
IActivityCollection	Represents a collection of activities.
IResourceCollection	Represents a collection of resources.

IConstraintCollection	Represents a set of constraints.
IReservationCollection	Represents a set of reservations.

Implementing a collection interface is straightforward: you can use one of the many existing collections of .NET Framework, such as an **ArrayList**. Each method of the interface that modifies the collection must notify the Gantt data model of the change.

The following C# code sample shows a possible implementation of the Add method of the **IResourceCollection** interface:

```
public int Add(IResource resource)
{
    ActivitiesChangeEventArgs args =
        new ResourcesChangeEventArgs(
            ResourcesAction.Add,
            Resource,
            Count,
            new IResource[] { resource });
    GanttModel.OnResourcesChanging(args);
    if (args.Cancel)
        return -1;
    _arrayList.Add(resource);
    UpdateParent(resource);
    GanttModel.OnResourcesChanged(args);
    return Count-1;
}
```

First, the event data representing the Add operation is created. The event data contains the following information:

- ◆ Resources are added.
- ◆ The parent resource of the new resource is `Resource`, that is, the resource that owns the collection.
- ◆ The new resources are added at the index `Count`, that is, at the end of the list.
- ◆ The resources that are added.

Then, the model is notified that a change occurs. If the operation is not canceled, the resource will be added to the internal list wrapped by the collection. For example, in the C# code sample above, the `_arrayList` member can be an **ArrayList**.

The call to `UpdateParent` should update the internal member of the resource that references its parent.

Finally, the model is notified that a change has occurred.

Implementing the other methods of the interfaces is done in a similar way.

Implementing the Scheduling Entities

The Gantt data model defines four scheduling entities. Each of these scheduling entities is represented by a specific interface:

Class	Description
IActivity	Represents an activity.
IResource	Represents a resource.
IConstraint	Represents a constraint.
IReservation	Represents a reservation.

Each interface is mainly composed of properties. The implementation of the property follows a typical schema:

- ◆ First, the event data that represents the property change operation is created.
- ◆ Then, the model is notified that a change is occurring.
- ◆ If the operation is not canceled, the member that represents the property has to be updated.
- ◆ Finally, if the operation is not canceled, the model is notified that a change has occurred.

The C# code sample below shows a possible implementation of the Info property of the **IActivity** interface:

```
public string Info
{
    get
    {
        return _info;
    }
    set
    {
        if (_info == value)
            return;
        ActivitiesChangeEventArgs args =
            new ActivitiesChangeEventArgs(this, "Info", value, _info);
        GanttModel.OnActivitiesChanging(args);
        if (args.Cancel)
            return;
        _info = (string)args.Value;
        GanttModel.OnActivitiesChanged(args);
    }
}
```

The implementation of other properties of the interfaces is performed in a similar way.

Listening to Gantt Data Model Events

Each time a Gantt data model is modified, events are raised to allow listeners to be notified of the modification.

The following kinds of events, each corresponding to a schedule entity, are fired by a Gantt data model:

- ◆ activity event
- ◆ resource event
- ◆ constraint event
- ◆ reservation event

A modification to the model is usually achieved in two phases. First, the model notifies its listeners that something is going to change, so that one of them can set a veto on the modification, if necessary. Then, the model notifies its listeners that the change has been completed.

In This Section

Catching Activity Events

Explains how to listen for modifications made to the activities of a Gantt data model.

Catching Resource Events

Explains how to listen for modifications made to the resources of a Gantt data model.

Catching Constraint Events

Explains how to listen for modifications made to the constraints of a Gantt data model.

Catching Reservation Events

Explains how to listen for modifications made to the reservations of a Gantt data model.

Catching Activity Events

Each time an activity of a Gantt data model is modified, the model triggers an activity event that contains the description of the change made to the model.

Listening for Activity Events

To listen for the modifications made to activities in a Gantt data model, use the `IGanttModel.ActivitiesChanging` and `IGanttModel.ActivitiesChanged` events, as shown in the following C# code sample:

```
IGanttModel model = new SimpleGanttModel();
model.ActivitiesChanging += new
ActivitiesChangeEventHandler(OnActivitiesChanging);
model.ActivitiesChanged += new
ActivitiesChangeEventHandler(OnActivitiesChanged);
```

Have `OnActivitiesChanging` and `OnActivitiesChanged` declared as follows:

```
public void OnActivitiesChanging(object sender, ActivitiesChangeEventArgs
arg);
public void OnActivitiesChanged(object sender, ActivitiesChangeEventArgs arg);
```

The `OnActivitiesChanging` method is called when a modification is going to occur to the activities of the data model.

The `OnActivitiesChanged` method is called after a modification to the activities of the data model has been completed.

Understanding Activity Events

The first argument received by the event handler is a reference to the Gantt data model that is modified.

The second argument received by the event handler is an `ActivitiesChangeEventArgs` instance that contains the description of the modification to the data model. The

ActivitiesChangeEventArgs.Action property contains the type of modification, as explained in the following table:

ActivitiesAction	Meaning
Add	Activities have been added to the Gantt model.
Delete	Activities have been deleted from the Gantt model.
PropertyChange	An activity had one of its properties changed.
Move	An activity has been moved.
Reset	Activities have changed completely.

Controlling Modifications

Before a modification occurs to the activities of a Gantt data model, the ActivitiesChanging event is fired. At this time, the modification to the model is not yet completed.

The event can be used to control the modification in the following ways:

- ◆ It can be canceled with the Cancel property of the ActivitiesChangeEventArgs class. For example, the following C# code shows how to prevent activities from being removed:

```
public void OnActivitiesChanging(object sender, ActivitiesChangeEventArgs
arg)
{
    if (arg.Action == ActivitiesAction.Delete)
        arg.Cancel = true;
}
```

- ◆ When the modification reflects a property change to an activity, the new value of the property can be adjusted before it is set. The following C# code prevents the **StartTime** property of an activity from being moved before a specified date:

```
public void OnActivitiesChanging(object sender, ActivitiesChangeEventArgs
arg)
{
    if (arg.Action == ActivitiesAction.PropertyChange && arg.Property ==
"StartTime")
    {
        DateTime startTime = (DateTime)arg.Value;
        DateTime limitTime = new DateTime(1, 1, 2000);
        if (startTime < limitTime)
            arg.Value = limitTime;
    }
}
```

Catching Resource Events

Each time a resource of a Gantt data model is modified, the model triggers a resource event that contains the description of the change that was made to the model.

Listening to Resource Events

To listen for the modifications made to resources in a Gantt data model, use the `IGanttModel.ResourcesChanging` and `IGanttModel.ResourcesChanged` events, as shown in the following C# code sample:

```
IGanttModel model = new SimpleGanttModel();  
model.ResourcesChanging += new  
ResourcesChangeEventHandler(OnResourcesChanging);  
model.ResourcesChanged += new ResourcesChangeEventHandler(OnResourcesChanged);
```

Have `OnResourcesChanging` and `OnResourcesChanged` declared as follows:

```
public void OnResourcesChanging(object sender, ResourcesChangeEventArgs arg);  
public void OnResourcesChanged(object sender, ResourcesChangeEventArgs arg);
```

The `OnResourcesChanging` method is called when a modification is going to occur to the resources of the data model.

The `OnResourcesChanged` method is called after a modification to the resources of the data model has been completed.

Understanding Resource Events

The first argument received by the event handler is a reference to the Gantt data model being modified.

The second argument received by the event handler is a `ResourcesChangeEventArgs` instance that contains the description of the modification to the data model. The `ResourcesChangeEventArgs.Action` property contains the type of modification, as explained in the following table:

ResourcesAction	Meaning
Add	Resources have been added to the Gantt model.
Delete	Resources have been deleted from the Gantt model.
PropertyChange	A resource had one of its properties changed.
Move	A resource has been moved.
Reset	Resources have changed completely.

Controlling Modifications

Before a modification occurs to the resources of a Gantt data model, the `ResourcesChanging` event is fired. At this time, the modification to the model has not yet been completed.

The event can be used to control the modification in the following ways:

- ◆ It can be canceled with the `Cancel` property of the `ResourcesChangeEventArgs` class. For example, the following C# code shows how to prevent resources from being removed:

```
public void OnResourcesChanging(object sender, ResourcesChangeEventArgs arg)
{
    if (arg.Action == ResourcesAction.Delete)
        arg.Cancel = true;
}
```

- ◆ When the modification reflects a property change to a resource, the new value of the property can be adjusted before it is set. The following C# code prevents the **MaxUnits** property of a resource from being set to a value greater than 1:

```
public void OnResourcesChanging(object sender, ResourcesChangeEventArgs arg)
{
    if (arg.Action == ResourcesAction.PropertyChange && arg.Property ==
        "MaxUnits")
    {
        float maxUnits = (float)arg.Value;
        if (maxUnits > 1)
            arg.Value = (float)1;
    }
}
```

Catching Constraint Events

Each time a constraint of a Gantt data model is modified, the model triggers a constraint event that contains the description of the change that was made to the model.

Listening to Constraint Events

To listen to the modifications made to constraints in a Gantt data model, use the `IGanttModel.ConstraintsChanging` and `IGanttModel.ConstraintsChanged` events, as shown in the following C# code sample:

```
IGanttModel model = new SimpleGanttModel();
model.ConstraintsChanging += new
ConstraintsChangeEventHandler(OnConstraintsChanging);
model.ConstraintsChanged += new
ConstraintsChangeEventHandler(OnConstraintsChanged);
```

Have `OnConstraintsChanging` and `OnConstraintsChanged` declared as follows:

Listening to Gantt Data Model Events

```
public void OnConstraintsChanging (object sender, ConstraintsChangeEventArgs  
arg);  
public void OnConstraintsChanged (object sender, ConstraintsChangeEventArgs  
arg);
```

The *OnConstraintsChanging* method is called when a modification is going to occur to the constraints of the data model.

The *OnConstraintsChanged* method is called after a modification to the constraints of the data model has been completed.

Understanding Constraint Events

The first argument received by the event handler is a reference to the Gantt data model being modified.

The second argument received by the event handler is a *ConstraintsChangeEventArgs* instance that contains the description of the modification to the data model. The *ConstraintsChangeEventArgs.Action* property contains the type of modification, as explained in the following table:

ConstraintsAction	Meaning
Add	Constraints have been added to the Gantt model.
Delete	Constraints have been deleted from the Gantt model.
PropertyChange	A constraint had one of its properties changed.
Reset	Constraints have changed completely.

Controlling Modifications

Before a modification occurs to the constraints of a Gantt data model, the *ConstraintsChanging* event is fired. At this time, the modification to the model has not yet been completed.

The event can be used to control the modification in the following ways:

- ◆ It can be canceled with the *Cancel* property of the *ConstraintsChangeEventArgs* class. The following C# code shows how to prevent constraints from being removed:

```
public void OnConstraintsChanging(object sender, ConstraintsChangeEventArgs  
arg)  
{  
    if (arg.Action == ConstraintsAction.Delete)  
        arg.Cancel = true;  
}
```


- ◆ When the modification reflects a property change to a constraint, the new value of the property can be adjusted before it is set. The following C# code forces the **Type** property of a constraint to a specific value:

```
public void OnConstraintsChanging(object sender, ConstraintsChangeEventArgs
arg)
{
    if (arg.Action == ConstraintsAction.PropertyChange && arg.Property ==
"Type")
        arg.Value = ConstraintType.EndToStart;
}
```

Catching Reservation Events

Each time a reservation of a Gantt data model is modified, the model triggers a reservation event that contains the description of the change that was made to the model.

Listening to Reservation Events

To listen for the modifications made to reservations in a Gantt data model, use the `IGanttModel.ReservationsChanging` and `IGanttModel.ReservationsChanged` events, as shown in the following C# code sample:

```
IGanttModel model = new SimpleGanttModel();
model.ReservationsChanging += new
ReservationsChangeEventHandler(OnReservationsChanging);
model.ReservationsChanged += new
ReservationsChangeEventHandler(OnReservationsChanged);
```

Have `OnReservationsChanging` and `OnReservationsChanged` declared as follows:

```
public void OnReservationsChanging (object sender, ReservationsChangeEventArgs
arg);
public void OnReservationsChanged (object sender, ReservationsChangeEventArgs
arg);
```

The `OnReservationsChanging` method is called when a modification is going to occur to the reservations of the data model.

The `OnReservationsChanged` method is called after a modification to the reservations of the data model has been completed.

Understanding Reservation Event

The first argument received by the event handler is a reference to the Gantt data model being modified.

The second argument received by the event handler is a `ReservationsChangeEventArgs` instance that contains the description of the modification to the data model. The

ReservationsChangeEventArgs.Action property contains the type of modification, as explained in the following table:

ReservationsAction	Meaning
Add	Reservations have been added to the Gantt model.
Delete	Reservations have been deleted from the Gantt model.
PropertyChange	A reservation had one of its properties changed.
Reset	Reservations have changed completely.

Controlling Modifications

Before a modification occurs to the reservations of a Gantt data model, the ReservationsChanging event is fired. At this time, the modification to the model has not yet been completed.

The event can be used to control the modification in the following ways:

- ◆ It can be canceled with the Cancel property of the ReservationsChangeEventArgs class. The following C# code shows how to prevent reservations from being removed:

```
public void OnReservationsChanging(object sender,
ReservationsChangeEventArgs arg)
{
    if (arg.Action == ReservationsAction.Delete)
        arg.Cancel = true;
}
```

- ◆ When the modification reflects a property change to a reservation, the new value of the property can be adjusted before it is set. The following C# code prevents the **Units** property of a reservation from being set to a value greater than 1:

```
public void OnReservationsChanging(object sender,
ReservationsChangeEventArgs arg)
{
    if (arg.Action == ReservationsAction.PropertyChange && arg.Property ==
"Units")
    {
        float units = (float)arg.Value;
        if (units > 1)
            arg.Value = (float)1;
    }
}
```

Displaying Scheduling Data Using Gantt Charts

The library features three high-level controls: the Gantt Chart control, the Schedule Chart control, and the Reservation Chart control. These controls are built upon lower-level controls and provide the most commonly used types of display.

The API of these controls is based on the GanttChart, ScheduleChart, and ReservationChart classes, which are subclasses of HierarchyChart. In this section, the term *Gantt chart* is used to refer to a Gantt Chart, a Resource Chart, or a Reservation Chart.

Together with the IGanttModel interface, these controls make up the main classes of the IBM® ILOG® Gantt for .NET API.

In This Section

Introducing the Gantt Chart, Schedule Chart, and Reservation Chart Controls

Describes the Gantt Chart, Schedule Chart, and Reservation Chart controls.

Displaying Scheduling Data

Describes how to connect the Gantt Chart controls to data.

Accessing the Lower-Level Controls

Describes how you can access the subcontrols of Gantt Chart controls and why you might need to.

Displaying Scheduling Data Using Gantt Charts

Modifying the Appearance of Gantt Chart Controls

Describes how to use properties to change the appearance of Gantt Chart controls.

Using the Predefined Behavior of the Gantt Chart Controls

Describes the predefined behavior of Gantt Chart controls.

Expanding or Collapsing Rows

Describes the methods for expanding or collapsing rows in Gantt Chart controls.

Grouping, Filtering and Sorting Rows

Describes how to sort and filter rows in the Gantt Chart controls.

Controlling the Displayed Time Interval

Describes how to modify and control the displayed time interval.

Related Sections

Displaying Scheduling Data Using Tables

Describes how to display scheduling data in tables using the IBM ILOG Gantt for .NET controls.

Creating Custom Gantt Representations

Describes how to develop custom components by assembling and connecting Gantt controls together.

Introducing the Gantt Chart, Schedule Chart, and Reservation Chart Controls

The Gantt Chart, Schedule Chart, and Reservation Chart controls are divided into three areas:

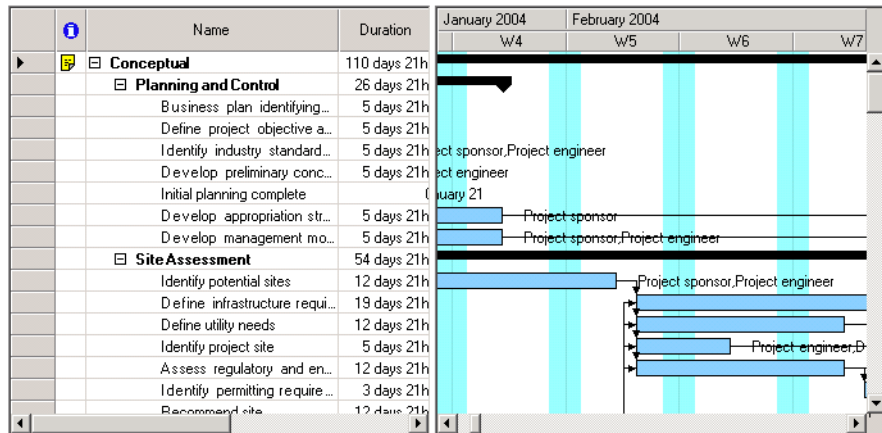
- ◆ The left part of the chart is a table view, an instance of the GanttTable class.
- ◆ The right part of the chart is a Gantt Sheet, an instance of the GanttSheet class.
- ◆ The area just above the Gantt sheet has a zoomable time scale, an instance of the TimeScale class.

A standard, adjustable splitter separates the left part from the right part.

The Gantt Chart Control

In the Gantt Chart control, each row represents an activity. Each column of the table displays a property of the activity. Each row in the Gantt sheet represents the duration of the activity. A row can also display other properties of the activity, such as start time, end time, or resources assigned to this activity.

The following illustration shows a Gantt chart.



In the default implementation, activities that have no child activities are displayed as plain horizontal bars. Activities that have child activities are displayed as horizontal bars of a different color, delimited by special symbols at the end. These attributes are completely customizable. See *Representing Activity Bars in Gantt Sheets* for details.

In the Gantt sheet, constraints between activities are represented by directional polyline links. The type of the constraint determines how the link is attached to the activity bars.

In the default implementation of the Gantt chart, the resources assigned to an activity are represented in the Gantt sheet on the right of an activity bar and also in the **Resources** column of the table.

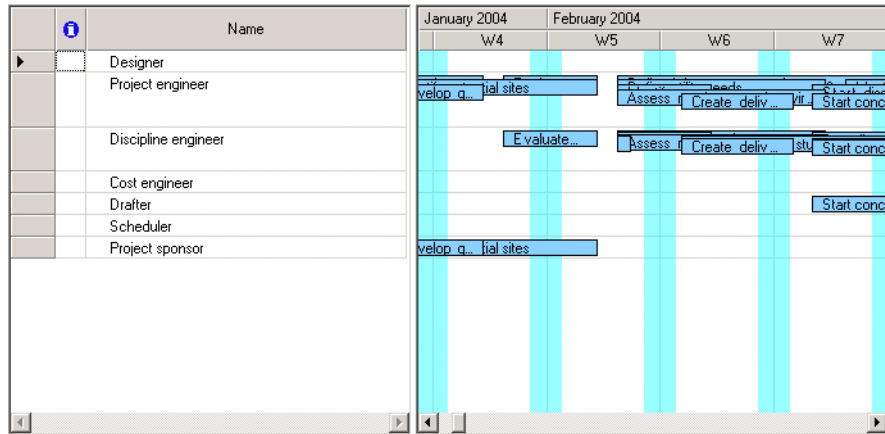
In the Gantt Chart control, the table is an instance of the ActivityTable class; the Gantt sheet is an instance of the ActivitySheet class.

The Schedule Chart Control

In the Schedule Chart, each row represents a resource. Each column in the table displays a property of the resource. Each row in the Gantt sheet contains 0, 1, or more bars to represent the activities for which the matching resource is reserved.

The following illustration shows a Schedule chart.

Displaying Scheduling Data Using Gantt Charts



Because the same resource can be reserved for more than one activity during the same time span (see *reservation*), it could happen that several activity bars occupy the same horizontal area in the same row. To address this problem, a specific activity layout algorithm is used to position the bars for the best legibility. Three different layout algorithms are provided to manage potentially overlapping reservation graphics. See *Activity Layout Style in a Schedule Sheet* for more information.

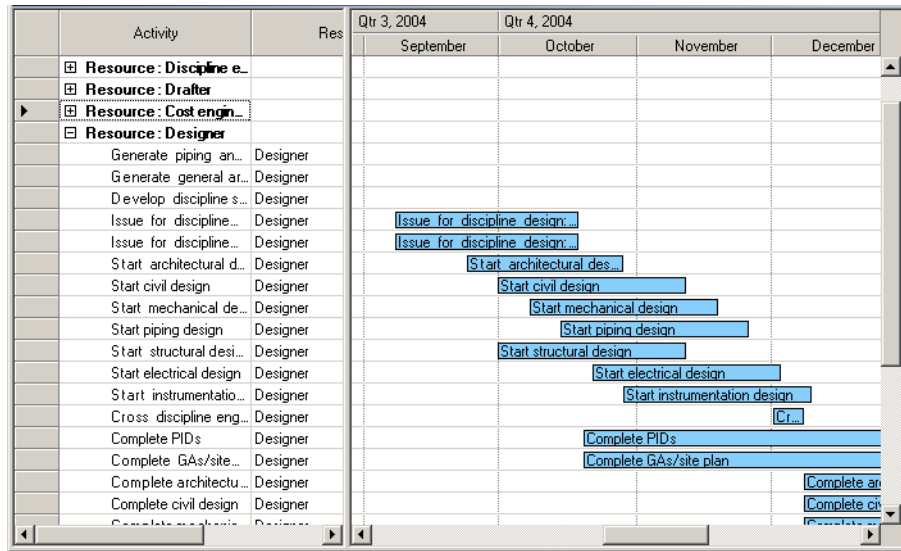
In the general case, one activity may reserve several resources and appear as several activity bars in the Schedule Chart. For this reason, constraints between activities are not displayed in the Schedule Chart.

In the Schedule Chart control, the table is an instance of the `ResourceTable` class; the Gantt sheet is an instance of the `ScheduleSheet` class.

The Reservation Chart Control

In the Reservation Chart, each row represents a reservation. Each column in the table displays a property of the reservation. Each row in the Gantt sheet contains 0, 1, or more bars to represent the corresponding reservation.

The following illustration shows a Reservation chart.



In the Reservation Chart control, the table is an instance of the ReservationTable class; the Gantt sheet is an instance of the ReservationSheet class.

Displaying Scheduling Data

In order to display information, the Gantt Chart, Schedule Chart and Reservation Chart controls must be connected to a Gantt data model, an instance of the IGanttModel interface.

The Gantt data model associated with a Gantt control can be set using the GanttModel property of the HierarchyChart class, the base class of the GanttChart, ScheduleChart and ReservationChart classes.

Here is a small C# example:

```
GanttChart myGanttChart = new GanttChart();
IGanttModel model = new SimpleGanttModel();
myGanttChart.GanttModel = model;
```

When the control is connected to a data model, the control listens for events from the Gantt model and is updated for each modification of the data model. For example, when an activity is added to or removed from the data model, a corresponding row will be added or removed in the Gantt Chart control.

For more information on the Gantt data model, see *Creating and Using Gantt Data Models*.

Displaying Scheduling Data Using Gantt Charts

To modify the scheduling data displayed in a Gantt Chart or Schedule Chart control by code you do not use methods from the **ScheduleChart** or **GanttChart** class. You use the classes and methods of the data model.

See Also *Creating and Using Gantt Data Models*

Accessing the Lower-Level Controls

The **HierarchyChart** class, the base class for the **GanttChart**, **ScheduleChart**, and **ReservationChart** classes, is built by assembling three subcontrols:

- ◆ A table (instance of **GanttTable**)
- ◆ A Gantt sheet (instance of **GanttSheet**)
- ◆ A time scale (instance of **TimeScale**)

The most important properties of these subcontrols are available directly at the **HierarchyChart** level. For example, the collection of columns in the table can be accessed directly through the **TableColumns** property of the **HierarchyChart** class.

In some cases, you may need to access the subcontrols. The **HierarchyChart** class provides the following properties for this purpose:

- ◆ The **GanttTable** property is a read-only property that returns the table.
- ◆ The **GanttSheet** property is a read-only property that returns the Gantt sheet.
- ◆ The **TimeScale** property is a read-only property that returns the time scale.

Thus, assuming that `chart` is an instance of the class **GanttChart**, **ScheduleChart** or **ReservationChart**, the following lines are equivalent:

```
chart.TableColumns  
chart.GanttTable.Columns
```

Note: *The WebForm version of the **HierarchyChart** does not give access to its subcontrols.*

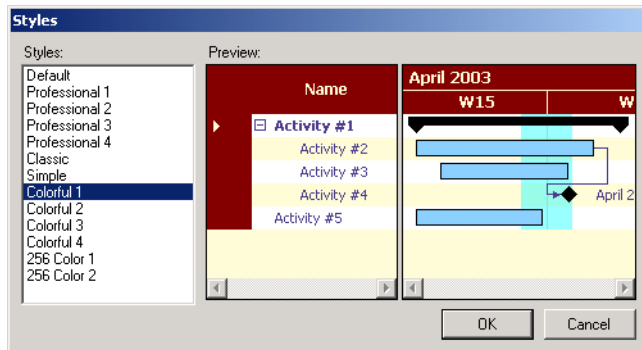
See Also *Introducing the Activity, Resource, and Reservation Tables / Introducing the Activity Sheet, Schedule Sheet, and ReservationSheet / Using Time Scales*

Modifying the Appearance of Gantt Chart Controls

The Gantt Chart, Schedule Chart, and Reservation Chart controls define several properties for changing their appearance.

Inside Visual Studio .NET you can choose between predefined styles for the appearance of the chart with the Auto Format functionality. Once you have dragged a Gantt Chart, a Schedule Chart, or a Reservation Chart into the Visual Designer, you can right-click it and choose Auto Format from the menu. In the dialog box, you can choose from several predefined color schemes.

The following illustration shows the Auto Format dialog box:



Since the controls are made of a Gantt table and a Gantt sheet, some properties apply to the table, some properties apply to the sheet, and some properties apply to both the table and the sheet. For example, changing the **BackColor** property of the Gantt Chart control will change the back color of the table and the back color of the sheet. You can have a different back color for the table and the sheet by changing the **BackColor** property of the underlying controls individually.

The following tables show the appearance properties of the Gantt Chart, Schedule Chart and Reservation Chart controls.

Color Properties

Property	Description
BackColor	The color used for the background of rows in the table and for the background of rows in the sheet.
AlternatingBackColor	The color used for alternating rows in the table and the Gantt sheet. By default, this color is the same as the BackColor property.
ForeColor	The color of text in the table and in the Gantt sheet.
BackgroundColor	The color used for the area of the table and Gantt sheet that is not made up of rows.
HeaderBackColor	The background color of the column and row headers of the table and of the background color of the time scale.
HeaderForeColor	The color of text in the column and row headers of the table and of text in the time scale.
SelectionBackColor	The background color of selected rows in the table and the Gantt sheet.
SelectionForeColor	The color of text for selected rows in the table and the Gantt sheet.
ConstraintsColor	The color of constraint links in the Gantt sheet.
GridLineColor	The color of horizontal grid lines in the table and the Gantt sheet.

Font Properties

Property	Description
Font	The font used for text in the Gantt table cells and in the Gantt sheet.
HeaderFont	The font used in the column headers of the table as well as in the time scale.

Horizontal Grid

The horizontal grid lines that separate the rows of the chart and the columns of the table are controlled by the following properties:

Property	Description
GridLineColor	The color of horizontal grid lines.
GridLineStyle	The style of horizontal grid lines.

Vertical Time Grid

The Gantt sheet can display vertical grid lines or areas that separate time periods on the time scale or that display nonworking time with a specific appearance.

In addition the Gantt sheet can display vertical grid lines that indicate some specific date of your project, such as the current date. The following properties control time grids and date indicators.

Property	Description
TimeGrids	A collection of time grids.
DateIndicators	A collection of vertical grid lines to mark specific dates.
VerticalGridToBottom	Indicates whether vertical grid lines are drawn in the area of the sheet that does not contain rows.

For more information see *Using Time Grids and Date Indicators*.

Splitter

The splitter that separates the Gantt table and the Gantt sheet can be controlled through the following properties:

Property	Description
SplitterPosition	The position of the splitter.
SplitterWidth	The width of the splitter.

Miscellaneous Appearance Properties

Property	Description
BorderStyle	The style of the border of the control.
BackgroundImage	An image that is displayed in the background of the control.
FlatStyle	The flatness style of the control.

Appearance of Activity Bars

The appearance of the rectangular bars that represent activities in the Gantt sheet can be completely customized by the styling mechanism. This styling mechanism is fully described in *Representing Activity Bars in Gantt Sheets*.

Appearance of Table Columns

The appearance of each column of the table can also be customized. See *Modifying the Appearance of Table Columns* for a detailed description.

Right-to-left Mode

All the controls can be used in right-to-left mode for Arabic and other languages that are written right-to-left. When the **RightToLeft** property of the Gantt Chart or Schedule Chart control is set to **RightToLeft.Yes**, the table is displayed on the right and the Gantt sheet on the left.

Note that bidirectional features of Windows® are only available in a bidirectional Microsoft® Windows® environment, such as an Arabic version of Microsoft® Windows®.

Using the Predefined Behavior of the Gantt Chart Controls

The Gantt Chart, Schedule Chart and Reservation Chart controls are built from a Gantt table, a Gantt sheet, and a time scale. The behavior of these high level controls is then defined through the behavior of the table, the Gantt sheet, and the time scale.

Editing Table Cells

To learn about how to edit the values in the Gantt table and modify the predefined cell editors, see *Editing Values in the Table*.

Manipulating Rows or Columns

To learn about how to resize rows or columns and how to change the order of columns in the table with the mouse pointer, see *Using Predefined Behavior to Manipulate Rows and Columns*.

Editing Activity Bars

To learn about editing the activity bars in the Gantt sheet, see *Using the Predefined Behavior of Gantt Sheet Controls*.

Zooming and Panning Using Time Scale

To learn about how to use the time scale with the mouse pointer for zooming and panning, see *Using the Predefined Behavior of Time Scales*.

Resizing Hierarchy Charts Areas

In addition, the GanttChart, ScheduleChart and ReservationChart classes contain a splitter that allows you to allocate more or less space for viewing the table and the sheet. Double-click the splitter to make the table fully visible.

Expanding or Collapsing Rows

The GanttChart class displays the hierarchical structure of activities, such that each row represents one activity. Similarly, the ScheduleChart class displays the hierarchical tree of resources, such that each row represents a resource.

In both charts, rows are numbered from index 0. Row numbering ignores vertical scrolling and is not affected by the current vertical scrolling position of the chart.

In the rest of this section, the activities in a Gantt chart and the resources in a Schedule chart are referred to as *data node*. In this way, the common behavior of both charts can be concisely described.

The following terms refer to the visibility of data nodes and the rows they are displayed on.

◆ *visible data node*

A node represented by a row that is only visible to the application user if the display area is large enough. A visible data node has all its ancestors expanded.

◆ *expanded data node*

Displaying Scheduling Data Using Gantt Charts

A node that shows its child nodes. A collapsed node may or may not be visible, depending on whether its parent node is expanded or not.

- ◆ *collapsed data node*

A node that hides its child nodes. A collapsed node may or may not be visible, depending on whether its parent node is expanded or not.

- ◆ *displayed data node*

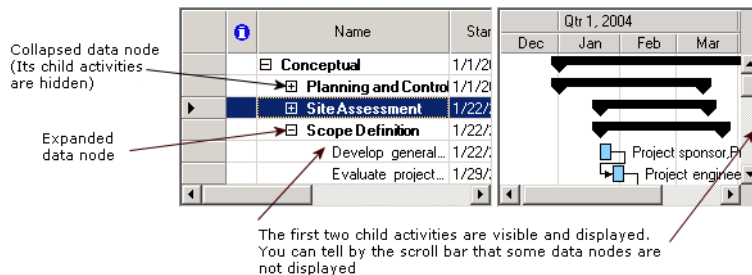
A node that is both visible, that is, its parent node is expanded, and currently within the display area, where it can be seen.

Note: Scrolling through a window changes the display status of a row, not its visibility status.

- ◆ *hidden data node*

A node that is the opposite of visible. It is a child of a collapsed parent and is not represented by a row.

The following illustration shows the different types of nodes:



The classes GanttChart and ScheduleChart have in common methods and properties for controlling the visibility of data nodes:

Member	Description
RowCount	To get the number of rows.
IsRowExpanded	To indicate whether the row is expanded.
ExpandRow	To expand a row.
CollapseRow	To collapse a row.
ExpandAll	To expand a row and all its children recursively.

CollapseAll	To collapse a row and all its children recursively.
ShowOutlineLevel	To make the nodes with the specified outline visible and to hide the lower levels.

The **GanttChart** class has methods for controlling the visibility of an activity:

Member	Description
IsExpanded	To indicate whether an activity is expanded.
Expand	To expand an activity.
Collapse	To collapse an activity.
IsRowVisible	To indicate whether an activity is visible.
ExpandAll	To expand an activity and all its subactivities.
CollapseAll	To collapse an activity and all its subactivities.

The **ScheduleChart** class has methods for controlling the visibility of a resource:

Member	Description
IsExpanded	To indicate whether a resource is expanded.
Expand	To expand a resource.
Collapse	To collapse a resource.
IsRowVisible	To indicate whether a resource is visible.
ExpandAll	To expand a resource and all its subresources.
CollapseAll	To collapse a resource and all its subresources.

You can use specific methods and properties to scroll the chart vertically to make sure that a data node is in the displayed area:

Member	Description
FirstVisibleRow	To get or set the index of the first row displayed in the chart.
EnsureRowVisible	To scroll the chart, so that the specified row is in the displayed area.

Grouping, Filtering and Sorting Rows

The `GanttChart`, `ScheduleChart` and `ReservationChart` controls provide capabilities for grouping, filtering and sorting data. The grouping allows you to create a hierarchy into the data. The filtering allows you to show only specific rows of the table. The sorting allows you to sort the rows of the table depending on specific criteria.

Grouping Rows

Using the `SetRowGroup` method of the `HierarchyChart`, you can group rows based on their column value. The `IRowGroup` interface is used to define a hierarchy of nodes into a `TreeModelView`.

The following C# code sample shows how to group reservations having the same resource property in an **ReservationChart**:

```
ReservationChart chart = new ReservationChart();  
chart.SetRowGroup = new DefaultRowGroup("Resource");
```

Note that once the group has been set, the table will update groups as the model gets modified.

You can create your own group by implementing the **IRowGroup** interface.

The grouping is implemented by the **TreeModelView** class, a wrapper for a tree model (see `ITreeModel`). For details about filtering, see the **TreeModelView** class.

Filtering Rows

Using the `RowFilter` property of the `HierarchyChart`, you can specify subsets of rows based on their column values. The **RowFilter** property is an expression that is evaluated for each row of the table to check whether the row should be displayed by the table.

The following C# code sample shows how to display only milestone activities scheduled after the 1/1/2000 in an **GanttChart**:

```
GanttChart chart = new GanttChart();  
chart.RowFilter = "Milestone && StartTime > #1/1/2000#";
```

Note that once the filter has been set, the table will stop displaying rows that do not satisfy the filter. Now if the data model is modified, only rows satisfying the filter criteria will be displayed by the chart.

If your filter criteria cannot be expressed using an expression, you can create your own filter by implementing the `IRowFilter` interface, and use the `SetRowFilter` method of the **GanttChart** class to apply the filter:

```
public class MyRowFilter : IRowFilter
```



```

{
    public bool FilterRow(TreeModelView view, object row)
    {
        // Returns true to make the row visible; otherwise, false
    }
}

GanttChart chart = new GanttChart();
IRowFilter filter = new MyRowFilter();
chart.SetRowFilter(filter);

```

Filtering is implemented by the **TreeModelView** class, a wrapper for a tree model (see `ITreeModel`). For details about filtering, see the `TreeModelView` class.

Sorting Rows

The `SortRows` methods of the `HierarchyChart` class provide means to sort the rows displayed by the table without modifying the Gantt Data Model being displayed. Unlike the filtering feature, which is set once and then applied until unset, sorting is done by calling the **SortRows** methods. The following C# code shows how to sort the activities displayed by a `GanttChart` according to their **StartTime** property:

```

GanttChart chart = new GanttChart();
chart.GanttModel = model; // Assume the model contains data.
chart.SortRows("StartTime");

```

Resetting Group, Filter and Sort Order

Once the table has been filtered or sorted, to remove the group and the filter and to restore the original order of the Gantt Data Model entities, use the `ResetGanttTableView` method.

Controlling the Displayed Time Interval

The Gantt Chart and Schedule Chart define the following properties for controlling the displayed time interval:

Property	Description
<code>FirstVisibleTime</code>	Gets or sets the first visible time of the Gantt Sheet.
<code>VisibleDuration</code>	Gets or sets the visible duration of the Gantt Sheet.
<code>LastVisibleTime</code>	Gets or sets the last visible time of the Gantt Sheet.
<code>VisibleTimeInterval</code>	Gets or sets the time interval displayed by the Gantt sheet in one single operation.

Displaying Scheduling Data Using Gantt Charts

The `GanttSheet` class displays a horizontal scroll bar for scrolling in time. The minimum and maximum time of this scroll bar is automatically defined by the content of the Gantt data model displayed by the Gantt sheet. The minimum time is the earliest start time of the activities in the model. The maximum time is the latest end time of the activities in the model. Thus, when the Gantt sheet displays the minimum and maximum values of the scroll bar, all the activities are visible.

The minimum and maximum time is controlled by the `TimeBounds` property of **`GanttSheet`**.

A margin can be added around the minimum and maximum time. This margin is defined in pixels by the `TimeMargin` property. This margin is useful for making the text around activity bars more easily visible.

Although the time scroll bar has a minimum and maximum time, the Gantt sheet can display dates before or after this minimum and maximum scrolling time. Clicking the scroll bar arrows allows scrolling in time before and after the minimum and maximum values.

When the Gantt sheet is scrolled in time with the horizontal scroll bar or the Time Scale, the new time period is instantaneously displayed in the Gantt sheet with each dragging movement of the mouse. You can change this behavior by setting the `InstantTimeScrolling` property to **`false`**. In this case, the new time period is displayed only when the mouse is released.

To learn how several controls for displaying time can be synchronized, see *Synchronizing the Time of Several Controls*.

Displaying Scheduling Data Using Tables

IBM® ILOG® Gantt for .NET provides many controls for displaying various representations of scheduling data. Representing data using tables is a compact way of displaying information.

IBM ILOG Gantt for .NET provides three types of table for displaying scheduling information:

- ◆ An activity table
- ◆ A resource table
- ◆ A reservation table

In this section, the term *Gantt table* is used to refer to an activity table a resource table, or a reservation table.

Note: The *GanttTable* class inherits from a *Generic Table* control implemented by the class *TreeTable* in the namespace *ILOG.Views.Windows.Forms*.

In This Section

Introducing the Activity, Resource, and Reservation Tables

Describes the content of the Activity, Resource, and Reservation Tables.

Connecting a Gantt Table to a Gantt Data Model

Describes how to display information in a Gantt table.

Modifying the Appearance of a Gantt Table

Describes the properties for customizing the appearance of a Gantt table.

Managing Columns of a Gantt Table

Describes how to manage the columns of a Gantt table.

Modifying the Appearance of Table Columns

Describes the properties for customizing the appearance of the columns of a Gantt table.

Editing Values in the Table

Describes how to make values in a cell of a table editable and how to customize editors.

Default Columns for Activity, Resource, and Reservation Tables

Describes the columns available by default when you create a Gantt chart or a Gantt table.

Dialog Box for Editing the Columns of a Gantt Table

Describes how to edit columns through the `GanttTableColumnDialog` class.

Using Predefined Behavior to Manipulate Rows and Columns

Describes the predefined behavior and keyboard mapping of the table.

Expanding or Collapsing Rows

Describes the API for expanding and collapsing rows of the table.

Scrolling the Gantt Table

Describes the API for scrolling the table.

Getting and Setting the Current Cell

Describes the concept of current cell and how to handle this object.

Controlling Selection in the Table

Describes the selection API of the table.

Hit Testing in the Gantt Table

Describes hit testing methods in the table.

Grouping, Filtering and Sorting Rows

Describes grouping, filtering and sorting methods in the table.

Introducing the Activity, Resource, and Reservation Tables

The following table lists the tables available to display Gantt data:

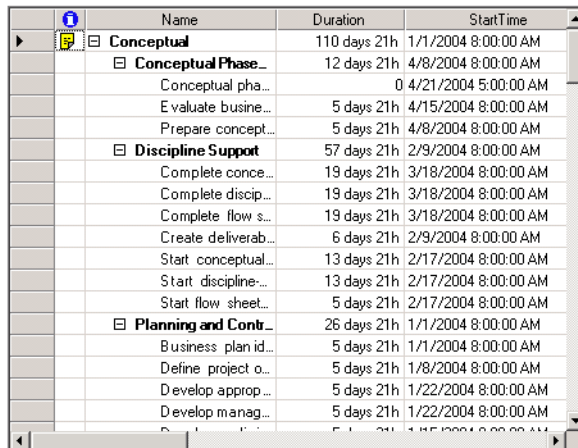
Class	Description
ActivityTable	A table to display the hierarchy of activities of a Gantt data model.
ResourceTable	A table to display the hierarchy of resources of a Gantt data model.
ReservationTable	A table to display the reservations of a Gantt data model.

The **ActivityTable**, **ResourceTable**, and **ReservationTable** classes share the same base class, the **GanttTable** class. The **GanttTable** class implements the functionalities that are common to the activity and resource tables, such as the connection to the Gantt data model.

The Activity Table

In an activity table each row of the table represents an activity and each column of the table represents a property of the activity in the row. The activity table is implemented by the class **ActivityTable**.

The following illustration shows an activity table.



Name	Duration	StartTime
Conceptual	110 days 21h	1/1/2004 8:00:00 AM
Conceptual Phase...	12 days 21h	4/8/2004 8:00:00 AM
Conceptual pha...	0	4/21/2004 5:00:00 AM
Evaluate busine...	5 days 21h	4/15/2004 8:00:00 AM
Prepare concept...	5 days 21h	4/8/2004 8:00:00 AM
Discipline Support	57 days 21h	2/9/2004 8:00:00 AM
Complete conce...	19 days 21h	3/18/2004 8:00:00 AM
Complete discip...	19 days 21h	3/18/2004 8:00:00 AM
Complete flow s...	19 days 21h	3/18/2004 8:00:00 AM
Create deliverab...	6 days 21h	2/9/2004 8:00:00 AM
Start conceptual...	13 days 21h	2/17/2004 8:00:00 AM
Start discipline...	13 days 21h	2/17/2004 8:00:00 AM
Start flow sheet...	5 days 21h	2/17/2004 8:00:00 AM
Planning and Contr...	26 days 21h	1/1/2004 8:00:00 AM
Business plan id...	5 days 21h	1/1/2004 8:00:00 AM
Define project o...	5 days 21h	1/8/2004 8:00:00 AM
Develop approp...	5 days 21h	1/22/2004 8:00:00 AM
Develop manag...	5 days 21h	1/22/2004 8:00:00 AM

The Resource Table

In a resource table each row of the table represents a resource and each column of the table represents a property of the resource in the row. The resource table is implemented by the class ResourceTable.

The following illustration shows a resource table.

	Name	MaxUnits	Note
▶	Designer	100 %	
	Project engineer	100 %	
	Discipline engineer	100 %	
☐	Cost engineer	100 %	
	David	100 %	
	Engineer 1	100 %	
	Engineer 2	100 %	
☐	Drafter	100 %	
	Robert	100 %	
	Drafter 1	100 %	
	Drafter 2	100 %	
☐	Scheduler	100 %	
	John	50 %	
☐	Project sponsor	100 %	
	Michael	100 %	
	Sponsor	100 %	

The Reservation Table

In a reservation table each row of the table represents a reservation and each column of the table represents a property of the reservation in the row. The reservation table is implemented by the class ReservationTable.

The following illustration shows a reservation table.

	Activity	Resource	Units
▶	Business plan identifyin...	Project sponsor	1
	Define project objective...	Project sponsor	1
	Identify industry standar...	Project sponsor	1
	Identify industry standar...	Project engineer	1
	Develop preliminary con...	Project engineer	1
	Develop appropriation s...	Project sponsor	1
	Develop management...	Project sponsor	1

Connecting a Gantt Table to a Gantt Data Model

To display information, a Gantt table control must be connected to a Gantt data model, which is an instance of the `IGanttModel` interface.

The Gantt data model associated with a Gantt table control can be set using the `GanttModel` property of the `GanttTable` class.

When the Gantt table control is connected to a data model, the Gantt table control listens to events from the Gantt model and is updated with each modification of the data model. For example, when an activity is added or removed from the data model, a corresponding row is added or removed in the `ActivityTable` control.

When a Gantt table is embedded in a Gantt Chart, a Schedule Chart, or a Reservation Chart control, the connection to the data model is automatically handled. You do not need to connect the tables yourself.

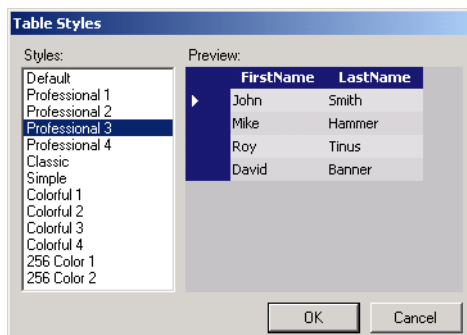
See *Creating and Using Gantt Data Models* for more information on the Gantt data model.

Modifying the Appearance of a Gantt Table

The Gantt table defines several properties for changing its appearance.

Inside Visual Studio .NET you can choose between predefined styles for the appearance of the Gantt table using the Auto Format functionality. Once you have dragged a Gantt table into the Visual Designer, you can right-click the table and choose Auto Format from the menu. In the dialog box, you can choose from several predefined color schemes.

The following illustration shows the auto format dialog box.



The following tables describe the appearance properties of a Gantt table.

Color Properties

Property	Description
BackColor	The color used for the background of rows in the table.
AlternatingBackColor	The color used for alternating rows in the table. By default this color is the same as the BackColor property.
GroupBackColor	The color used for the background of rows representing groups. By default this color is the same as the BackColor property.
ForeColor	The color of text in the table.
BackgroundColor	The color used for the area of the table that is not made up of rows.
HeaderBackColor	The background color of the column and row headers of the table.
HeaderForeColor	The color of text in the column and row headers of the table.
SelectionBackColor	The background color of selected rows in the table.
SelectionForeColor	The color of text for selected rows in the table.
GridLineColor	The color of grid lines in the table.

Font Properties

Property	Description
Font	The font used for text in the Gantt table cells.
HeaderFont	The font used in the column headers of the table.

Grids

Property	Description
GridLineStyle	The style of grid lines that separate the rows and the columns.
GridLineColor	The color of grid lines.

Layout Properties

Property	Description
ColumnHeadersVisible	Indicates whether column headers are visible.
ColumnHeaderHeight	The height of column headers.
RowHeadersVisible	Indicates whether the row headers are visible.
AutoColumnHeaderSize	Indicates whether column headers height are computed automatically using the size of the header font.
RowHeaderWidth	The height of row headers.

Miscellaneous Appearance Properties

Property	Description
BorderStyle	The style of the border of the control.
BackgroundImage	An image that is displayed in the background of the table.
FlatStyle	The flatness style of the control.

Appearance of Table Columns

The appearance of each column of the table can also be customized. See *Modifying the Appearance of Table Columns* for more details.

Right-to-left Mode

All the controls can be used in right-to-left mode for Arabic and other languages that are written right-to-left. Note that bidirectional features of Windows® are only available in a bidirectional Microsoft® Windows® environment, such as an Arabic version of Microsoft® Windows®.

Managing Columns of a Gantt Table

The activity and resource tables are very similar, even though they do not display the same type of information. They both display hierarchical information: the hierarchy of resources or the hierarchy of activities. Thus, the way to customize these tables is the same for an activity table or a resource table.

Each column of the table displays a property of the activity or resource in the row. The columns of a `GanttTable` instance can be accessed through the `Columns` property. This property holds the collection of columns displayed by the table. The class that describes a column is the `TableColumn` class.

To add or remove a column, add or remove an instance of the **TableColumn** class in the collection of columns.

To indicate which property of the activity or resource you want to display in the column, pass the name of the property in the constructor of the column.

For example, the following C# code adds a column that displays the start time of an activity:

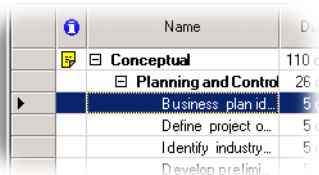
```
table.Columns.Add(new TableColumn("Start Time", "StartTime"));
```

The first parameter is the text that will appear in the header.

The second parameter is the name of the property to display.



To show the hierarchy of objects in a column, use the subclass of **TableColumn** named `TreeColumn`. The **TreeColumn** class shows the hierarchy of objects by displaying information like a tree control with (+) and (-) signs for expanding and collapsing the hierarchy.

The following illustration shows a tree column.






The InfoColumn class is a special column that displays several images in each cell. The images displayed depend on a certain state of the activity or resource in the row.

When this column is added to a resource table, the column shows the following images:

Image	Description
	Indicates that the resource has a note.
	Indicates that the resource does not have reservations.

When this column is added to an activity table, the column shows the following images:

Image	Description
	Indicates that the activity has a note.
	Indicates that a constraint is violated.
	Indicates that the activity has no resource attached to it.

Modifying the Appearance of Table Columns

In a table column, you can specify the following appearance properties:

Property	Description
HeaderText	The text that appears in the column header.
HeaderImage	An image that appears in the column header.
Width	The width of the column.
TextAlign	The horizontal alignment of the text in a cell.
VerticalTextAlignment	The vertical alignment of the text in a cell.
Format	The format used to format the value in a cell.
NullText	The text displayed when the value of the displayed property is null (Nothing in Visual Basic®).

In addition, the `TreeColumn` class provides additional appearance properties:

Property	Description
Indent	The number of pixels used to indent a parent and a child.
ShowPlusMinus	Indicates whether (+) and (-) signs should be displayed to indicate whether a row is expanded or collapsed.
CollapsedImage	Image used when a row is collapsed.
ExpandedImage	Image used when a row is expanded.
LeafImage	Image used when a row is a leaf.
LineStyle	The style of lines linking parent and child nodes.

Editing Values in the Table

A column displays the value of a property of an activity, a resource or a reservation. Therefore, the content of the column can be edited if the property displayed is not read only. The `ReadOnly` property of the column indicates whether the column is editable or not. Even if the property is not typically a read-only property, the column can become noneditable by setting the **ReadOnly** property to **true**.

When the displayed property is not read only, a default editor is assigned to the column, so that the column can be edited. The default editor depends on the type of the property that is displayed. For example, a column displaying dates (instances of the **DateTime** class) is edited with a drop-down calendar. A column that displays text is edited with a basic text editor. A column that displays an **enum** is edited through a drop-down combo box that shows enumerated values.



The editing is done using the **UITypeEditor** and **TypeConverter** associated with the property through the same mechanism used inside Visual Studio .NET to edit the property of

an object. Thus, an easy way to customize the editing of a column is to associate a **UITypeEditor** with the property through the **Editor** attribute on the property.

This default behavior for editing values in the table is implemented by the class `DefaultTreeTableCellEditor`.

Although the default editing behavior covers almost all common needs, you may need to create a custom editor that does not use the concept of **UITypeEditor**. A **UITypeEditor** can have two forms only: a drop-down editor or a dialog box. For example, if you want to edit the values of a column by showing a slider in the cell, you cannot do this with a **UITypeEditor**.

If you want to define the control to be used to edit the values of a column, you must implement the `ITreeTableCellEditor` interface and associate the editor you create with the column through the `CellEditor` property of the `TableColumn` class.

The following methods are used to control the editing in a table:

Method	Description
<code>EditCell</code>	To start editing a table cell by code.
<code>IsEditing</code>	To find out whether the table is currently editing a cell. The <code>EditingColumn</code> and <code>EditingRow</code> properties give you the row and column index of the currently edited cell.
<code>ValidateAndStopEditing</code>	To stop the current editing and accept the current edited values.
<code>StopEditing</code>	To cancel the current editing.

The following C# code extract shows you how to use `ITreeTableCellEditor` to create a custom editor. This code is from the Table Editors sample in:

`<install-dir>/Samples/QuickStart/TableEditors`

where `<install-dir>` is the directory in which you installed IBM ILOG Gantt for .NET.

```
/// <summary>
/// A cell editor that allows editing of the values in the cells of a TreeTable
/// using a Windows Forms TrackBar control.
/// </summary>
public class TrackBarCellEditor : System.Windows.Forms.TrackBar,
ITreeTableCellEditor
{
    /// <summary>
    /// Event fired when the editor stops the editing without validating.
    /// </summary>
    public event EventHandler EditingStopped;

    /// <summary>
    /// Event fired when the editor validates the editing without stopping the
    /// editing.

```

```

/// </summary>
public event EventHandler EditingValidated;

/// <summary>
/// Creates and initializes the editor.
/// </summary>
public TrackBarCellEditor()
{
    // Hides the ticks.
    this.TickStyle = TickStyle.None;
    // Removes the autosize so that the trackbar can take the height of
    // the cell.
    this.AutoSize = false;
}

/// <summary>
/// Returns the TrackBar control to edit the value.
/// </summary>
/// <param name="table">The <see cref="TreeTable"/> currently edited.
/// </param>
/// <param name="value">The current value in the edited cell.</param>
/// <param name="isSelected">Indicates whether the edited cell is
/// selected.</param>
/// <param name="row">The zero-based row index of the edited cell.</param>
/// <param name="column">The zero-based column index of the edited cell.
/// </param>
public Control GetEditorControl(
    TreeTable table,
    object value,
    bool isSelected,
    int row,
    int column)
{
    Value = value;
    return this;
}

/// <summary>
/// Gets or sets the value of the editor.
/// </summary>
public new object Value
{
    get
    {
        return base.Value;
    }
    set
    {
        base.Value = (int)value;
    }
}

/// <summary>
/// Tells the editor to stop the editing without validating.
/// </summary>
public void StopEditing()
{
    if (EditingStopped != null)

```

```

        EditingStopped(this, EventArgs.Empty);
    }

    /// <summary>
    /// Tells the editor to validate the editing.
    /// </summary>
    public void ValidateEditing()
    {
        if (EditingValidated != null)
            EditingValidated(this, EventArgs.Empty);
    }
}

```

Default Columns for Activity, Resource, and Reservation Tables

When you create a `GanttChart` instance or an `ActivityTable` instance, the table contains the following set of columns:

Column	Description
Information column	An instance of the <code>InfoColumn</code> class that displays images that show some information on the activity.
Name column	A tree column that shows the name of an activity (<code>Name</code> property of <code>IActivity</code>).
Duration column	A column that shows the duration of the activity (<code>Duration</code> property of <code>IActivity</code>).
Start time column	A column that shows the start time of the activity (<code>StartTime</code> property of <code>IActivity</code>).
End time column	A column that shows the end time of the activity (<code>EndTime</code> property of <code>IActivity</code>).
Resources column	A read-only column that shows the resources assigned to the activity (Resources property of <code>IActivity</code>).

When you create a `ScheduleChart` instance or a `ResourceTable` instance, the control already contains the following set of columns:

Column	Description
Information column	An instance of the <code>InfoColumn</code> class that displays images that show some information on the resource.
Name column	A tree column that shows the name of the resource (<code>Name</code> property of <code>IResource</code>).

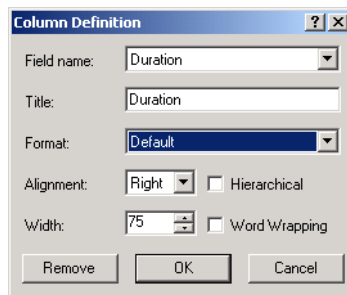
When you create a ReservationChart instance or a ReservationTable instance, the control already contains the following set of columns:

Column	Description
Activity column	A table column that shows the activity of the reservation (Activity property of IReservation).
Resource column	A table column that shows the resource of the reservation (Resource property of IReservation).
Units column	A table column that shows the units of the reservation (Units property of IReservation).

Dialog Box for Editing the Columns of a Gantt Table

IBM® ILOG® Gantt for .NET provides a predefined dialog box for editing the columns of a Gantt table. This dialog box is used to edit the properties of a column and to remove columns. It can also be used to add new columns to a Gantt table.

The following illustration shows the table column dialog box.



This dialog box is implemented by the class GanttTableColumnDialog in the ILOG.Views.Gantt.Windows.Forms namespace. To make use of the dialog box, you specify the column to edit and the type of object that is displayed in a row of the table. With these two items of information, the dialog box can show information on the specified column and also suggest different properties for viewing in the column using introspection to list the properties of the object displayed in a row of the table. For example, when the table is an activity table, each row of the table displays an activity. The dialog box will list the properties of the activity class in the Field Name combo box.

Here is a C# code fragment that shows how to edit the property of a column of an ActivityTable:


```

void EditActivityColumn(TableColumn column)
{
    dialog dialog = new GanttTableColumnDialog();
    dialog.TableRowType = typeof(SimpleActivity);
    dialog.TableColumn = column;
    dialog.ShowDialog();
}

```

Here is another C# code fragment that shows how to add a new column to a ResourceTable through the dialog box:

```

void NewTableColumn(ResourceTable table)
{
    GanttTableColumnDialog dialog = new GanttTableColumnDialog();
    dialog.TableRowType = typeof(SimpleResource);
    dialog.TableColumn = new TableColumn();
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        table.Columns.Add(dialog.TableColumn);
        table.EnsureColumnVisible(table.Columns.Count - 1);
    }
}

```

Using Predefined Behavior to Manipulate Rows and Columns

In addition to the behavior demonstrated for editing the values in a cell, the table shows the following behavior in the following contexts:

Resizing a Row

A row can be resized by clicking between two rows in the row headers and dragging the mouse pointer. You can disable this behavior by setting the `CanResizeRows` property to **false**.

Resizing a Column

A column can be resized by clicking between two columns in the column headers and dragging the mouse pointer. You can disable this behavior by setting the `CanResizeColumns` property to **false**.

Setting the Preferred Height of a Row

The height of a row can be set to the preferred height by double-clicking between two rows in the row headers. You can disable this behavior by setting the `CanResizeRows` property to **false**.

Setting the Preferred Width of a Column

The width of a column can be set to the preferred width by double-clicking between two columns in the column headers. You can disable this behavior by setting the `CanResizeColumns` property to **false**.

Changing the Order of Columns

The order of columns can be changed by clicking the column header and dragging the column to a new location. You can disable this behavior by setting the `CanMoveColumns` property to **false**.

Keyboard Mapping

Key	Behavior
F2	Starts editing the current cell.
ESC	Cancels the current edits.
ANY CHARACTER	Starts editing the current cell.
RIGHT ARROW	Moves the current cell to the right, unless the current cell is a cell of a tree column that can be expanded. In this case, it expands the row.
LEFT ARROW	Moves the current cell to the left, unless the current cell is a cell of a tree column that can be collapsed. In this case, it collapses the row.
UP/DOWN ARROW	Selects the row above/below the current row.
SHIFT+UP/DOWN ARROW	Extends the selection to the row above/below the current row.
CTRL+UP/DOWN ARROW	Moves the current row up/down without changing selection.
CTRL+SHIFT+UP/DOWN ARROW	Adds the row above/below the current row to the selection.
HOME	Selects the first row.
SHIFT+HOME	Extends the selection from the selection anchor to the first row.
CTRL+HOME	Moves the current row to the first row without changing selection.

Key	Behavior
CTRL+SHIFT+HOME	Adds the rows from the current row to the first row to the selection.
END	Selects the last row.
SHIFT+END	Extends the selection from the selection anchor to the end.
CTRL+END	Moves the current row to the last row without changing selection.
CTRL+SHIFT+END	Adds the rows from the current row to the last row to the selection.
PAGEUP/PAGEDOWN	Selects the row one page above/below the current row.
SHIFT+PAGEUP/ PAGEDOWN	Extends the selection one page up/down.
CTRL+PAGEUP/ PAGEDOWN	Moves the current row one page up/down without changing the selection.
CTRL+SHIFT+PAGEUP/ DOWN	Adds one page up/down from the current row to the selection.
*	Expands the entire subtree, starting from the current cell.

Expanding or Collapsing Rows

The `ActivityTable` class displays the hierarchical structure of activities, such that each row represents one activity. Similarly, the `ResourceTable` class displays the hierarchical tree of resources, such that each row represents a resource. In both tables, rows are numbered from index 0. Row numbering ignores vertical scrolling and is not affected by the current vertical scrolling position of the table.

The classes **ActivityTable** and **ResourceTable** have in common methods and properties for controlling the expanded or collapsed status of a row.

Member	Description
RowCount	To get the number of rows.
IsRowExpanded	To indicate whether the row is expanded.
ExpandRow	To expand a row.
CollapseRow	To collapse a row.

ExpandAll	To expand a row and all its children recursively.
CollapseAll	To collapse a row and all its children recursively.
ShowOutlineLevel	To make the nodes with the specified outline visible and to hide the lower levels.
IsRowVisible	To indicate whether a row is visible.

Scrolling the Gantt Table

The table contains a vertical and a horizontal scroll bar that allow scrolling in the table. By default, these scroll bars are visible only when they are needed. You can use the `VScrollBar` and `HScrollBar` properties to change this behavior.

The `ScrollBarVisibility` enumeration has three values that define the supported scroll bar policies:

Enumeration Member	Description
ScrollBarVisibility.AsNeeded	The scroll bar is visible only when it is needed.
ScrollBarVisibility.Visible	The scroll bar is always visible.
ScrollBarVisibility.Hidden	The scroll bar is hidden.

For example, if you want the vertical scroll bar to be always visible, you can write:

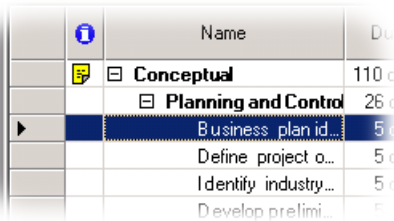
```
myTable.VScrollBar = ScrollBarVisibility.Visible
```

Here are the methods and properties you can use to scroll the table:

Member	Description
FirstVisibleRow	To get or set the index of the first visible row in the table.
HorizontalOffset	To get or set the horizontal scrolling offset of the table.
EnsureColumnVisible	To make sure that a column is visible.
EnsureRowVisible	To make sure that a row is visible.
EnsureVisible	To make sure that a cell is visible.

Getting and Setting the Current Cell

The table defines a notion of current cell, the current cell is indicated on the screen with the focus rectangle when the table has the keyboard focus. The row that contains the current cell is marked with an arrow in the cell headers.



Here are the methods and properties that you can use to obtain and change the current cell:

Member	Description
CurrentRow	To get or set the current row of the table.
CurrentColumn	To get or set the current column of the table.
SetCurrentCell	To set the current cell of the table.

Here is a small C# example that shows how to set the current cell:

```
private void SetCurrentCellAt(GanttTable myTable)
{
    // Set the current cell to column 0, row 0.
    myTable.SetCurrentCell(0,0);
}
```

Controlling Selection in the Table

The Gantt table can have multiple rows selected. The selection can be made with the mouse or the keyboard. You can use the following methods to control selection in the Gantt table:

Method	Description
IsSelected	To indicate whether a row is selected.
SetSelected	To select or deselect a row.
GetSelection	To return an array of the indexes of the table that are selected.

IsSelectionEmpty	To indicate whether the selection is empty.
ClearSelection	To deselect all the selected rows.
SetSelectionInterval	To set the selection to a range.
AddSelectionInterval	To add a range to the selection.
RemoveSelectionInterval	To remove the specified interval from the selection.

You can use the `GetSelectedActivities` method of the `ActivityTable` class to return an array of selected activities. Here is a small C# example that iterates on selected activities in an activity table:

```
ActivityTable table = ...
foreach (IActivity activity in table.GetSelectedActivities())
{
    // do something on the activity
}
```

You can use the `GetSelectedResources` method of the `ResourceTable` class to return an array of selected resources. Here is a small C# example that iterates on selected resources in a resource table:

```
ResourceTable table = ...
foreach (IResource resource in table.GetSelectedResources())
{
    // do something on the resource
}
```

When the selection changes in the table, the `SelectionChanged` event is raised. The event handler for this event is:

```
public delegate void TableSelectionChangedEventHandler(object sender,
TableSelectionChangedEventArgs args);
```

When a multiple selection is made by dragging the mouse pointer in the table, then several `SelectionChanged` events are raised. The event argument passed to **SelectionChanged** tells you whether the event is part of a series of selection events. (See `TableSelectionChangedEventArgs.IsAdjusting`.)

Hit Testing in the Gantt Table

If you need to create custom behavior in a Gantt table, you will need to get information on the Gantt table at a specific location on the screen. You can do this with the `HitTest` method

of the Gantt table class. Calling this method returns an instance of the `HitTestInfo` structure that gives the following information:

Property	Description
Row	The Row under the specified point or -1 .
Column	The Column under the specified point or -1 .
Type	One of the HitTestType values. Each value is explained in the next table.

This table explains the different values of the `HitTestType` enumeration:

HitTestType Value	Location of the Point
None	An area of the table that does not contain rows was clicked.
Cell	A cell.
ColumnHeader	A column header.
RowHeader	A row header.
ColumnResize	The area between two columns in the column headers.
RowResize	The area between two rows in the row headers.

Here is a small C# example that shows how to use hit testing on a Gantt table:

```
private void OnActivityTableMouseDown(object sender, MouseEventArgs e)
{
    ActivityTable table = (ActivityTable)sender;
    if (e.Clicks == 2)
    {
        TreeTable.HitTestInfo info = table.HitTest(new Point(e.X, e.Y));
        if (info.Type == TreeTable.HitTestType.ColumnHeader)
            Console.WriteLine("Double click on a column header");
        else if (info.Type == TreeTable.HitTestType.RowHeader)
            Console.WriteLine("Double click on a row header");
    }
}
```

Grouping, Filtering and Sorting Rows

The Gantt table provides capabilities for grouping, filtering and sorting data. The grouping allows you to create a hierarchy in the rows of the table. The filtering allows you to show only specific rows of the table. The sorting allows you to sort the rows of the table depending on a specific criteria.

Grouping Rows

Using the `SetRowGroup` method of the `GanttTable`, you can group rows based on their column value. The `IRowGroup` interface is used to define a hierarchy of nodes into a `TreeModelView`.

The following C# code sample shows how to group reservations having the same resource property in a `ReservationTable`:

```
ReservationTable table = new ReservationTable();
table.SetRowGroup = new DefaultRowGroup("Resource");
```

Note: *Once the group has been set, the table will update groups as the model gets modified.*

You can create your own group by implementing the **`IRowGroup`** interface.

Grouping is implemented by the **`TreeModelView`** class, a wrapper for a tree model (see `ITreeModel`). For details about filtering, see the `TreeModelView` class.

Filtering Rows

Using the `RowFilter` property of the `GanttTable`, you can specify subsets of rows based on their column values. The **`RowFilter`** property is an expression that is evaluated for each row of the table to check whether the row should be displayed by the table.

The following C# code sample shows how to display only milestone activities scheduled after the 1/1/2000 in an `ActivityTable`:

```
ActivityTable table = new ActivityTable();
table.RowFilter = "Milestone && StartTime > #1/1/2000#";
```

Note that once the filter has been set, the table will stop displaying rows that do not satisfy the filter. Now if the Gantt Data Model is modified, only rows satisfying the filter criteria will be displayed by the table.

If your filter criteria cannot be expressed using an expression, you can create your own filter by implementing the `IRowFilter` interface, and use the `SetRowFilter` method of the **`GanttTable`** class to apply the filter:

```
public class MyRowFilter : IRowFilter
{
    ...
}

ActivityTable table = new ActivityTable();
IRowFilter filter = new MyRowFilter();
table.SetRowFilter(filter);
```


Filtering is implemented by the **TreeModelView** class, a wrapper for a tree model (see `ITreeModel`). For details about filtering, see the `TreeModelView` class.

Sorting Rows

The `SortRows` methods of the `GanttTable` class provide means to sort the rows displayed by the table without modifying the Gantt Data Model being displayed. Unlike the filtering feature, which is set once and then applied until unset, sorting is done by calling the **SortRows** methods. The following C# code shows how to sort the activities displayed by an `ActivityTable` according to their **StartTime** property:

```
ActivityTable table = new ActivityTable();
table.GanttModel = model; // Assume the model contains data.
table.SortRows("StartTime");
```

Resetting Group, Filter and Sort Order

Once the table has been grouped, filtered or sorted, to remove the group or filter and restore the original order of the Gantt Data Model entities, use the `ResetView` method.

Displaying the Load of a Resource

IBM® ILOG® Gantt for .NET provides a control for displaying the load of a resource over time. This control is called the load chart control.

In This Section

Introducing the LoadChart Class

Describes the LoadChart class.

Connecting the Load Chart to a Resource

Describes how to specify the resource to display.

Modifying the Appearance of a Load Chart

Describes the properties for customizing the appearance of a load chart.

Controlling the Displayed Time Interval

Describes how to control the way the time period is displayed.

Related Sections

Displaying Time-based Information

Describes the TimeControl class.

Using Time Grids and Date Indicators

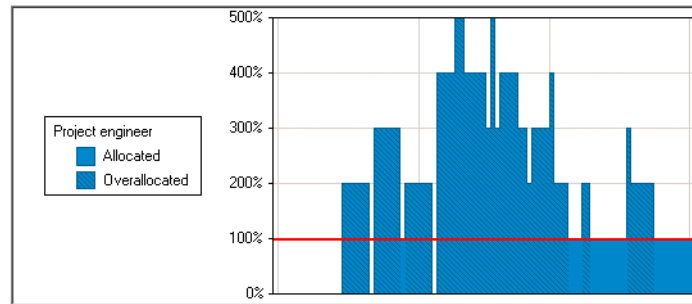
Explains how time grids and date indicators are displayed by the Gantt controls.

Introducing the LoadChart Class

The LoadChart class is a control that displays the load of a resource over time.

The control displays an area chart where the x-axis represents time and the y-axis represents the load of the resource.

The following illustration shows a load chart.



The load of a resource over a period of time is calculated by computing the sum of work that the activity does during the period.

The chart collects all the reservations of the resource during the period of time. Then, calculates the sum of the `Units` property of the reservations. The **Units** property represents how much of the resource the reservation is using. If the result is greater than the `IResource.MaxUnit` of the resource, then the resource is over-allocated.

Connecting the Load Chart to a Resource

In order to display information on the load of a resource, the Load Chart control must be connected to a resource (instance of `IResource`) through its `Resource` property.

Once the Load Chart control is connected to a resource, the Load Chart listens to events in the Gantt Data model and is updated when a resource changes or when a reservation changes, is added, or is removed from the resource.

Modifying the Appearance of a Load Chart

The LoadChart class defines several properties for changing its appearance.

Color Properties

Property	Description
BackColor	The color used for the background of the chart.
ForeColor	The color of text in the legend.
BackgroundColor	The color used for the background of the control.
ChartColor	The foreground color of the chart.

Horizontal Grid Lines

The horizontal grid lines of the Load Chart are controlled by the following properties:

Property	Description
GridLineColor	The color of horizontal grid lines.
GridLineStyle	The style of horizontal grid lines.

Vertical Time Grid

The Load chart can display vertical grid lines or areas that separate time periods on the time scale or display nonworking time with a specific appearance.

The Load chart can also display vertical grid lines that indicate a particular date of your project, for example, the current date.

For more information see *Using Time Grids and Date Indicators*.

Miscellaneous Appearance Properties

Property	Description
BorderStyle	The style of the border of the control.
IsMaterialResource	Indicates whether the resource should be considered as equipment. In this case, the vertical scale shows number instead of percentage.

AutoScaleUpdate	Indicates whether the chart should automatically adapt its vertical scale to the maximum load in the visible period.
MaxScaleValue	The maximum value of the scale. This value is only used when the AutoScaleUpdate property is false .
ChartPosition	The position of the chart in the control.

Right-to-left Mode

All the controls can be used in right-to-left mode for Arabic and other languages that are written right-to-left. Note that bidirectional features of Windows® are available only in a bidirectional Microsoft® Windows® environment, such as an Arabic version of Microsoft® Windows®.

Controlling the Displayed Time Interval

The LoadChart class defines the following properties for controlling the displayed time interval:

Property	Description
FirstVisibleTime	Gets or sets the first visible time of the Load Chart.
VisibleDuration	Gets or sets the visible duration of the Load Chart.
LastVisibleTime	Gets or sets the last visible time of the Load Chart.
VisibleTimeInterval	Gets or sets the time interval displayed by the Load Chart in one single operation.

The LoadChart class inherits from the TimeControl class that is the base class for controls that display information based on time. The **TimeControl** class defines the first visible time of the control, as well as the visible duration of time in the control. For more information on this class and to learn how to synchronize several controls that display time, see *Displaying Time-based Information*.

Displaying Activities Using a Calendar View

IBM® ILOG® Gantt for .NET provides a control for displaying activities in a calendar. This control is called the Calendar View control.

In This Section

Introducing the CalendarView class

Describes the CalendarView class.

Displaying Activities in the Calendar View.

Describes how to connect a Calendar View to visualize data.

Modifying the Appearance of the Calendar View

Describes the properties that control the appearance of the Calendar View.

Representing Activity bars in the Calendar View

Describes how to control the appearance of bars in the Calendar View.

Controlling the Layout of Activities in the Calendar View

Describes how to control the automatic layout of bars in the Calendar View.

Using the Predefined Behavior of the Calendar View

Describes the predefined behavior in the Calendar View.

Displaying Activities Using a Calendar View

Hit Testing in the Calendar View

Describes the Hit testing API of the Calendar View.

Related Sections

Displaying Scheduling Data Using Gantt Charts

Describes how to display scheduling data using the main controls of IBM ILOG Gantt for .NET.

Displaying Scheduling Data Using Tables

Describes the Activity Table and Resource Table controls for displaying scheduling information.

Displaying the Load of a Resource

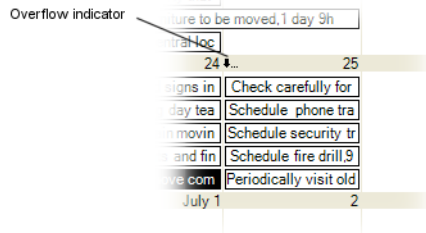
Describes how to use the Load Chart control, a control for displaying the load of a resource.

Introducing the CalendarView class

The calendar view control represents the activities of a Gantt Model in a calendar. Each row of the calendar represents one week. Activities displayed by the Calendar View are represented by a bar. The bars are layouted so that there is no overlap and the vertical order represents the order of the activities in the Gantt model. The control defines many appearance properties that will allow you to control the representation and the order of the bars.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
January 18	19	20	21	22	23	24
Evaluate internal product,6 days 9h				Identify production process steps required for		
25	26	27	28	29	30	31
Identify production process steps required for manufacture,6				Assess manufacturing capabilities,13 days 9h		
				Determine safety issues,13 days 9h		
				Determine environmental issues,13 days 9h		
				Review legal issues,13 days 9h		
February 1	2	3	4	5	6	7
Assess manufacturing capabilities,13 days 9h						
Determine safety issues,13 days 9h						

When some activities cannot be displayed in a cell of the calendar, then the Calendar view will display an overflow indicator as in the picture below:



Displaying Activities in the Calendar View

To display activities, the calendar view must be connected to a Gantt data model. You connect the Calendar View to the Gantt model using the GanttModel property.

When the Calendar View control is connected to a Gantt data model, the Calendar View control listens to events from the Gantt data model and is updated for each modification of the data model. For example, when an activity is added to or removed from the data model, a corresponding bar is added to or removed from the calendar and activity bars will be laid out so that there is no overlap on the calendar.

You might want to display only activities assigned to a specific resource. In this case you can set the Resource property of the Calendar view.

Modifying the Appearance of the Calendar View

The Calendar View defines several properties for changing its appearance.

Colors

Property	Description
ForeColor	Foreground color used for text in cell headers.
BackColor	Background color of cells.
AlternatingBackColor	Second background color used for alternate month.
BackgroundColor	Color of non calendar area of the control.
HeaderBackColor	Background color of column headers.

HeaderForeColor	Foreground color of column headers.
CellHeaderColor	Background color of cell headers.

Grid lines

Property	Description
DayGridLineColor	Color of vertical grid lines between two days.
WeekGridLineColor	Color of horizontal grid lines between two weeks.
DayGridLineStyle	Style of vertical grid lines between two days.
WeekGridLineStyle	Style of horizontal grid lines between two weeks.

Font Properties

Property	Description
Font	Font used for bars and cell headers.
HeaderFont	Font of column headers.

Miscellaneous Appearance Properties

Property	Description
BorderStyle	The style of the border of the control.
FlatStyle	The flat style appearance of the control.
AutoFit	When set to true , ensures that all columns are always visible.
CellHeaderDateFormat	A string representing the format of the date displayed in the cell headers.
CompactWeekEnd	When set to true the Saturday and Sunday are displayed in one column.

Appearance of activity bars

The appearance of the rectangular bars that represent activities in the Calendar View can be completely customized by a styling mechanism. This styling mechanism is fully described in *Representing Activity bars in the Calendar View*.

Right-to-left Mode

All the controls can be used in right-to-left mode for Arabic and other languages that are written right-to-left. Note that bidirectional features of Windows® are available only in a bidirectional Microsoft® Windows® environment, such as an Arabic version of Microsoft® Windows®.

Representing Activity bars in the Calendar View

By default the Calendar View displays all the activities in a Gantt model except summary activities.

You can control which activities will be displayed as well as how they will be displayed using the `BarStyles` property. The `BarStyles` property of the `CalendarView` class is a collection of `CalendarViewBarStyle` class. Each instance of `CalendarViewBarStyle` represents the style of bars for a particular type of activity. By default the collection contains two bar styles: One for "Normal" activities (activities that are not milestone and not summary) and one for "Milestone" activities.

Each instance of `CalendarViewBarStyle` defines the appearance of the bar displayed in the calendar and defines also the type of activity for which this style should be used.

The appearance properties of a bar defined in the `CalendarViewBarStyle` class are the following:

Property	Description
Color	The background color of the bar.
BorderColor	The color of the border.
FillStyle	The style for filling the bar (solid color or hatch)
HatchStyle	The hatch style for the background of the bar
SecondColor	Another color defining the second color when the bar is hatched.

DisplayedText	An expression that defines the text displayed in the bar.
BarRounding	Defines whether the bar start and end time should be rounded to the beginning or end of a day.

Finally the **CalendarViewBarStyle** defines the **StyleFor** property. This property is an expression that will be evaluated to decide if this style can be used for a particular activity. See *Expression Language Reference* for details.

When the **CalendarView** needs to display an activity, he will look inside the bar style collection and evaluate the **StyleFor** property. If one style matches the activity, this style will be used to draw the activity, otherwise the activity is not displayed.

Controlling the Layout of Activities in the Calendar View

The calendar view layouts automatically the activity bars so that they do not overlap. By default the activity bar appear in the same order as in the Gantt data model. Every time the data model is modified, for example when an activity is added, removed or when the start time or end time changes, the calendar view will redo the layout of the bars so that the displayed order always reflect the data model content and there is no overlap of bars.

The **CalendarView** defines several properties for controlling the layout of activities.

Sorting Activities

The **SortBy** property can be used to change the sorting order of the bars. For example if you set the **SortBy** property to "Name", then the bars will be sorted by the "Name" property. You can also reverse the sorting order using the **AscendingSort** property.

Using Compact Layout

The **CalendarView** also defines the **CompactLayout** property. When set to **true**, the **CalendarView** will try to place activity bar so that more activity bar become visible, but in this case the sorting order will not be respected.

Threading

With very large data model containing ten of thousands of activities, the layout operation can become time consuming, thus the layout is always done in a separate thread. The layout system sends two events (**LayoutStarted** and **LayoutDone** events) to notify when the layout starts and finishes, through these events it is possible to give a feedback to the final user that the calendar is performing a layout operation.

Rounding Bars

Finally the layout is also controlled by the styles used for displaying the bars. The `CalendarViewBarStyle` defines the `BarRounding` property. When this property is **true** for a style, the start time and end time of a bar will be rounded to the beginning and end of a day and thus fewer bars can be displayed in a day. Note that activities with a zero duration will be rounded whatever the value of the **BarRounding** property.

Using the Predefined Behavior of the Calendar View

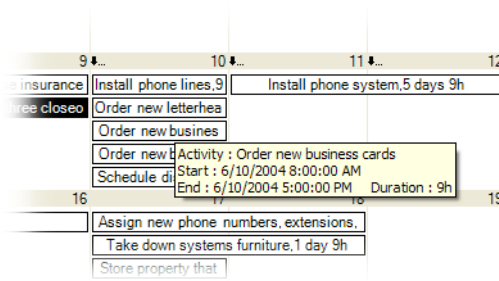
A Calendar View has the following predefined behavior.

Resizing Rows and Columns

In the Calendar View, an application user can resize the cells of the calendar by clicking on the vertical or horizontal grid lines and dragging the mouse pointer. Note that when the `AutoFit` property is set to **true**, the size of columns are automatically adjusted so that all columns are visible, in this case it is not possible to change the column width using the mouse pointer.

Tooltips

Tooltips appear in the Calendar View when the mouse hovers an activity bar. By default, the tooltip displays the name, start time, end time, and duration of the activity.



The text appearing in the tooltip can be modified using the `ActivityToolTip` property.

The value of this property is not a static string, but a string that represents an expression that will be evaluated using the current values of the activity. See *Expression Language Reference* for details.

Displaying Activities Using a Calendar View

You can disable the tooltips displayed in the Calendar View by setting the ShowTooltips property to **false**.

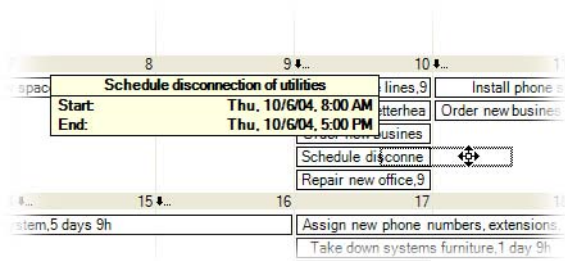
Changing the Start Time, End Time of an Activity

In a Calendar View, an application user can change the time interval of an activity using the mouse pointer:

Action	User Interaction
Change only the end time	Click at the end of the activity bar and drag the mouse pointer.
Change the start time while keeping the duration	Click in the middle of the activity bar and drag the mouse pointer.

You can disable this default behavior by setting the CanEditActivities property to **false**.

When an application user performs one of these operations, a tooltip appears showing the values that are modified:



You can disable this tooltip by setting the ShowEditingToolTip property to **false**.

The CalendarView class provides the BeforeEditActivity event that is raised before the activity is modified. For example, this event allows the application user to cancel the editing operation in some cases or to modify the chosen time interval.

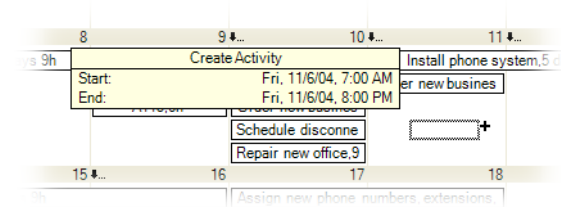
Finally you can control the cursors that will be used during this operation through the following properties:

Property	Description
----------	-------------

ResizeCursor	Cursor used for changing the end time
MoveCursor	Cursor used to indicate that an activity bar can be moved.

Creating an activity using the mouse pointer

In the Calendar View a final user can create an activity with the mouse pointer. To create an activity, simply click in the calendar and drag the mouse pointer.



You can disable this default behavior by setting the `CanCreateActivities` property to **false**.

When the user creates an activity with the mouse pointer a tooltip appears showing the time interval of the new activity. This tooltip can be disabled by setting the `ShowEditingToolTips` property to **false**.

The `CalendarView` class provides the `CreateActivity` event that is raised before the activity is created. You can listen to this event if you want to write the code that creates the activity and add it in the Gantt data model.

Finally you can control the cursor that is used while creating an activity using the `CreateActivityCursorChanged` property.

Hit Testing in the Calendar View

If you need to create custom behavior in a Calendar View, you will need to get information on the Calendar View at a specific location on the screen. You can do this with the `HitTest` method. Calling this method returns an instance of the `HitTestInfo` structure that gives the following information:

Property	Description
Date	The date under the specified point.

Displaying Activities Using a Calendar View

Activity	The activity under the specified point or null (Nothing in Visual Basic).
Type	One of the HitTestType values. The values are explained in the next table.

The following table explains the **HitTestType** values:

HitTestType value	Description
None	The background of the calendar.
Activity	An activity is located at the specified point.
Day	A cell of the calendar
ColumnHeader	The column header
ColumnResize	Vertical grid line for resizing a column
RowResize	Horizontal grid line for resizing a row
DayHeader	The header of a cell

Here is a small C# example that shows an event handler that handles the Mouse-down event of a **CalendarView** instance and checks whether a double-click occurs on an activity:

```
private void OnCalendarMouseDown(object sender, MouseEventArgs e)
{
    CalendarView calendar = (CalendarView)sender;
    if (e.Clicks == 2)
    {
        CalendarView.HitTestInfo info = calendar.HitTest(new Point(e.X, e.Y));
        if (info.Type == CalendarView.HitTestType.Activity)
        {
            // double click occurs on activity
        }
    }
}
```


Customizing the Drawing of Gantt Components

IBM® ILOG® Gantt for .NET provides several control that display scheduling data over time. These controls are sharing classes used to render grids, date indicators, and other drawing components. This section describes the classes and properties available to the programmer to control the drawing of the Gantt components.

In This Section

Using Time Grids and Date Indicators

Describes how to program the vertical grid lines that indicate time periods.

Representing Activity Bars in Gantt Sheets

Describes the styling mechanism for customizing the appearance of activity bars.

Creating Owner-Drawn Gantt Components

Describes how to create owner-drawn Gantt components.

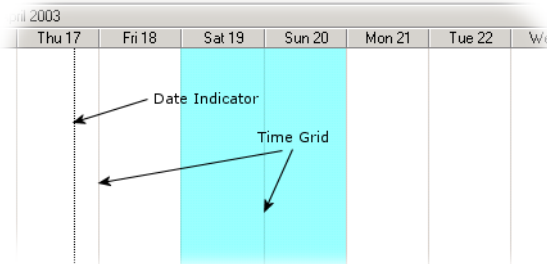
Using Time Grids and Date Indicators

Gantt controls that display information based on time, such as the LoadChart or the GanttSheet controls, can display vertical grid lines to separate time periods or to show

nonworking time such as weekends. They can also display a vertical line to indicate some particular date, such as the current date.

The vertical grid lines that separate time periods and show nonworking time periods are called a *time grid*. The vertical grid lines that indicate a particular date are called a *date indicator*.

The following illustration shows time grids and indicators.



The controls that can display time grids define a property named **TimeGrids**. See, for example, the `GanttSheet.TimeGrids` property or the `LoadChart.TimeGrids` property. The type of this property is `TimeGridCollection`, a collection of objects of the class `TimeGrid`.

TimeGrid Class

The **TimeGrid** class defines the following properties applicable to all time grids:

Property	Description
Visible	Indicates whether the grid should be displayed.
ShowOnTop	Indicates whether the grid should be drawn on top or beneath the content of the control.
UseDataContext	Gets or sets a value that indicates if the grid should be painted using its data context. The data context of a grid depends on the control that displays the grid. When this property is set to true, the grid may be painted in several passes, one per data context. For example, a ScheduleSheet displaying a grid will draw the grid on a per row basis, using the resource of each row as data context.
DataContext	Gets the data context used to paint this grid.

The **TimeGrid** class is the base class for all the time grids. The subclasses of **TimeGrid** are:

- ◆ DefaultTimeGrid

- ◆ WorkingTimesGrid
- ◆ TimeRangeGrid

DefaultTimeGrid

A time grid that displays vertical grid lines spaced by a fixed time interval. The **DefaultTimeGrid** class defines the following properties:

Property	Description
Color	The color of the lines.
DashStyle	The dash style of the lines.
TimeUnit	The duration that separates two lines.
Steps	The number of TimeUnit properties that separate two lines.

WorkingTimesGrid

The **WorkingTimesGrid** displays nonworking times defined by a WorkCalendar. The following properties are available to customize the grid:

Property	Description
Calendar	The calendar used by the grid.
ForeColor	The color used to paint nonworking time periods.
BackColor	The background color of the hatch when the UseHatch property is true .
HatchStyle	The hatch style when the UseHatch property is true .
UseHatch	Indicates whether the nonworking time periods should be hatched.

TimeRangeGrid

The **TimeRangeGrid** fills a time area defined by an ITimeRange object. The following properties are available to customize the grid:

Property	Description
TimeRange	The time range of the grid.
ForeColor	The color used to fill the time range of the grid.

BackColor	The background color of the hatch when the UseHatch property is true .
HatchStyle	The hatch style when the UseHatch property is true .
UseHatch	Indicates whether the time range of the grid should be hatched.
PaintInside	Indicates whether the grid should paint inside its time range.

Date Indicators

The controls that can display date indicators define a property named **DateIndicators**. For example, see the `GanttSheet.DateIndicators` property or the `LoadChart.DateIndicators` property. The type of this property is `DateIndicatorCollection`, a collection of objects of the class `DateIndicator`.

The **DateIndicator** class defines the following properties:

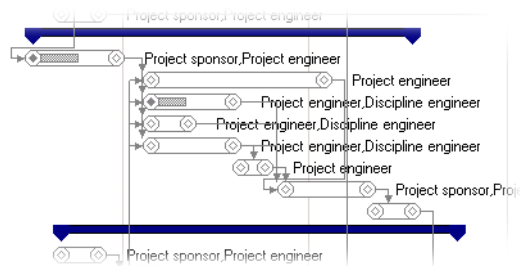
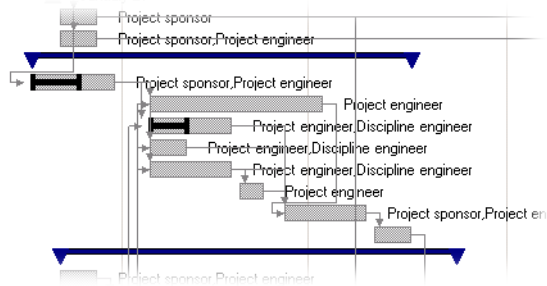
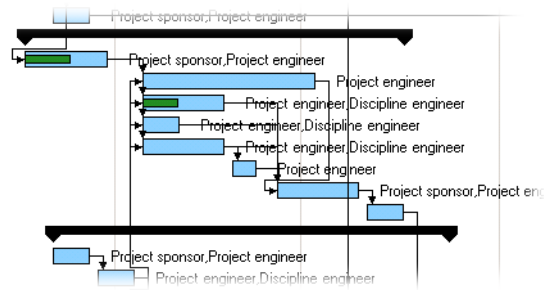
Property	Description
Color	The color of the line.
DashStyle	The dash style of the line.
Time	The time where the indicator is to be drawn.

The `CurrentDateIndicator` class is a subclass of the **DateIndicator** class. The `Time` property in this date indicator is constantly updated to the current time.

Representing Activity Bars in Gantt Sheets

The Gantt Chart, Schedule Chart and Reservation Chart controls display activity bars to represent an activity along a time scale. Although these controls have a predefined way of representing activity bars, you can fully customize the appearance of the bars to fit your needs.

The following illustrations show some representations of activity bars:



In This Section

Styling Activity Bars in Gantt Sheets

Explains the classes involved in styling activity bars.

Dialog Box Control for Styling Activity Bars

Introduces the dialog box for defining activity bar styles inside Visual Studio .NET or at runtime.

Styling Activity Bars in Gantt Sheets

The way activity bars are represented on the screen is controlled by a collection of styles that can be accessed by the `ActivityBarStyles` property of the `GanttSheet` class.

The `GanttChart`, `ScheduleChart`, and `ReservationChart` classes also have an `ActivityBarStyles` property that refers to the `ActivityBarStyles` property of the internal `GanttSheet` class used by these controls.

When the `GanttSheet` class needs to render an activity, it looks inside its collection of styles to find all the styles that are relevant to this activity. Matching styles are then used to render the activity. Applying several styles to render a single activity can be used to give several items of information about the activity. For example, a first style can be used to display a rectangle that shows the duration of the activity; a second style can superimpose a rectangle that shows the percentage completion of the activity.

Important: *The order of the styles in the collection is significant. Since the Gantt Sheet can use several styles to render a single activity, the styles that appear first in the collection are rendered before, thus underneath, the styles that appear later in the collection.*

The collection of styles is defined by the class `ActivityBarStyleCollection` that holds a collection of instances of the class `ActivityBarStyle`. The class `ActivityBarStyle` fully defines the way the bar will be rendered on the screen, as well as the kind of activity to which this style applies.

In This Section

Modifying the Appearance of Activity Bars

Explains the appearance properties of an activity bar.

Defining When a Bar Style Applies

Explains how to control which style applies to which activity.

Interacting and Styling

Explains how to control interactions using styling.

Example of Styling

Provides a full example of styling.

Related Sections

Expression Language Reference

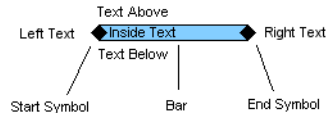
Explains how to define expressions based on the property of an object.

Dialog Box Control for Styling Activity Bars

Describes the dialog box used to style activity bars.

Modifying the Appearance of Activity Bars

The `ActivityBarStyle` class defines many visual attributes used to render an activity on the screen. An activity can be rendered by a rectangle along the time scale, two optional symbols at the beginning and end of this rectangle, and five items of textual information to the right, to the left, above, below and inside the rectangle.



The **ActivityBarStyle** and its associated classes allow you to customize all these visual attributes.

An instance of the **ActivityBarStyle** class contains height other objects:

- ◆ One instance of the `ActivityBar` class that defines the appearance of the rectangular bar.
- ◆ Two instances of the `BarSymbol` class that define the symbols at the beginning and end of the bar.
- ◆ Five instances of the `ActivityBarText` class that define the text to draw on the right, on the left, above, below and inside the bar.

Defining the Start and End Time of the Bar

The start time and end time of the rectangular area is defined by two properties of the `ActivityBarStyle` class.

The `ActivityBarStyle.FromProperty` property contains a string that represents the name of a property of the activity that defines the starting date where the activity bar should be painted. By default, the value is **StartTime**, meaning that the bar starts drawing at the start time of the activity. (See `IActivity.StartTime`) The **FromProperty** property must then define a property of type **DateTime**.

The `ActivityBarStyle.ToProperty` property contains the name of a property of an activity that defines the end date of the activity bar on screen. By default, the value is **EndTime**, meaning that the bar rectangle will end at the end time of the activity (See `IActivity.EndTime`).

The property defined by **ToProperty** can be of type **DateTime**. In this case, the date defines the end of the bar.

It can also be a duration of type **TimeSpan**. In this case, the end date of the bar is computed by adding the duration to the start date of the bar.

The property can also be a numeric value of type **Single**, **Double**, or **Int32**. In this case, the value of the property is considered to be a percentage of the duration of the activity (See `IActivity.Duration`.) The end date of the bar reflects the percentage of the start time of the activity.

For example, to draw a rectangle that displays the percentage completion of an activity, you create an activity bar style with the `FromProperty` set to `StartTime` and the `ToProperty` set to `WorkComplete`:

```
ActivityBarStyle style = ...
style.FromProperty = "StartTime";
style.ToProperty = "WorkComplete";
```

Notes:

1. When the **FromProperty** and the **ToProperty** are the same, no rectangular bar is displayed. This does not mean that the style renders nothing, since the style may define a start or end symbol.
2. Styles can be used to represent reservations, not activities. This is the case when using resources-oriented or reservations-oriented controls such as `ResourceChart` or `ReservationChart` controls. In this case, the properties defined by **ToProperty** and **FromProperty** can be properties located on the reservations. If such properties are not available on the reservations, they are taken from the activities.

Defining the Appearance and Height of the Bar

The appearance and height of the rectangular area is defined by the class `ActivityBar`. An instance of the `ActivityBarStyle` class contains one instance of **ActivityBar** that can be modified using the `Bar` property of the **ActivityBarStyle** class.

The **ActivityBar** class defines the following appearance properties:

Property	Description
<code>Color</code>	The color of the background of the bar.
<code>BorderColor</code>	The color of the border of the bar.
<code>HatchStyle</code>	The hatch style for the background of the bar when the FillStyle property is set to ActivityBarFillStyle.Hatch .
<code>FillStyle</code>	Indicates the fill style for the background of the bar.

SecondColor	Defines the second color of the bar. If this color is different from the Color property and FillStyle is not ActivityBarFillStyle.Hatch , then the bar will be drawn with a linear gradient between Color and SecondColor . When FillStyle is ActivityBarFillStyle.Hatch , this color is used as the background color of the hatch.
TopMargin	A value between 0 and 1 that defines the top margin of the bar with respect to a normal bar height, that is, a bar that has no top or bottom margin.
BottomMargin	A value between 0 and 1 that defines the top margin of the bar with respect to a normal bar height, that is, a bar that has no top or bottom margin.

Defining Start and End Symbols

The class **BarSymbol** defines the appearance of the symbols that appear on the left and right of the bar. These symbols are defined by the **StartSymbol** and **EndSymbol** properties of the **ActivityBarStyle** class.




The **BarSymbol** class defines the following properties:

Property	Description
Color	Defines the background color of the symbol.
BorderColor	Defines the color of the outline of the symbol.
Shape	One of the values of the BarSymbolShape enumeration that defines the shape of the symbol. (See the table of possible values of Shape .)

The possible values of **Shape** are:

Shape	Appearance
None	
UpPentagon	▲
DownPentagon	▼
Circle	●
Diamond	◆

UpTriangle	▲
DownTriangle	▼
RightTriangle	▶
LeftTriangle	◀
Square	■
SmallSquare	▪
DownArrow	↓
UpArrow	↑
CircleUpArrow	⬆
FilledCircleUpArrow	⬇
CircleDownArrow	⬇
FilledCircleDownArrow	⬆
SmallUpPentagon	▲
SmallDownPentagon	▼
VerticalBar	
Star	★
CircleUpTriangle	⬆
FilledCircleUpTriangle	⬇
CircleDownTriangle	⬇

FilledCircleDownTriangle	
CircleDiamond	
FilledCircleDiamond	

Defining Additional Text to Display

Five texts can be displayed: a text on the right, a text on the left, a text above, a text below and a text inside the bar. The text to be displayed is usually the value of a property of the rendered activity.

The `ActivityBarStyle` class defines five properties of type `ActivityBarText` for accessing the texts:

- ◆ **RightText**
- ◆ **LeftText**
- ◆ **InsideText**
- ◆ **TopText**
- ◆ **BottomText**

The definition of what is to be displayed is contained in the `Value` property of the `ActivityBarText` class.

For example, defining a style with the following C# code renders the start time of the activity on the right of the bar:

```
ActivityBarStyle style = ...
style.RightText.Value = "StartTime";
```

The **Value** property of the **ActivityBarText** class can be an expression based on the properties of the rendered activity. This expression is defined in *Expression Language Reference*.

Here are some other examples:

Example of value	Displayed Result
{StartTime:m}	Displays the start time of the activity formatted with the 'm' format that corresponds to the Month/Day pattern of the .NET date formatter.
Reservations	Displays the list of reservations of the rendered activity.
Name	Displays the name of the activity.

'Activity:' + Name	Concatenates the text 'Activity:' and the name of the activity.
{Duration:g}	The duration of the activity formatted with the 'g' pattern of the Time Span formatter.

Defining When a Bar Style Applies

When the Gantt sheet needs to render an activity or a reservation on the screen, first it looks in its collection of styles (ActivityBarStyles property) to collect all the styles that are relevant. It uses the StyleFor property of the ActivityBarStyle class to know which instances of the **ActivityBarStyle** class are relevant to the activity or the reservation.

The **StyleFor** property contains a Boolean expression that will be evaluated to determine if the style matches the object being rendered. The evaluation context is given by the activity being rendered when using activity-oriented controls such as the GanttChart. For other controls, the context will be the reservation. The evaluation context is used to resolve property names used in the **StyleFor** property.

Note: When using resource-oriented controls, the evaluation context used to resolve property names is the reservation being rendered. If a property cannot be located on the reservation, it will be searched in the corresponding activity.

Here is a list of examples:

Expression	Applies To
Normal	The style applies to activities that are not milestones or summaries.
IsSummary	The style applies to summary activities.
Milestone	The style applies to milestone activities.
Normal && WorkComplete != 0	The style applies to normal activities with the WorkComplete property not equal to zero.
StartTime > #1967/2/10#	The style applies to activities with a start time later than 1967/2/10.
Name == 'MyActivity'	The style applies to activities named ' MyActivity '.
ID == '1'	The style applies to the activity with the ID ' 1 '.
MyBooleanProperty	The style applies to the activity with your own Boolean property set to true .

Note: When several activity bar styles match one activity, the bars are superimposed in the order of the **ActivityBarStyle** collection.

For a complete description of expressions see *Expression Language Reference*.

Interacting and Styling

An activity bar style displays information based on activity or reservation properties. In particular, the **ActivityBarStyle.FromProperty** and **ActivityBarStyle.ToProperty** properties of the bar style are used to render the bar displayed by the style. If the properties referenced by the **FromProperty** and **ToProperty** are not read-only properties, it is possible to interactively edit those properties by clicking and dragging on the rendered bar. The following table shows the properties and the methods that can be used to control the behavior of a bar style:

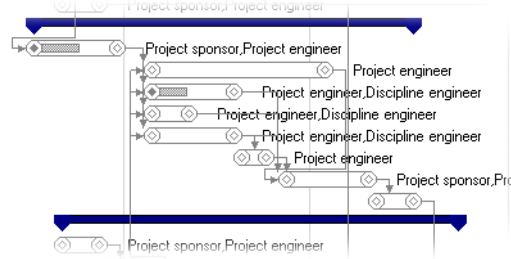
Property or Method	Description
Interactive	Gets or sets a value indicating whether this style is interactive.
CanEditFromProperty	Gets or sets a value indicating whether the from property of this style is editable.
CanEditToProperty	Gets or sets a value indicating whether the to property of this style is editable.
FromPropertyCursor	Gets or sets the cursor used to modify the from property using the mouse pointer.
ToPropertyCursor	Gets or sets the cursor used to modify the to property using the mouse pointer.
MoveCursor	Gets or sets the cursor used to indicate that a bar style can be moved.
ShowToolTip	Gets or sets a value indicating whether this style should show tooltips.
ShowEditingToolTip	Gets or sets a value indicating whether this style should show tooltips during edition.
ToolTip	Gets or sets the expression for tooltips.
SnapController	Gets or sets a time iterator that defines the way date-time values are snapped.
AllowConstraintLinkConnection	Gets or sets a value indicating whether this style can be used to connect constraint links.

See Also

Using the Predefined Behavior of Gantt Sheet Controls

Example of Styling

Here is a full C# example of code extracted from the *The Gantt Editor Sample* that will lead to the following graphical result:



You can take a look at *The Gantt Editor Sample* for more styling examples.

```

ActivityBarStyle activityBarStyle1 = new ActivityBarStyle();
ActivityBarStyle activityBarStyle2 = new ActivityBarStyle();
ActivityBarStyle activityBarStyle3 = new ActivityBarStyle();
ActivityBarStyle activityBarStyle4 = new ActivityBarStyle();

// The style for normal activities

activityBarStyle1.Bar.Color = Color.White;
activityBarStyle1.Bar.BorderColor = Color.Gray;
activityBarStyle1.EndSymbol.Color = Color.White;
activityBarStyle1.EndSymbol.BorderColor = Color.Gray;
activityBarStyle1.EndSymbol.Shape = BarSymbolShape.CircleDiamond;
activityBarStyle1.Name = "Activity";
activityBarStyle1.RightText.Value = "Reservations";
activityBarStyle1.StartSymbol.Color = Color.White;
activityBarStyle1.StartSymbol.BorderColor = Color.Gray;
activityBarStyle1.StartSymbol.Shape = BarSymbolShape.CircleDiamond;
activityBarStyle1.StyleFor = "Normal";

// Additional style for normal activities with a WorkComplete != 0

activityBarStyle2.Bar.BottomMargin = 0.25F;
activityBarStyle2.Bar.Color = Color.White;
activityBarStyle2.Bar.SecondColor = Color.Gray;
activityBarStyle2.Bar.BorderColor = Color.Gray;
activityBarStyle2.Bar.TopMargin = 0.25F;
activityBarStyle2.Bar.FillStyle = ActivityBarFillStyle.Hatch;
activityBarStyle2.Name = "Progress";
activityBarStyle2.StartSymbol.Color = Color.White;
activityBarStyle2.StartSymbol.BorderColor = Color.Gray;
activityBarStyle2.StartSymbol.Shape =
    BarSymbolShape.FilledCircleDiamond;
activityBarStyle2.StyleFor = "Normal && WorkComplete!=0";
activityBarStyle2.ToProperty = "WorkComplete";

// The style for Summary activities

```

```

activityBarStyle3.Bar.BottomMargin = 0.5F;
activityBarStyle3.Bar.Color = Color.Navy;
activityBarStyle3.EndSymbol.BorderColor = Color.Navy;
activityBarStyle3.EndSymbol.Color = Color.Navy;
activityBarStyle3.EndSymbol.Shape = BarSymbolShape.DownPentagon;
activityBarStyle3.Name = "Summary";
activityBarStyle3.StartSymbol.BorderColor = Color.Navy;
activityBarStyle3.StartSymbol.Color = Color.Navy;
activityBarStyle3.StartSymbol.Shape = BarSymbolShape.DownPentagon;
activityBarStyle3.StyleFor = "IsSummary";

// The style for Milestone activities

activityBarStyle4.Bar.TopMargin = 1F;
activityBarStyle4.Name = "Milestone";
activityBarStyle4.RightText.Value = "{StartTime:m}";
activityBarStyle4.StartSymbol.BorderColor = Color.Navy;
activityBarStyle4.StartSymbol.Color = Color.Navy;
activityBarStyle4.StartSymbol.Shape = BarSymbolShape.Diamond;
activityBarStyle4.StyleFor = "Milestone";
activityBarStyle4.ToProperty = "StartTime";

// Set the styles to the Gantt chart control

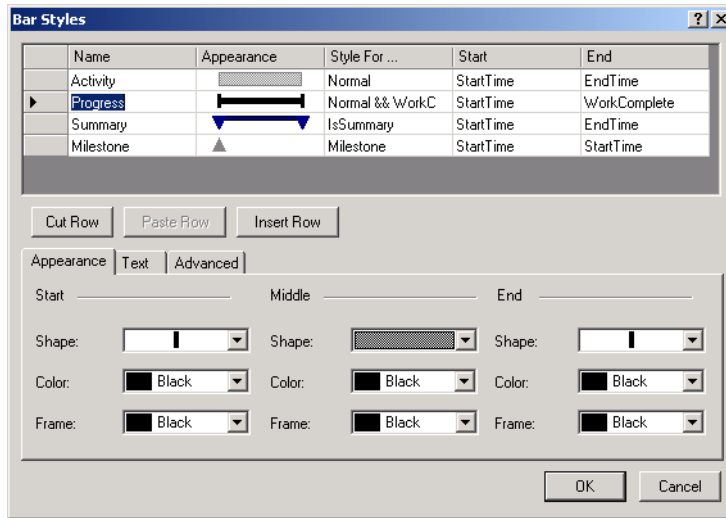
gantChart.ActivityBarStyles.Clear();
gantChart.ActivityBarStyles.AddRange(new ActivityBarStyle[] {
    activityBarStyle1,
    activityBarStyle2,
    activityBarStyle3,
    activityBarStyle4 });

```

Dialog Box Control for Styling Activity Bars

IBM® ILOG® Gantt for .NET provides a predefined dialog box for modifying the styles of activity bars. This dialog box is used at design time to allow the designer to define the styles of the activity bars. It can also be used in your application at run time to let the application user change the style of activity bars.

The following illustration shows the dialog box used to edit the style of activity bars.



The dialog box is implemented by the class `ActivityBarStylesDialog`.

In This Section

Modifying Styles

Describes how to modify the properties of the style in a table.

Programming the Activity Bar Style Dialog Box

Explains how to use the dialog box to customize the bar style.

Modifying Styles

Styles can be modified through a table of styles and a tab control that contains several pages for modifying the properties of the style selected in the table.

The Bar Style Table

The table of styles that are currently defined is shown in the top area of the dialog box.

Name	Appearance	Style For ...	Start	End
Activity		Normal	StartTime	EndTime
▶ Progress		Normal && WorkC	StartTime	WorkComplete
Summary		IsSummary	StartTime	EndTime
Milestone		Milestone	StartTime	StartTime

The table has the following columns:

Column	Description
Name	Gives the name of the style.
Appearance	Displays a preview of the bar defined by the style.
Style For...	Contains the expression that defines to which activity the style applies. (See <i>Defining When a Bar Style Applies.</i>)
Start	Contains the name of the property that defines the start date of the activity bar. (See <i>Defining the Start and End Time of the Bar.</i>)
End	Contains the name of the property that defines the end date of the activity bar. (See <i>Defining the Start and End Time of the Bar.</i>)

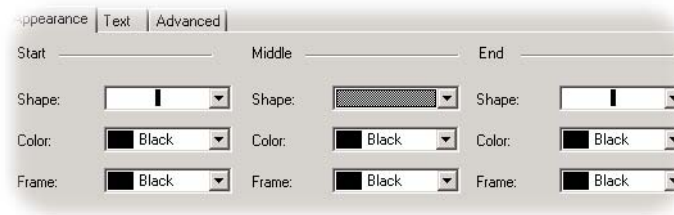
Beneath the table of styles are three buttons for cutting, pasting, and inserting a new row in the table.

The Tab Control

At the bottom is a tab control with three tab pages. The content of these tab pages depends on the style selected in the table.

The Appearance Page

The Appearance page is divided into three sections. The first section is for setting the shape and the colors of the start symbol of the bar style. The second section is for setting the shape and the colors of the rectangular bar. The last section is for setting the shape and colors of the end symbol.

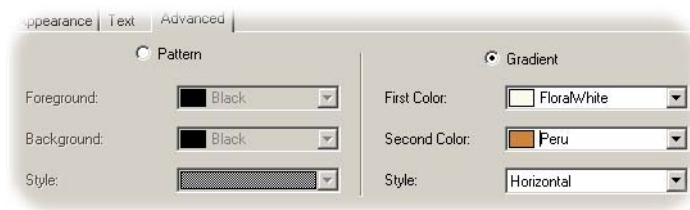


The Text Page

The Text page has five combo boxes for specifying the property of the activity to be displayed on the right, left, above, below or in the middle of the bar (See *Defining Additional Text to Display.*)



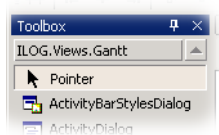
Advanced Page



The Advanced page is for specifying advanced appearance properties, such as for using a pattern or a gradient in the activity bar.

Programming the Activity Bar Style Dialog Box

The dialog box is a control that can be dragged from the toolbox in Visual Studio.



Once you have dragged the dialog box, you can use it in your code to customize the bar style of a GanttSheet instance.

Here is a small C# example of how to show the dialog box and set the result in a Gantt sheet. In the example, sheet is an instance of **GanttSheet**.

```

ActivityBarStylesDialog editor = new ActivityBarStylesDialog();
editor.Styles = sheet.ActivityBarStyles;
if (editor.ShowDialog() == DialogResult.OK)
{
    sheet.ActivityBarStyles.Clear();
    sheet.ActivityBarStyles.AddRange(editor.Styles);
}

```

Creating Owner-Drawn Gantt Components

If you need to have the entire control over what's being drawn inside a Gantt sheet, or if you want to modify the drawing of the time scale, you can provide some drawing code that will be used instead of the original code. This section explains how to create customized Gantt representations by using this owner-drawn feature.

In This Section

Providing User Code to Draw Gantt Table Cells

Describes how to control the drawing of each cell of a `GanttTable`.

Providing User Code to Draw Gantt Sheet Rows

Describes how to control the drawing of each row of a `GanttSheet`.

Providing User Code to Draw Activity Bar Styles

Describes how to control the drawing of each activity or reservation in a **`GanttSheet`**.

Providing User Code to Draw Time Scale Rows

Describes how to control the drawing of each row of the time scale.

Providing User Code to Draw Constraint Links

Describes how to control the drawing of constraint links in an `ActivitySheet`.

Providing User Code to Draw Gantt Table Cells

A `GanttTable` is made of rows and columns that display scheduling data: The `ActivityTable` displays one activity per row, and the `ResourceTable` displays one resource per row.

If you need to add custom drawings to the default drawing, or if you need to completely change this default drawing, you can provide some user code to manage your own drawing.

Enabling Gantt Table Owner Draw

Changing the `GanttTable.OwnerDraw` property to **`true`** will tell the **`GanttTable`** to use some user code to manage drawing operations. After setting this property, the `GanttTable.DrawTableCell` event will be raised each time a cell of the **`GanttTable`** is drawn.

Using the `GanttTable.DrawTableCell` Event

The `GanttTable.DrawTableCell` event carries a `DrawTableCellEventArgs` structure that contains information on the cell being drawn. The **`DrawTableCellEventArgs`** structure also contains methods to paint the cell using its original drawing. The following C# code shows how to handle the **`GanttTable.DrawTableCell`** event:

```
private void OnDrawTableCell(object sender, DrawTableCellEventArgs e) {
```

```

TreeTable table = e.TreeTable;
object value = table.GetValueAt(e.Row);
// If the node being drawn has children and is expanded, draw it using
// an italic font.
if (table.TreeModel.GetChildCount(value) != 0 &&
    table.IsRowExpanded(value)) {
    Font font = new Font(e.Font, e.Font.Style | FontStyle.Italic);
    e.DrawCell(e.Bounds, e.BackBrush, e.ForeBrush, font);
} else
    e.DrawCell();
}

```

This C# code fragment displays texts of expanded rows using an italic font.

Providing User Code to Draw Gantt Sheet Rows

A **GanttSheet** is made of rows that display scheduling data over time: The **ActivitySheet** displays one activity per row, and the **ScheduleSheet** displays several reservations per row. If you need to add custom drawings to the default drawing, or if you need to completely change this default drawing, you can provide some user code to manage your own drawing.

Enabling Gantt Sheet Owner Draw

Changing the **GanttSheet.OwnerDraw** property to **true** will tell the **GanttSheet** to use some user code to manage drawing operations. After setting this property, the **GanttSheet.DrawRow** event will be raised each time a row of the **GanttSheet** is drawn.

Using the GanttSheet.DrawRow Event

The **GanttSheet.DrawRow** event carries a **DrawGanttSheetRowEventArgs** structure that contains information on the row being drawn. The **DrawGanttSheetRowEventArgs** structure also contains methods to paint the row using its original drawing. The following C# code shows how to handle the **GanttSheet.DrawRow** event:

```

ActivitySheet sheet = new ActivitySheet();
sheet.OwnerDraw = true;
sheet.DrawRow += new DrawGanttSheetRowEventHandler(OnDrawSheetRow);

private void OnDrawSheetRow(object sender, DrawGanttSheetRowEventArgs e) {
    if (!e.IsRowSelected)
        e.DrawRow();
    else {
        if (e.IsPaintingBackground)
            e.Graphics.FillRectangle(e.BackBrush, e.Bounds);
        else {
            Font font = new Font(e.GanttSheet.Font, FontStyle.Italic);
            e.DrawRow(e.Bounds, e.BackBrush, e.ForeBrush, font);
            font.Dispose();
        }
    }
}

```

This C# code fragment displays texts of selected rows using an italic font.

***Note:** The event is raised twice per row being drawn: First it is raised to paint the row background, then to paint the row content. The `DrawGanttSheetRowEventArgs.IsPaintingBackground` property can be used to determine what should be drawn.*

Providing User Code to Draw Activity Bar Styles

The `ActivityBarStyle` class is used to represent an activity or reservation in a Gantt sheet. Instead of subclassing the `ActivityBarStyle` class, you can use the `ActivityBarStyle.OwnerDraw` property and provide some user code to handle the drawing.

Enabling Activity Bar Style Owner Draw

Changing the `ActivityBarStyle.OwnerDraw` property to `true` will tell the `ActivityBarStyle` to use some user code to manage drawing operations. After setting this property, the `ActivityBarStyle.DrawBarStyle` event will be raised each time the style is used to draw an activity or reservation. In addition, the `ActivityBarStyle.HitTestBarStyle` event will be raised to do hit testing on the customized bar style.

Using the ActivityBarStyle.DrawBarStyle Event

The `ActivityBarStyle.DrawBarStyle` event carries a `DrawBarStyleEventArgs` structure that contains information on the activity or reservation being drawn. The `DrawBarStyleEventArgs` structure also contains methods to paint the activity or reservation using its original drawing. The following C# code shows how to handle the `ActivityBarStyle.DrawBarStyle` event:

```
ActivityBarStyle style = new ActivityBarStyle();
style.OwnerDraw = true;
style.DrawBarStyle += new DrawBarStyleEventHandler(OnDrawStyle);

private void OnDrawStyle(object sender, DrawBarStyleEventArgs e) {
    if (e.IsRowSelected) {
        e.DrawBar();
        e.DrawStartSymbol();
        e.DrawEndSymbol();
        using (Font font = new Font(e.Context.Font, FontStyle.Italic)) {
            e.DrawLeftText(font, null);
            e.DrawRightText(font, null);
            e.DrawInsideText(font, null);
            e.DrawTopText(font, null);
            e.DrawBottomText(font, null);
        }
    } else
        e.DrawBarStyle();
}
```

This C# code fragment displays texts of selected rows using an italic font.

Using the `ActivityBarStyle.HitTestBarStyle` Event

The `ActivityBarStyle.HitTestBarStyle` event carries a `HitTestBarStyleEventArgs` structure that contains information on the activity or reservation being hit tested. The `HitTestBarStyleEventArgs` structure also contains methods to do default hit testing. The following C# code shows how to handle the `ActivityBarStyle.HitTestBarStyle` event:

```
ActivityBarStyle style = new ActivityBarStyle();
style.OwnerDraw = true;
style.HitTestBarStyle += new HitTestBarStyleEventHandler(OnHitTestStyle);

private void OnHitTestStyle(object sender, HitTestBarStyleEventArgs e) {
    e.DefaultHitTesting();
}
```

Providing User Code to Draw Time Scale Rows

The `TimeScaleRow` class is used to represent a row in a `TimeScale`. Instead of subclassing the `TimeScaleRow` class, you can use the `TimeScale.OwnerDraw` property and provide some user code to handle the drawing.

Enabling Time Scale Row Owner Draw

Changing the `TimeScale.OwnerDraw` property to `true` will tell the `TimeScale` to use some user code to manage drawing operations. After setting this property, the `TimeScale.DrawRow` event will be raised each time the time scale row is drawn.

Using the `TimeScale.DrawRow` Event

The `TimeScale.DrawRow` event carries a `DrawTimeScaleRowEventArgs` structure that contains information on the time scale row being drawn. The `DrawTimeScaleRowEventArgs` structure also contains methods to paint the time scale row using its original drawing. The following C# code shows how to handle the `TimeScale.DrawRow` event:

```
TimeScale tscale = new TimeScale();
tscale.DrawRow += new DrawTimeScaleRowEventHandler(OnDrawTimeScaleRow);

private void OnDrawTimeScaleRow(object sender, DrawTimeScaleRowEventArgs e) {
    if (e.Row.TimeScale.TimeLine.Adjusting)
        e.Graphics.FillRectangle(SystemBrushes.ControlLight, e.Bounds);
    Brush textBrush = new SolidBrush(e.Row.TextColor);
    Pen tickPen = new Pen(e.Row.TickColor);
    e.DrawRow(null, textBrush, e.Row.TextFont, tickPen);
    textBrush.Dispose();
    tickPen.Dispose();
}
```

This C# code fragment changes the background of the time scale when its time line is being adjusted, that is, when the user is dragging the time scale using the mouse to adjust the first visible time.

Providing User Code to Draw Constraint Links

Constraint links are used to represent constraints between activities in an `ActivitySheet`. To customize the drawing of constraint links, you can use the `ActivitySheet.ConstraintsOwnerDraw` property and provide some code to handle the drawing.

Enabling Constraint Links Owner Draw

When the `ActivitySheet.ConstraintsOwnerDraw` property is set to `true`, the `ActivitySheet` no longer manages drawing operations on constraint links. To handle the drawing you need to provide some code. After setting this property, the `ActivitySheet.DrawConstraintLink` event will be raised each time the activity sheet needs to draw its constraint links. In addition, the `ActivitySheet.HitTestConstraintLink` event will be raised to do hit testing on the customized constraint links.

Using the ActivitySheet.DrawConstraintLink Event

The `ActivitySheet.DrawConstraintLink` event carries a `DrawConstraintLinkEventArgs` structure that contains information on the constraint link being drawn. The `DrawConstraintLinkEventArgs` structure also contains methods to paint the constraint link using its original drawing. The following C# code shows how to handle the `ActivitySheet.DrawConstraintLink` event:

```
ActivitySheet sheet = new ActivitySheet();
sheet.DrawConstraintLink += new DrawConstraintLinkEventHandler(OnDrawLink);

private void OnDrawLink(object sender, DrawConstraintLinkEventArgs e) {
    if (e.Constraint.Type == ConstraintType.EndToStart) {
        PointF[] points = e.Points;
        e.Graphics.DrawLine(e.Pen, points[0], points[points.Length-1]);
    } else
        e.DrawConstraintLink();
}
```

This C# code fragment draws constraint links as straight lines for **EndToStart** constraints. Other constraints are drawn using the default drawing.

Using the ActivitySheet.HitTestConstraintLink Event

The `ActivitySheet.HitTestConstraintLink` event carries a `HitTestConstraintLinkEventArgs` structure that contains information on the constraint link being hit tested. The `HitTestConstraintLinkEventArgs` structure also contains methods to do default hit testing.

The following C# code shows how to handle the **ActivitySheet.HitTestConstraintLink** event:

```
ActivitySheet sheet = new ActivitySheet();
sheet.HitTestConstraintLink += new
HitTestConstraintLinkEventHandler(OnHitTestLink);

private void OnHitTestLink(object sender, HitTestConstraintLinkEventArgs e) {
    if (e.Constraint.Type == ConstraintType.EndToStart) {
        PointF[] points = e.Points;
        e.HitTest = PointInLine(points[0], points[points.Length-1], e.Point);
    } else
        e.DefaultHitTesting();
}
```

This C# code fragment assumes that constraint links for **EndToStart** constraints are being drawn as straight lines.

Creating Custom Gantt Representations

IBM® ILOG® Gantt for .NET provides ready-to-use controls to display the most common representations of scheduling data. When you need to create your own custom representation, you can use these controls and connect them together to create more powerful controls. This section describes the core controls of the product, and explains how to connect them together.

In This Section

Displaying Scheduling Data using Gantt Sheets

Describes how to develop with the Gantt sheets, controls for displaying activities and reservations along a time scale.

Using Time Scales

Describes how to use the TimeScale class, a control that display time scales.

Displaying Time-based Information

Describes the base class for displaying time-based information.

Synchronizing the Time of Several Controls

Explains how to synchronize the time of several controls.

Using Time Lines

Describes what time lines are, and how they can be customized.

Using Time Scrollbars

Describes how to use the `TimeScrollBar` class, a scroll bar that scrolls over time.

Displaying Scheduling Data using Gantt Sheets

IBM® ILOG® Gantt for .NET provides three types of control for displaying scheduling information over time: an activity sheet, a schedule sheet, and a reservation sheet.

The activity sheet displays activities in time and the schedule and reservation sheets display reservations of resources in time.

In this section, the term *Gantt sheet* is used to refer to an activity sheet, a schedule sheet, or a reservation sheet.

In This Section

Introducing the Activity Sheet, Schedule Sheet, and ReservationSheet

Describes the different types of Gantt Sheet controls.

Displaying Scheduling Data in Gantt Sheet Controls

Describes how to connect Gantt Sheet controls to visualize data.

Modifying the Appearance of Gantt Sheet Controls

Describes the properties that control the appearance of Gantt sheet controls.

Using the Predefined Behavior of Gantt Sheet Controls

Describes the predefined behavior in Gantt sheet controls.

Controlling the Displayed Time Interval

Describes the API for controlling the displayed time interval.

Hit Testing in the Gantt Sheet Controls

Describes the Hit testing API on Gantt Sheet controls.

Related Section

Displaying Time-based Information

Describes the base class for displaying time-based information.

Introducing the Activity Sheet, Schedule Sheet, and ReservationSheet

IBM® ILOG® Gantt for .NET provides three types of control for displaying scheduling information over time: an activity sheet, a schedule sheet, and a reservation sheet.

The activity sheet displays activities in time, the schedule and reservation sheet display the reservations of resources in time.

Activity Sheet

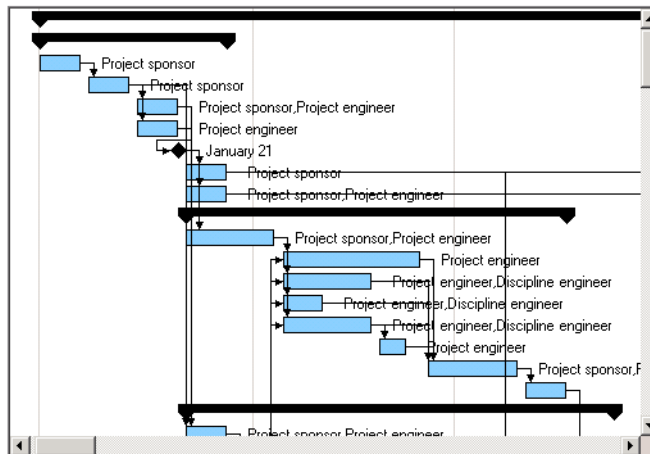
In the activity sheet, each row in the sheet represents the duration of one activity.

In the default implementation, activities with no child activities are displayed as plain horizontal bars. Activities with child activities are displayed as horizontal bars of a different color, delimited by special symbols at the end. These attributes are completely customizable.

In an activity sheet, constraints between activities are represented by directional polyline links. The type of the constraint determines how the link is attached to the activity bars.

The activity sheet is implemented by the `ActivitySheet` class.

The following illustration shows an activity sheet.



Schedule Sheet

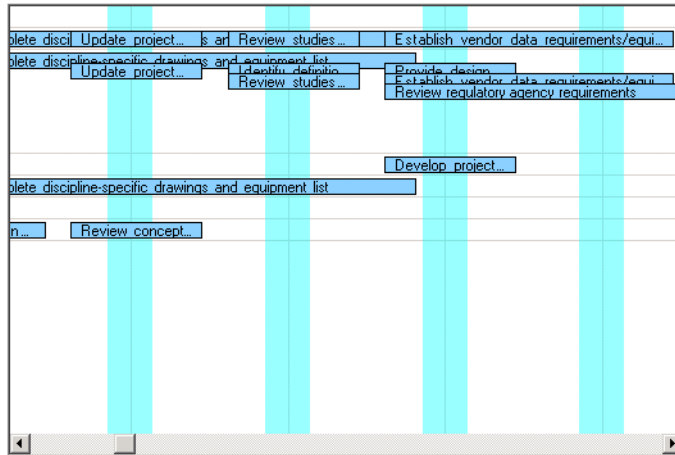
In the schedule sheet, each row of the sheet represents the activities assigned to a resource. Because the same resource can be reserved for more than one activity during the same time span, it could happen that several activity bars occupy the same horizontal area in the same row. To address this problem, a specific activity layout algorithm is used to position the bars for the best legibility. Three different layout algorithms are provided to manage potentially overlapping activity bars.

In the general case, one activity may reserve several resources and appear as several activity bars in the Schedule Sheet. For this reason, constraints between activities are not displayed in the schedule sheet.

The schedule sheet is implemented by the `ScheduleSheet` class.

The following illustration shows a schedule sheet.

Creating Custom Gantt Representations

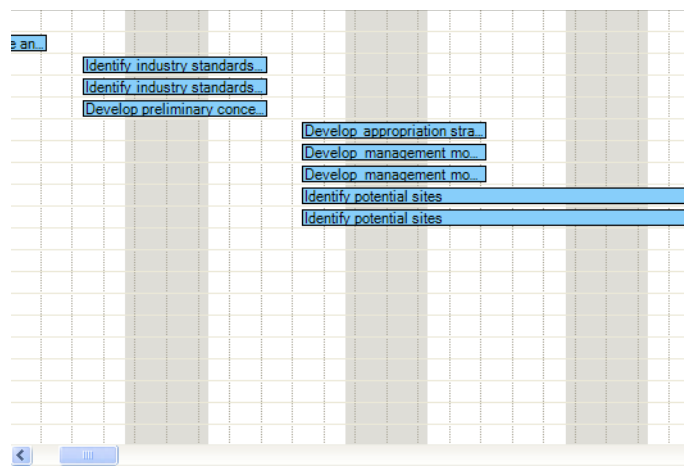


Reservation Sheet

In the reservation sheet, each row of the sheet represents a single reservation. In the general case, one activity may reserve several resources and appear as several activity bars in the reservation sheet. For this reason, constraints between activities are not displayed in the reservation sheet.

The reservation sheet is implemented by the `ReservationSheet` class.

The following illustration shows a reservation sheet.



The **ActivitySheet**, **ScheduleSheet**, and **ReservationSheet** classes share the same base class, the **GanttSheet** class. The **GanttSheet** class implements the functionalities that are common to the activity, schedule and reservation sheets, such as the connection to the Gantt data model or the appearance of activity bars.

In this section, the term *Gantt sheet* is used to describe an activity sheet, a schedule sheet, or a reservation sheet.

Displaying Scheduling Data in Gantt Sheet Controls

To display scheduling information, a Gantt sheet must be connected to a Gantt table. You connect a Gantt Sheet to a Gantt Table using the `RowController` property of the **GanttSheet** class.

An Activity Sheet must be connected to an Activity Table, a Schedule Sheet to a Resource Table, and a Reservation Sheet to a Reservation Table. For more information on the Gantt tables see *Introducing the Activity, Resource, and Reservation Tables*.

When the Gantt Sheet control is connected to a table, the Gantt Sheet control listens to events from the Gantt data model attached to the table and is updated for each modification of the data model. For example, when an activity is added to or removed from the data model, a corresponding row is added to or removed from an Activity Sheet control. When a resource is assigned to an activity in the Gantt data model, the activity appears in the corresponding row of the Schedule Sheet control.

When a Gantt sheet is embedded in a Gantt Chart, Schedule Chart or Reservation Chart control, all connections are made by the **GanttChart**, **ScheduleChart** and **ReservationChart** classes. You do not need to connect the tables and the sheets yourself.

In the ActivitySheet class

Method	Description
<code>GetActivityAt</code>	To get the activity displayed at a specific row.
<code>GetRowIndex</code>	To get the row where an activity is displayed.

In the `ScheduleSheet` class

Method	Description
<code>GetResourceAt</code>	To get the resource displayed at a specific row.
<code>GetRowIndex</code>	To get the row where a resource is displayed.

In the `ReservationSheet` class

Method	Description
<code>GetReservationAt</code>	To get the reservation displayed at a specific row.
<code>GetRowIndex</code>	To get the row where a reservation is displayed.

Modifying the Appearance of Gantt Sheet Controls

The Gantt sheet defines several properties for changing its appearance.

Color Properties

Property	Description
<code>BackColor</code>	The color used for the background of rows in the sheet.
<code>AlternatingBackColor</code>	The color used for alternating rows in the sheet. By default, this color is the same as the BackColor property.
<code>ForeColor</code>	The color of text in the sheet.
<code>BackgroundColor</code>	The color used for the area of the sheet that is not made up of rows.
<code>SelectionBackColor</code>	The background color of selected rows in the sheet.
<code>SelectionForeColor</code>	The color of text for selected rows in the sheet.

Font Property

The `Font` property defines the font used to draw text in the activity bars.

Horizontal Grid

The horizontal grid lines that separate the rows of the Gantt sheet are controlled by the following properties:

Property	Description
GridLineColor	The color of horizontal grid lines.
GridLineStyle	The style of horizontal grid lines.

Vertical Time Grid

The Gantt sheet can display vertical grid lines or areas that separate time periods on the time scale or display nonworking time with a specific appearance.

In addition, the Gantt sheet can display vertical grid lines that indicate a particular date in the project, such as the current date. The following properties control time grids and date indicators.

Property	Description
TimeGrids	A collection of time grids.
DateIndicators	A collection of vertical grid lines to mark specific dates.
VerticalGridToBottom	Indicates whether vertical grid lines are drawn in the area of the sheet that does not contain rows.

For more information, see *Using Time Grids and Date Indicators*.

Appearance of Activity Bars

The appearance of the rectangular bars that represent activities in the Gantt sheet can be completely customized by a styling mechanism. This styling mechanism is fully described in *Representing Activity Bars in Gantt Sheets*.

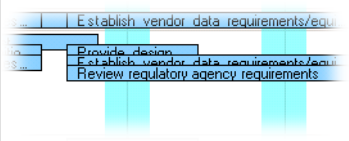
Miscellaneous Appearance Properties

Property	Description
BackgroundImage	An Image to display as the background of the Gantt sheet.
BorderStyle	The style of the border of the control.
ShowSelectedRows	Indicates whether the selected rows will be displayed in the selection colors.

Activity Layout Style in a Schedule Sheet

Each row in the schedule sheet represents the activities assigned to a resource. When the resource has several activities assigned for the same time period, the activity bars may overlap. The Schedule Sheet provides several ways of laying out the activity bars to remove overlapping.

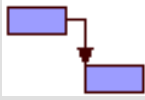
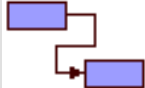
The layout style is controlled by the ActivityLayoutStyle property of the ScheduleSheet class. The property can take the values of the ActivityLayoutStyle enumeration:

Enumeration Member	Description
ActivityLayoutStyle.Simple	All activity bars on a given Gantt row have the same y position. They are all aligned on the top of the Gantt row and have the same height. The layout does not change the stacking order of the activity bars (z axis).
ActivityLayoutStyle.Pretty	The overlapping reservations are arranged with a slight vertical offset. The reservations are stacked so that the higher one (the one that has the greater y position) is displayed behind the lower one and both reservations are visible. 
ActivityLayoutStyle.Cascade	The Cascade layout is similar to the Pretty layout, except that it does not change the stacking order (z axis) of the activity bars.

Constraint Link Style

In an Activity Sheet the constraint links can be displayed in two different ways. The style of the constraint links is controlled by the `ConstraintLinkStyle` property of the `ActivitySheet` class.

Here are the different values of the `ConstraintLinkStyle` enumeration:

Enumeration Member	Description	Example
None	No constraint links are displayed.	
Direct	Constraints of type EndToStart are displayed as links with two segments if the destination activity starts after the end of the origin activity. The first segment is horizontal and the second is vertical. Otherwise, the constraint is displayed as in the Horizontal case.	
Horizontal	Constraints are displayed as links with 3 or 5 segments on the type of constraint. The first and last segments are horizontal.	

***Note:** You can customize the constraint links by setting the `ActivitySheet.ConstraintsOwnerDraw` property. See [Providing User Code to Draw Constraint Links](#) for details.*

Right-to-left Mode

All the controls can be used in right-to-left mode for Arabic and other languages that are written right-to-left. Note that bidirectional features of Windows® are available only in a bidirectional Microsoft® Windows® environment, such as an Arabic version of Microsoft® Windows®.

Using the Predefined Behavior of Gantt Sheet Controls

A Gantt sheet has the following predefined behavior.

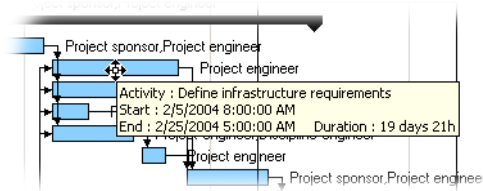
Selection

Clicking a row of the Gantt sheet selects the row. Clicking a row while pressing the **SHIFT** key selects the rows from the last selection anchor to the clicked row. Clicking a row while

pressing CTRL+SHIFT keys adds to the current selection, from the last selection anchor to the clicked row.

Tooltips

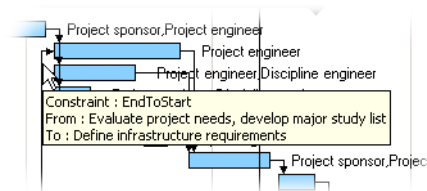
Tooltips appear in the Gantt sheet when the mouse hovers an activity bar. By default, the tooltip displays information related to the bar style being hovered: The bar style name, the activity name, and the from and to properties used by the bar style.



The tooltip depends on the bar style being hovered. The text appearing in the tooltip can be modified using the `ActivityBarStyle.ToolTip` property.

The value of this property is not a static string, but a string that represents an expression that will be evaluated using the current values of the activity. For more information on expressions see *Expression Language Reference*.

The `ActivitySheet` class also displays a tooltip when the mouse hovers a constraint. The text of this tooltip is an expression based on the property of the constraint defined by the `ConstraintToolTip` property.



You can disable the tooltips displayed in the Gantt sheet by setting the `ShowTooltips` property to **false**.

See the *Interacting and Styling* section for details about bar styles and interactions.

Changing the Start Time, End Time, or Duration of an Activity

In a Gantt Sheet, you can use the mouse pointer to change the property values displayed by an activity bar. The default bar styles of the different Gantt Sheet objects are designed to

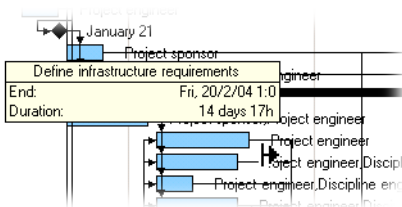
display the start time, the end time, and the duration of an activity. To modify these values, the user can use the following interactions:

Action	User Interaction
Change only the EndTime	Click at the end of the activity bar and drag the mouse pointer.
Change the whole interval	Click in the middle of the activity bar and drag the mouse pointer.

You can disable this default behavior by setting the CanEditActivities property to **false**.

***Note:** This default interaction allows you to modify the activity or reservation properties represented by an activity bar. By default, these are the **StartTime** and **EndTime** properties, but you can change these settings to represent any other activity or reservation property. See *Representing Activity Bars in Gantt Sheets and Interacting and Styling* for details.*

When an application user performs one of these operations, a tooltip appears showing the values that are modified:



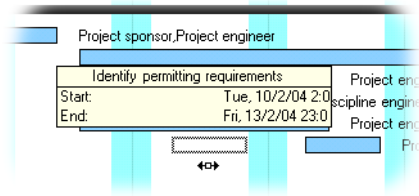
You can disable this tooltip by setting the ShowEditingTooltip property to **false**.

Editing can be done in two modes controlled by the InstantEditing property of the GanttSheet class:

Instant Editing Value	User Interaction
True	The data model is modified at each mouse move and the position of the activity bar automatically reflects the new values at each mouse move.
False	The data model is modified only when the mouse pointer is released. In this case, while the mouse is being dragged, a rectangle shows the position and size the activity will have when the mouse pointer is released.

Creating Custom Gantt Representations

The following illustration shows an activity being edited with **InstantEditing** set to **false**.



The GanttSheet class provides the BeforeEditActivity event that is raised before the activity is modified. For example, this event allows the application user to cancel the editing operation in some cases or to modify the chosen time interval.

Finally you can control the cursors that will be used during this operation through the following properties:

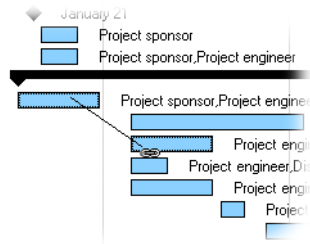
Property	Description
SizeEastCursor	Cursor used for changing the end time (or the start time in right-to-left mode).
SizeWestCursor	Cursor used for changing the start time (or the end time in right-to-left mode).
MoveCursor	Cursor used to indicate that an activity bar can be moved.
MovingActivityCursor	Cursor used for moving the activity bar.

See the *Interacting and Styling* section for details about bar styles and interactions.

Creating a Constraint in the Activity Sheet

In an Activity Sheet constraints can be created with the mouse pointer by selecting the origin and destination of the constraint. You can disable this behavior by setting the CanCreateConstraint property to **false**.

When the InstantEditing property is set to **true**, the application user creates a constraint by clicking the origin activity with the mouse pointer with the CTRL key pressed. The sheet then shows a line starting at the origin activity. The user must drag the mouse pointer to the destination activity and release the mouse pointer.



When `InstantEditing` is set to **false**, the application user does not need to press the `CTRL` key. The user can click the origin activity and drag the mouse pointer to the destination activity. When the mouse pointer is moved to a destination activity, the Gantt sheet automatically switches to the constraint creation mode.

Note that the constraint that is created is always an End-to-start constraint. If another type of constraint is required, the type of constraint must be changed later.

The `ActivitySheet` class provides the `BeforeCreateConstraint` event that allows the creation of the constraint to be canceled or its parameter to be modified before the constraint is created.

Finally, you can change the cursor used by the sheet when a constraint is created by using the `CreateConstraintCursor` property.

Changing the Resource Assigned to an Activity in the Schedule Sheet

In the Schedule Sheet the resource assigned to a reservation can be changed by clicking a reservation with the `SHIFT` key pressed and dragging the reservation to a new resource (another row of the sheet).

You can disable this behavior by setting the `CanMoveReservation` property to **false**.

You can modify the cursor used during this operation by using the `MoveReservationCursor` property.

The `ScheduleSheet` class defines the `BeforeMoveReservation` event that is raised before the operation is performed in the Gantt model and that allows you to cancel the operation or alert the application user in certain cases.

Adding a Reservation to the Schedule Sheet

In the Schedule Sheet, the application user can copy a reservation by clicking a reservation with the `CONTROL` key pressed and dragging the copy to a new resource (another row of the sheet).

You can disable this behavior by setting the `CanCopyReservation` property to **false**.

You can modify the cursor used during this operation by using the `CopyReservationCursor` property.

The `ScheduleSheet` class defines the `BeforeMoveReservation` event that is raised before the operation is performed in the Gantt model and that allows you to cancel the operation or alert the application user in certain cases.

Controlling the Displayed Time Interval

The `GanttSheet` class defines the following properties for controlling the displayed time period:

Property	Description
<code>FirstVisibleTime</code>	Gets or sets the first visible time of the Gantt Sheet.
<code>VisibleDuration</code>	Gets or sets the visible duration of the Gantt Sheet.
<code>LastVisibleTime</code>	Gets or sets the last visible time of the Gantt Sheet.
<code>VisibleTimeInterval</code>	Gets or sets the time interval displayed by the Gantt sheet in one single operation.

The `GanttSheet` class displays a horizontal scroll bar for scrolling in time. The minimum and maximum time of this scroll bar is automatically defined by the content of the Gantt data model displayed by the Gantt sheet. The minimum time is the earliest start time of the activities in the model. The maximum time is the latest end time of the activities in the model. Therefore, when the Gantt sheet displays the minimum and maximum values of the scroll bar, all the activities are visible.

The minimum and maximum time scrolling is controlled by the `TimeBounds` property of the **GanttSheet**.

A margin can be added around the minimum and maximum times. This margin is defined in pixels by the `TimeMargin` property. This margin is useful for making the text around activity bars more easily visible.

Although the time scroll bar has a minimum and maximum time, the Gantt sheet can display dates before or after this minimum and maximum scrolling time. Clicking the scroll bar arrows always allows scrolling in time before and after the minimum and maximum values.

When the Gantt sheet is scrolled in time with the horizontal scroll bar, the new time period is instantaneously displayed in the Gantt sheet at every mouse drag. You can change this behavior by setting the `InstantTimeScrolling` property to **false**. In this case, the new time period is displayed only when the mouse is released.

The **GanttSheet** class inherits from the `TimeControl` class, which is the base class for controls that display information based on time. The **TimeControl** class defines the first visible time of the control as well as the visible duration of time in the control. For more

information on this class and to learn how to synchronize several controls that display time, see *Displaying Time-based Information*.

Hit Testing in the Gantt Sheet Controls

If you need to create custom behavior in a Gantt sheet, you will need to get information on the Gantt sheet at a specific location on the screen. You can do this with the `HitTest` method. Calling this method returns an instance of the `HitTestInfo` structure that gives the following information:

Property	Description
Row	The Row under the specified point or -1 if the point is in the area of the sheet that does not contain rows.
Date	The date under the specified point.
Constraint	The constraint under the specified point or null (Nothing in Visual Basic®).
Reservation	The reservation under the specified point or null (Nothing in Visual Basic).
Activity	The activity under the specified point or null (Nothing in Visual Basic).
Type	One of the <code>HitTestType</code> values. The values are explained in the next table.

The following table explains the `HitTestType` values:

HitTestType values	Description
None	The background of the sheet.
Activity	An activity is located at the specified point.
Constraint	A constraint is located at the specified point.
Reservation	A reservation is located at the specified point.

Here is a small C# example that shows an event handler that handles the **Mouse down** event of an `ActivitySheet` instance and checks whether a double-click occurs on an activity:

```
private void OnActivitySheetMouseDown(object sender, MouseEventArgs e)
{
    ActivitySheet sheet = (ActivitySheet)sender;
    if (e.Clicks == 2)
    {
        GanttSheet.HitTestInfo info = sheet.HitTest(new Point(e.X, e.Y));
        if (info.Type == GanttSheet.HitTestType.Activity)
    }
}
```

```
    {  
        // double click occurs on activity  
    }  
}
```

Using Time Scales

The `TimeScale` control displays a scale of time in one, two, or three rows.

In This Section

Introducing the Time Scale Class

Describes the **TimeScale** class.

Modifying the Appearance of the Time Scale

Describes the properties for customizing the appearance of a time scale.

Using the Predefined Behavior of Time Scales

Describes the predefined behavior of a time scale.

Controlling the Displayed Time Interval

Describes how to control the way the time interval is displayed in a time scale.

Customizing Time Scale Rows

Describes how to create a customized time scale.

Synchronizing a Time Scale and a Time Grid

Describes how to ensure that a time scale is synchronized with time grids.

Related Section

Displaying Time-based Information

Describes the base class for displaying time-based information.

Introducing the Time Scale Class

The `TimeScale` control displays a scale of time in one, two, or three rows. A Time Scale is implemented by the class **TimeScale** in the `ILOG.Views.Gantt.Windows.Forms` namespace. The following illustration shows a time scale.

Qtr 3, 2004											
August 2004						September 2004					
27	28	29	30	31	01	02	03	04	05	06	07

By default, the time scale automatically adjusts the information displayed in each row when the displayed time period changes.

The **TimeScale** class can also be used as a means of navigating in time. For example, double-clicking in a time period zooms the time scale to the period. Thus, when this control is connected to other controls that display time-based information, it can be used as a time scroll bar for those controls.

Modifying the Appearance of the Time Scale

The TimeScale class defines several properties for changing its appearance.

Color properties

Property	Description
BackColor	The color used for the background of rows in the time scale.
ForeColor	The color of text in the sheet.

Font Property

The Font property defines the font used to draw text in the time scale.

Row Policy

By default, a time scale displays two rows and automatically updates the ticks and the labels of the scale, depending on the time period that is displayed. You can change this behavior by changing the row policy of the time scale with the RowPolicy property.

The **RowPolicy** property can take the values defined in the TimeScaleRowPolicy enumeration. This enumeration defines the following row policies:

Enumeration Member	Description
OneRow	The time scale displays one row and automatically adjusts the ticks to the displayed time interval.
TwoRows	The time scale displays two rows and automatically adjusts the ticks to the displayed time interval.

ThreeRows	The time scale displays three rows and automatically adjusts the ticks to the displayed time interval.
Manual	You define the number of rows; the adjustment of ticks to the displayed time period must be done manually. See <i>Customizing Time Scale Rows</i> .

Miscellaneous Appearance Properties

Property	Description
FlatStyle	The flatness style of the control.
BorderStyle	The style of the border of the control.
AutoSize	Indicates whether the time scale automatically computes its height based on the font and the number of rows.

Using the Predefined Behavior of Time Scales

Panning by Using the Mouse Pointer

The time scale can be panned using the mouse pointer by clicking the time scale and dragging the mouse.

You can disable this behavior by setting the `CanPan` property to **false**.

When the time scale is panned with the mouse pointer, the controls that are synchronized in time with this time scale also pan at every mouse-move event. If you want the scrolling of controls connected to the time scale to take place only when the mouse pointer is released, you can set the `InstantTimeScrolling` property of the `TimeScale` class to **false**.

Zooming Time In and Out

Double-clicking a time period in the time scale zooms the time scale to the time period. When the `SHIFT` key is pressed, a double-click zooms the time scale out.

Another way of zooming to a specific period of time is to press the `CONTROL` key, click a start date, and drag the mouse to an end date. When the mouse is released, the time scale zooms to the specified period.

You can disable this behavior by setting the `CanZoom` property to **false**.

Controlling the Displayed Time Interval

The `TimeScale` class defines the following properties for controlling how the time interval is displayed:

Property	Description
<code>FirstVisibleTime</code>	Gets or sets the first visible time of the Time Scale.
<code>VisibleDuration</code>	Gets or sets the visible duration of the Time Scale.
<code>LastVisibleTime</code>	Gets or sets the last visible time of the Time Scale.
<code>VisibleTimeInterval</code>	Gets or sets the time interval displayed by the Time Scale in one single operation.

The `TimeScale` class inherits from the `TimeControl` class, which is the base class for controls that display information based on time. The **TimeControl** class defines the first visible time of the control, as well as the visible duration of time in the control. For more information on this class and to learn how to synchronize several controls that display time, see *Displaying Time-based Information*.

Customizing Time Scale Rows

The default behavior of the `TimeScale` control is to compute automatically the information displayed in each row when the zoom level changes. You can customize or completely change this default behavior.

What you can do is based on the value of the `RowPolicy` property. The **RowPolicy** property can take the values defined in the `TimeScaleRowPolicy` enumeration. When the row policy is not **TimeScaleRowPolicy.Manual**, then the **TimeScale** will contain one, two, or three rows that will be automatically updated when the displayed time period changes. In this case, after the rows have been updated by the **TimeScale**, the **TimeScale** class raises the `AdjustRows` event. You can add an event handler to this event to modify some properties of the rows. Here is a small example that sets the color of text of the first row to white:

```
// Add an event handler to be notified when the time scale adjusts its rows.
timeScale.AdjustRows += new EventHandler(AdjustTimeScaleRows);

// The event handler
private void AdjustTimeScaleRows(object sender, EventArgs e)
{
    TimeScale timeScale = (TimeScale)sender;
    timeScale.Rows[0].TextColor = Color.White;
}
```

When the row policy is not **Manual**, then you normally do not change the time unit displayed in each row. This time unit is computed by the time scale to be adapted to the displayed time period of the time scale.

If you want to compute the time unit displayed by each row, use the row policy **TimeScaleRowPolicy.Manual**. When you use this policy, you must create the rows of the time scale and add them to the row collection (**TimeScale.Rows**). A row of the time scale is an instance of the `TimeScaleRow` class. You can then decide how many rows you want to display. Then, you must adjust the time unit displayed in each row manually by adding an event handler to the **AdjustRows** event.

Synchronizing a Time Scale and a Time Grid

When the `TimeScale` is used in synchronization with another control that displays time information, such as the `LoadChart`, it is advantageous to synchronize the vertical grid lines of the control, the time grid, with the ticks of the time scale.

The `TimeGridCollection` class contains the `Synchronize` method for performing this synchronization. This method must be called when the rows of the time scale have been changed. Assuming that `timeScale` is an instance of **TimeScale** and that `loadChart` is an instance of **LoadChart**, use the following C# code:

```
// Add an event handler to be notified when the time scale adjusts its rows.
timeScale.RowsChanged += new EventHandler(TimeScaleRowsChanged);

// The event handler
private void TimeScaleRowsChanged(object sender, EventArgs args)
{
    loadChart.TimeGrids.Synchronize(timeScale.Rows);
}
```

Displaying Time-based Information

IBM® ILOG® Gantt for .NET provides several controls that display information over time. These controls inherit from the class `TimeControl`, the base class for controls that display information over time. This control regroups all the functionalities required to display time-based information, such as converting dates to and from pixels on the screen.

In IBM ILOG Gantt for .NET, the following controls inherit from **TimeControl** to display time-based information:

- ◆ The `GanttSheet` class and its subclasses for displaying the duration of activities along a time scale.
- ◆ The `LoadChart` class for displaying the load of a resource along a time scale.
- ◆ The `TimeScale` class for displaying a scale of time.

Controlling the Displayed Time Interval

The **TimeControl** class defines the following properties for controlling the displayed time interval:

Property	Description
FirstVisibleTime	Gets or sets the first visible time.
VisibleDuration	Gets or sets the visible duration.
LastVisibleTime	Gets or sets the last visible time.
VisibleTimeInterval	Gets or sets the time interval displayed by the TimeControl class in one single operation.

In addition, there are methods for scrolling or zooming in time:

Method	Description
EnsureVisible	To change the first visible time, so that the specified time is in the displayed area.
ScrollLeft	To scroll time to the left.
ScrollRight	To scroll time to the right.
ZoomIn	To zoom in on the time.
ZoomOut	To zoom out.
SetTimeInterval	To change the zoom level, so that the time control displays the specified time interval. These methods can be used to zoom the time control with an animation feedback. To get an animation feedback when zooming, use the <code>ZoomAnimationSteps</code> property. When the value of this property is not zero, some intermediate steps between the current displayed time interval and the desired time interval are computed. The time control displays these intermediate steps to give a sense of animation.

Conversion from Time to Pixels and from Pixels to Time

The **TimeControl** class can convert a **DateTime** instance to positions on the control and back again:

Method	Description
GetTime	To compute the time that corresponds to the x coordinate of the control.
GetLocation	To compute the x coordinate that corresponds to the specified time.

Time Rectangle

A **TimeControl** object defines a part of its client area where the time information is displayed. This area is called the *time rectangle*. By default, this area is the client area, but it may be necessary to draw time-based information on a part of the control only. For example, the **LoadChart** class displays the load of the resources in a chart; the legend displayed by the load chart is not part of the time rectangle.

The time rectangle can be modified in a subclass by overriding the **TimeRectangle** property of the **TimeControl** class. You can also modify the time rectangle by setting margins around the client area. These margins are defined by the **TimeMargins** property of the **TimeControl** class.

Synchronizing the Time of Several Controls

The way a **TimeControl** instance displays time information is controlled by an object that implements the **ITimeLine** interface. This object defines the first visible time of the **TimeControl** instance, as well as the way to convert time information to pixels on the screen. This object is called the *time line* of the **TimeControl**. To be able to synchronize several controls that display time information, each control must share the same time line.

You can change the time line of a **TimeControl** instance using the **TimeLine** property.

Important: *The **TimeControl** class itself implements the **ITimeLine** interface, so an instance of the **TimeControl** class can be used as a time line.*

For example, to synchronize a **TimeScale** and a **LoadChart**, both subclasses of the **TimeControl** class, a single association is needed:

```
TimeScale timeScale = new TimeScale();
LoadChart loadChart = new LoadChart();
loadChart.TimeLine = timeScale;
```

You can also synchronize two controls by creating a time line and assigning it to several controls:

```
ITimeLine linearTimeLine = new LinearTimeLine();
TimeScale timeScale1 = new TimeScale();
TimeScale timeScale2 = new TimeScale();
timeScale1.TimeLine = linearTimeLine;
timeScale2.TimeLine = linearTimeLine;
```

How it Works

The load chart and the time scale share the same time line. Thus, any modification of the first visible time or zoom level of one of the controls applies to both controls.

Note that the `ScheduleChart` and `GanttChart` classes also implement the **ITimeLine** interface, although they do not inherit from the **TimeControl** class. Thus, these two controls can also be used as a time line.

Using Time Lines

A time line is an object responsible for converting time information to pixels on the screen. Each control that display time-based information uses a time line to convert **DateTime** objects into control client-coordinates.

The ITimeLine interface

A time line is represented by the `ITimeLine` interface. It is defined as follows:

```
public interface ITimeLine {
    event TimeLineChangeEventHandler TimeLineChanging;
    event TimeLineChangeEventHandler TimeLineChanged;
    DateTime FirstVisibleTime { get; set; }
    bool Adjusting { get; set; }
    double GetUnits(DateTime startTime, DateTime endTime);
    DateTime GetTime(DateTime reference, double units);
    void SetTimeInterval(DateTime startTime, DateTime endTime, float width);
}
```

The **ITimeLine** defines the first visible time of a control through the `FirstVisibleTime` property. When the **FirstVisibleTime** property is about to change, a `TimeLineChanging` event is raised. When the **FirstVisibleTime** has changed, a `TimeLineChanged` event is raised. When the **FirstVisibleTime** changes a lot, for example, when the application user adjusts the first visible time with a scroll bar, the value of the `Adjusting` property is **true**.

The **ITimeLine** also defines the conversion between time and pixels on the screen. This conversion defines the number of pixels that a time period occupies on the screen. Thus, it

defines the time zoom level. The following methods of the **ITimeLine** interface allow you to control this conversion:

Method	Description
GetUnits	To get the distance in pixels between two dates.
GetTime	To convert the specified distance in pixels to a date.
SetTimeInterval	To change the conversion method of the ITimeLine , so that the period displayed between two given dates will occupy the specified width.

When the conversion method is about to change, a **TimeLineChanging** event is raised. When the conversion has changed, a **TimeLineChanged** event is raised.

The LinearTimeLine class

In general, you do not have to implement the **ITimeLine** interface. A default implementation of the **ITimeLine** interface is provided by the class `LinearTimeLine`.

The **LinearTimeLine** is an implementation of the **ITimeLine** interface that does a linear conversion between dates and pixels. You need only to implement an **ITimeLine** if you need a representation of time where some time periods do not have the same size on the screen as others. For example, if you want to hide some days. To see how to implement the **ITimeLine** interface look at the example located in:

`<install-dir>/Samples/QuickStart/TimeScale`

Listening to Time Line Changes

The following C# code fragment shows how to listen for the modification of the first visible time and zoom level of a `TimeScale` control:

```
// Add an event handler to be notified when the time line has changed.
TimeScale timeScale = new TimeScale();
timeScale.TimeLineChanged += new TimeLineChangeEventHandler(TimeLineChanged);

// The event handler
private void TimeLineChanged(object sender, TimeLineChangeEventArgs args) {
    switch (args.EventType) {
        case TimeLineEventType.FirstVisibleTimeChanged:
            // the first visible time has changed
            break;
        case TimeLineEventType.ConversionChanged:
            // the zoom level has changed, the first visible time may
            // also have changed.
            break;
    }
}
```

Using Time Scrollbars

The `TimeScrollBar` class is a subclass of the `HScrollBar` class. It allows to scroll the time of controls that implement the `ITimeScrollable` interface. Most of the controls displaying time-based information implement the `ITimeScrollable` interface.

The `ITimeScrollable` Interface

The `ITimeScrollable` interface extends the `ITimeLine` interface by adding the concept of visible size. Here is the definition of the interface:

```
public interface ITimeScrollable : ITimeLine {
    event EventHandler VisibleWidthChanged;
    int VisibleWidth { get; }
}
```

For details on how to use the `ITimeLine` interface, see *Using Time Lines*.

Connecting a `TimeScrollBar` to a Control

To connect a `TimeScrollBar` to a control that implements the `ITimeScrollable` interface, use the `TimeScrollBar.TimeScrollable` property. Once connected, the scroll bar listens to events raised by the `ITimeScrollable` instance and update itself accordingly. For example, to connect a scroll bar to a `LoadChart`, use the following C# code:

```
LoadChart loadChart = new LoadChart();
TimeScrollBar scrollbar = new TimeScrollBar();
scrollbar.TimeScrollable = loadChart;
```

Note: The `LoadChart` class is a subclass of the `TimeControl` class that implements the `ITimeScrollable` interface.

Setting the Time Bounds of a `TimeScrollBar`

The `ITimeScrollable` interface do not specify the bounds of the interval in which the user will navigate using the scroll bar. Instead, use the `MinimumTime` and `MaximumTime` properties of the `TimeScrollBar` class to set the time bounds. The following C# code shows how to set the time bounds of the scroll bar to a one year time interval starting today:

```
LoadChart loadChart = new LoadChart();
TimeScrollBar scrollbar = new TimeScrollBar();
scrollbar.TimeScrollable = loadChart;
scrollbar.MinimumTime = DateTime.Today;
scrollbar.MaximumTime = DateTime.Today.AddYear(1);
```

***Note:** Setting time bounds does not prevent the user from scrolling out of these time bounds by clicking the scroll bar arrows. If you want to strictly limit the time range in which the user will navigate, you can use the `ITimeLine.TimeLineChanging` event to control the time line.*

Showing Tooltips

When using the scroll bar to modify the time line, a tooltip displays the current time. To enable or disable the tooltip, use the `ShowTooltip` property of the **TimeScrollBar** class.

To change the format of the date displayed by the tooltip, use the `DateFormat` property of the **TimeScrollBar** class.

Setting Time Margins

A margin can be added around the minimum and maximum time. This margin is defined in pixels by the `TimeMargin` property of the **TimeScrollBar** class.

Instant Time Scrolling

When the scroll bar is scrolled by dragging its slider, the connected control is instantaneously updated to reflect the new visible time interval. You can change this behavior by setting the `InstantTimeScrolling` property to **false**. In this case, the new time period will be displayed only when the mouse is released.

Reading and Writing Scheduling Data Using XML

IBM® ILOG® Gantt for .NET allows you to serialize and deserialize scheduling data to and from Scheduling Data Exchange Language (SDXL) files. This serialization and deserialization is done through the `GanttModelXmlSerializer` class.

The **`GanttModelXmlSerializer`** class is located in the `ILOG.Views.Gantt.Data` namespace. It contains several methods to serialize or deserialize scheduling data, depending on what you need to do. The **`GanttModelXmlSerializer`** class is independent of the Gantt data model implementation. Therefore, it can be used with any implementation.

In This Section

Overview of the SDXL Language

Introduces the SDXL language and its applications.

Serializing Scheduling Data to SDXL

Explains how to serialize scheduling data to an SDXL stream.

Deserializing Scheduling Data from SDXL

Explains how to deserialize an SDXL stream into scheduling data.

Customizing XML Serialization or Deserialization

Explains how to customize the serialization and deserialization of scheduling data.

Related Sections

Creating and Using Gantt Data Models

Introduces the Gantt data model, that is, the classes that contain the scheduling data you want to display.

Working with ADO.NET

Describes how to use Gantt data models with ADO.NET.

Overview of the SDXL Language

SDXL is an application of World Wide Web Consortium (W3C) XML. It is designed to meet the following needs:

- ◆ Serializing or deserializing the scheduling data of a Gantt data model to save scheduling data to SDXL files or to load the saved scheduling data from SDXL files.
- ◆ Exchanging scheduling data with other programs developed with or without IBM® ILOG® Gantt for .NET.

Since SDXL is an application of W3C XML, it can be read easily by other programs that are capable of reading XML files. It can also be translated into other formats by using technologies such as XSL.

Scenarios for Using SDXL

Since SDXL is a flexible XML application, the scope for using it is extensive. To have a general idea, imagine the following scenarios:

1. You use an IBM ILOG Gantt for .NET program to manage your projects. The program places a heavy demand on system resources, because it is connected to a database and uses the database to store the scheduling data. SDXL can help distribute this scheduling data. You can save your schedules in SDXL files and distribute them by means of a lightweight IBM ILOG Gantt for .NET program that does not need database connections.
2. You use an IBM ILOG Gantt for .NET program to manage your schedules. You can save your schedules in SDXL files. An optimization program loads the SDXL files and runs optimization algorithms to make your schedules more efficient. Then, you reload the optimized schedules by using your IBM ILOG Gantt for .NET program to visualize them.

SDXL Example

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<schedule version="5.5" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="sdxl.xsd">
  <title>Scheduling Data</title>
  <desc>ILOG Views Scheduling Data Exchange Language</desc>
  <resources>
    <resource id="R0" name="Gate 1" quantity="1" />
    <resource id="R1" name="Gate 2" quantity="1" />
    <resource id="R2" name="Gate 3" quantity="1" />
    <resource id="R3" name="Gate 4" quantity="1" />
    <resource id="R4" name="Gate 5" quantity="1" />
    <resource id="R5" name="Gate 6" quantity="1" />
    <resource id="R6" name="Gate 7" quantity="1" />
    <resource id="R7" name="Gate 8" quantity="1" />
    <resource id="R8" name="Gate 9" quantity="1" />
    <resource id="R9" name="Gate 10" quantity="1" />
  </resources>
  <activities dateFormat="d-M-yyyy H:m:s">
    <activity id="A0" name="BA-501" start="13-11-2003 8:45:0" end="13-11-2003
9:50:0" />
    <activity id="A1" name="BA-332" start="13-11-2003 11:0:0" end="13-11-2003
12:0:0" />
    <activity id="A2" name="BA-228" start="13-11-2003 13:0:0" end="13-11-2003
17:0:0" />
    <activity id="A3" name="BA-22" start="13-11-2003 9:0:0" end="13-11-2003
11:0:0" />
    <activity id="A4" name="BA-222" start="13-11-2003 12:0:0" end="13-11-2003
14:0:0" />
    <activity id="A5" name="BA-405" start="13-11-2003 12:0:0" end="13-11-2003
14:0:0" />
    <activity id="A6" name="BA-408" start="13-11-2003 15:0:0" end="13-11-2003
17:0:0" />
  </activities>
  <constraints />
  <reservations>
    <reservation resource="R8" activity="A6" />
    <reservation resource="R6" activity="A4" />
    <reservation resource="R3" activity="A5" />
    <reservation resource="R4" activity="A3" />
    <reservation resource="R0" activity="A0" />
    <reservation resource="R1" activity="A1" />
    <reservation resource="R2" activity="A2" />
  </reservations>
</schedule>

```

See Also

XML Schema for the SDXL (Scheduling Data eXchange Language)

Serializing Scheduling Data to SDXL

XML serialization of scheduling data is the process of converting this scheduling data to an SDXL file. You can choose to serialize a whole model or part of a model only, depending on your needs.

Serializing a Gantt Data Model

To serialize an entire data model, open a stream and serialize the model in it. In the following C# code fragment, the `model` variable refers to a Gantt data model, that is, a class that implements the `IGanttModel` interface.

```
StreamWriter writer = new StreamWriter("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.Serialize(writer, model);
writer.Close();
```

Serializing Activities

To serialize only some activities of a model, use the `GanttModelXmlSerializer.SerializeActivities` methods. In the following C# code fragment, assume that the `activities` variable refers to an array of activities, that is, objects that implement the `IActivity` interface. The activities should be located in the same Gantt data model.

```
StreamWriter writer = new StreamWriter("activities.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.SerializeActivities(writer, activities);
writer.Close();
```

Note: *The constraints that have source and destination activities located in the `activities` array will also be serialized.*

Serializing Resources

To serialize only some resources of a model, use the `GanttModelXmlSerializer.SerializeResources` methods.

In the following C# code fragment, assume that the `resources` variable refers to an array of resources, that is, objects that implement the `IResource` interface. The resources should be located in the same Gantt data model.

```
StreamWriter writer = new StreamWriter("resources.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.SerializeResources(writer, resources);
writer.Close();
```

Serializing Constraints

To serialize only some constraints of a model, use the `GanttModelXmlSerializer.SerializeConstraints` methods. In the following C# code fragment, assume that the `constraints` variable refers to an array of constraints, that is, objects that

implement the `IConstraint` interface. The constraints should be located in the same Gantt data model.

```
StreamWriter writer = new StreamWriter("constraints.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.SerializeConstraints(writer, constraints);
writer.Close();
```

Serializing Reservations

To serialize only some reservations of a model, use the `GanttModelXmlSerializer.SerializeReservations` methods. In the following C# code fragment, assume that the `reservations` variable refers to an array of reservations, that is, objects that implement the `IReservation` interface. The reservations should be located in the same Gantt data model.

```
StreamWriter writer = new StreamWriter("reservations.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.SerializeReservations(writer, reservations);
writer.Close();
```

See Also

Deserializing Scheduling Data from SDXL

Deserializing Scheduling Data from SDXL

XML deserialization of an SDXL file is the process of converting the SDXL file to scheduling data. You can choose to deserialize an entire SDXL file or part of an SDXL file only, depending on your needs.

Deserializing a Gantt Data Model

To deserialize an SDXL file into a data model, create a new model, open a stream on the SDXL file, and then deserialize the file into the model.

```
IGanttModel model = new SimpleGanttModel();
StreamReader reader = new StreamReader("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
serializer.Deserialize(reader, model);
reader.Close();
```

Note: *If the model is not empty to start with, the scheduling data read from the SDXL file will be appended to the existing model.*

Deserializing Activities

To deserialize only the activities of an SDXL file, use the `GanttModelXmlSerializer.DeserializeActivities` methods.

In the following C# code fragment, assume that the `constraints` variable refers to an array of constraints, that is, objects that implement the `IConstraint` interface.

```
IGanttModel model = new SimpleGanttModel();
StreamReader reader = new StreamReader("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
IActivities[] activities = serializer.DeserializeActivities(reader,
model, constraints);
reader.Close();
```

Note: *The activities and constraints returned are not added to the model. You need to add them yourself.*

Deserializing Resources

To deserialize only the resources of an SDXL file, use the `GanttModelXmlSerializer.DeserializeResources` methods.

```
IGanttModel model = new SimpleGanttModel();
StreamReader reader = new StreamReader("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
IResource[] resources = serializer.DeserializeResources(reader, model);
reader.Close();
```

Note: *The resources returned are not added to the model. You need to add them yourself.*

Deserializing Constraints

To deserialize only the constraints of an SDXL file, use the `GanttModelXmlSerializer.DeserializeConstraints` methods.

```
IGanttModel model = new SimpleGanttModel();
StreamReader reader = new StreamReader("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
IConstraint[] constraints = serializer.DeserializeConstraints(reader, model);
reader.Close();
```

Note: *The constraints returned are not added to the model. You need to add them yourself.*

Deserializing Reservations

To deserialize only the reservations of an SDXL file, use the `GanttModelXmlSerializer.DeserializeReservations` methods.

```
IGanttModel model = new SimpleGanttModel();
StreamReader reader = new StreamReader("file.sdxl");
GanttModelXmlSerializer serializer = new GanttModelXmlSerializer();
IReservation[] reservations = serializer.DeserializeReservations(reader,
model);
reader.Close();
```

Note: *The reservations returned are not added to the model. You need to add them yourself.*

See Also

Serializing Scheduling Data to SDXL

Customizing XML Serialization or Deserialization

The `GanttModelXmlSerializer` class provides the functionalities for serializing and deserializing basic scheduling data. If you customize the Gantt data model, you might need to customize the way scheduling data is serialized.

Adding Properties to Scheduling Entities

If your customized Gantt data model extends the default model only by adding .NET properties to the scheduling entities (activities, resources, constraints, or reservations), the new public properties are automatically serialized or deserialized by the library.

For example, if you added a .NET property called `Priority` to the activities of your custom model, the property will be saved automatically. The C# code of the custom activity class could look like:

```
public enum ActivityPriority
{
    Undefined,
    Low,
    Medium,
    High
}

public class CustomActivity : SimpleActivity
{
    ...
    [DefaultValue(ActivityPriority.Undefined)]
    public ActivityPriority Priority
    {
        get
```

```
    {  
        ...  
    }  
    set  
    {  
        ...  
    }  
}  
...  
}
```

Note: The *DefaultValue* attribute is used to prevent serialization if the priority is the default priority.

You may want to prevent a specific property from being serialized. In this case, you must mark the property with the `GanttPropertyAttribute`. The C# code could look like:

```
public class CustomActivity : SimpleActivity  
{  
    ...  
    [GanttProperty(false)]  
    public int DummyProperty  
    {  
        get  
        {  
            ...  
        }  
        set  
        {  
            ...  
        }  
    }  
    ...  
}
```

The complete C# code can be found in the sample **CustomGantt** located in:

```
<install-dir>\Samples\QuickStart\CustomGantt\cs\CustomGanttModel.cs
```

Customizing the SDXL Language

If the model has been customized in a way that cannot be represented by adding .NET properties, you can subclass the `GanttModelXmlSerializer` to control XML serialization or deserialization.

The **GanttModelXmlSerializer** class uses a DOM implementation, which makes customization easier. Each method that serializes scheduling data has an XML element as its first parameter. For example, the method that serializes an activity is:

```
protected virtual void SerializeActivity(XmlElement element, IActivity  
activity, Context context);
```

When an activity is serialized, this method is called with `element` as the node of the DOM where `activity` is serialized. The third parameter, `context`, provides information on the current serialization.

If you need to customize the serialization of activities, override this method in a subclass of **GanttModelXmlSerializer**, call the base implementation, and modify the DOM using the given `element`. The C# code could look like:

```
protected override void SerializeActivity(XmlElement element, IActivity
activity, Context context)
{
    base.SerializeActivity(element, activity, context);
    // Serialize my own data into the element node
}
```

You must also override the method responsible for deserializing an activity or you will not be able to get your custom information back through deserialization. The method that deserializes an activity is:

```
protected virtual IActivity DeserializeActivity(XmlElement element, Context
context);
```

To retrieve the information stored during serialization, browse the activity node to get your information back. The C# code could look like:

```
protected override IActivity DeserializeActivity(XmlElement element, Context
context)
{
    IActivity activity = DeserializeActivity(element, context);
    // Browse the element node of the DOM and modify activity accordingly.
}
```

The same customization can be achieved for resources, constraints, and reservations. See the complete API of `GanttModelXmlSerializer` for details.

Working with ADO.NET

ADO.NET provides a rich set of components for creating distributed, data-sharing applications. IBM® ILOG® Gantt for .NET provides several classes that ease the integration with ADO.NET.

In This Section

Overview of the Architecture

Gives an overview of the architecture used to combine IBM ILOG Gantt for .NET and ADO.NET.

Using Gantt Model Adapters

Describes how to use a Gantt model adapter, a class that helps writing bridges between a Gantt data model and an ADO.NET **DataSet**.

Using the Generic Gantt Model Adapter

Describes the `GenericGanttModelAdapter` class, a ready-to-use bridge between an instance of **DataSet** and a Gantt data model.

Developing a Custom Gantt Model Adapter

Describes how to create a custom Gantt model adapter to support a user specific **DataSet** schema.

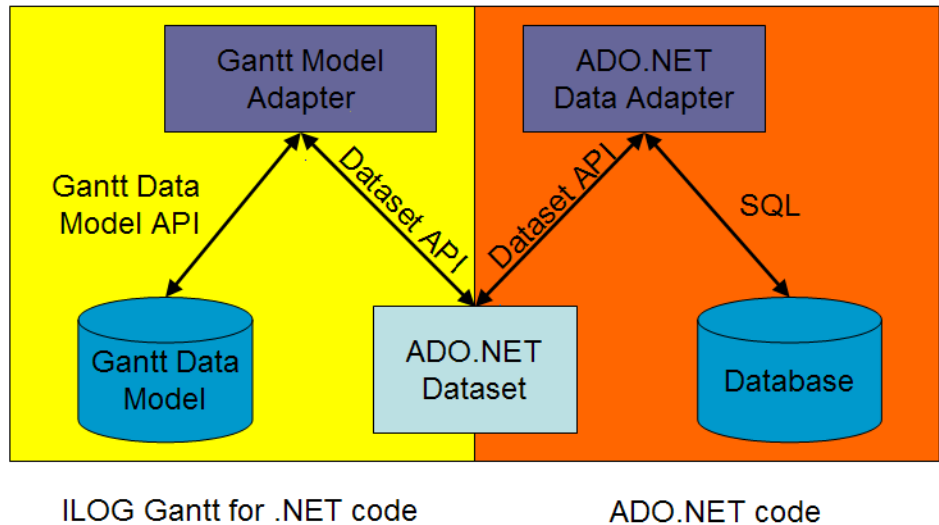
Overview of the Architecture

IBM® ILOG® Gantt for .NET provides a way to serialize a Gantt data model in a **DataSet** or to deserialize a Gantt data model from a **DataSet**. The object responsible for this is called a **Gantt Model Adapter**. It can be seen as a bridge between a Gantt data model and an ADO.NET **DataSet**.

A typical scenario for an application using IBM ILOG Gantt for .NET and a database can be:

- Read data from the database:
Fill a **DataSet** using an ADO.NET data adapter.
Use this **DataSet** to populate a Gantt data model using a Gantt model adapter.
- Modify the Gantt data model:
Use the Gantt data model API, or the controls available in IBM ILOG Gantt for .NET to modify the Gantt data model.
- Write back data to the database:
Use the modified Gantt data model to fill the **DataSet** using the Gantt model adapter.
Update the database using the **DataSet** and an ADO.NET data adapter.

The following figure illustrates this architecture:



See Also

The Data Model | Using Gantt Model Adapters

Using Gantt Model Adapters

A Gantt model adapter is a bridge between an ADO.NET **DataSet** and a Gantt data model. It's responsible for updating a Gantt data model from a **DataSet**, and for filling a **DataSet** from a Gantt data model.

A Gantt model adapter provides this bridge by mapping the Fill method, which changes the data in the **DataSet** to match the data in the Gantt data model, and Update, which changes the data in the Gantt data model to match the data in the **DataSet**.

The main class for Gantt model adapters is the GenericGanttModelAdapter class. This class does not make assumption on datasets structure, and thus can be used with almost any kind of **DataSet**. The only requirement for datasets is to store data representing the different Gantt entities (activities, resources, constraints, or reservations) into different tables.

The GenericGanttModelAdapter class can be configured to map the table and column names of the dataset to the Gantt data model.

In This Section

Updating a Gantt Data Model from a DataSet

Explains how to update a Gantt data model from a **DataSet**.

Filling a DataSet from a Gantt Data Model

Explains how to fill a Gantt **DataSet** with a Gantt data model.

Related Sections

Using the Generic Gantt Model Adapter

Describes the **GenericGanttModelAdapter** class, a ready-to-use bridge between an instance of **DataSet** and a Gantt data model.

Updating a Gantt Data Model from a DataSet

The Update method of the GenericGanttModelAdapter is called to resolve changes from a **DataSet** back to the Gantt data model.

When you call the **Update** method, the Gantt model adapter fills the Gantt data model using the rows located in the **DataSet**. Note that the Gantt model adapter clears the Gantt data model contents before starting the operation.

The following example demonstrates how to perform updates to a Gantt data model:

```
// Create a new dataset
Dataset dataset = new DataSet();
DataTable activities = new DataTable("Activities");
dataset.Tables.Add(activities);
activities.Columns.AddRange(new DataColumn[] {
    new DataColumn("ActivityID", typeof(string)),
```

```

        new DataColumn("ActivityName", typeof(string)),
        new DataColumn("ActivityStartTime", typeof(DateTime)),
        new DataColumn("ActivityEndTime", typeof(DateTime))
    });

    // Fill the dataset using a data adapter...
    // The code is not detailed here
    // as it's usually generated by the IDE.

    // Create the Gantt data model and configure the adapter.
    IGanttModel model = new SimpleGanttModel();
    GenericGanttModelAdapter adapter = new GenericModelAdapter(model);
    adapter.ActivitiesTableName = "Activities";
    adapter.ActivityIDColumnName = "ActivityID";
    adapter.ActivityProperties.AddRange(new
    GenericGanttModelAdapter.ColumnMapping[] {
        new GenericGanttModelAdapter.ColumnMapping("ActivityName", "Name"),
        new GenericGanttModelAdapter.ColumnMapping("ActivityStartTime",
"StartTime"),
        new GenericGanttModelAdapter.ColumnMapping("ActivityEndTime", "EndTime")
    });

    // Update the Gantt data model using the Gantt model adapter.
    adapter.Update(dataset);

```

Note: When the Gantt model is updated using the adapter, errors can be reported if the type of the objects in the dataset does not match the types of the Gantt model objects. For example, if a dataset column type is **int** whereas the type of the corresponding Gantt model property is **string**, there will be a mismatch. To fix this problem, you can listen to the **GenericGanttModelAdapter.RowUpdated** event, and try to make the conversion yourself in the event handler.

Filling a DataSet from a Gantt Data Model

The **Fill** method of the **GenericGanttModelAdapter** class is used to populate a **DataSet** from a Gantt data model. After the **DataSet** has been filled, it can be used like any other ADO.NET **DataSet**. In particular, it can be stored in a database.

The **Fill** method can be called several times on the same **DataSet**; the **DataSet** is updated according to the changes made to the Gantt data model since the last call to **Fill**. For example, the following C# code creates a Gantt model with three activities. Then, the model is used to fill a **DataSet**:

```

// Create the model and the activities
IGanttModel model = new SimpleGanttModel();
IActivity a1 = model.NewActivity();
a1.Name = "A1";
IActivity a2 = model.NewActivity();
a2.Name = "A2";
IActivity a3 = model.NewActivity();
a3.Name = "A3";

```



```

model.Activities.AddRange(new IActivity[] { a1, a2, a3 });

// Create a new dataset to store the Gantt model
DataSet dataset = new DataSet();
DataTable activities = new DataTable("Activities");
dataset.Tables.Add(activities);
activities.Columns.AddRange(new DataColumn[] {
    new DataColumn("ActivityID", typeof(string)),
    new DataColumn("ActivityName", typeof(string)),
    new DataColumn("ActivityStartTime", typeof(DateTime)),
    new DataColumn("ActivityEndTime", typeof(DateTime))
});
// Create and configure the adapter.
GenericGanttModelAdapter adapter = new GenericModelAdapter(model);
adapter.ActivitiesTableName = "Activities";
adapter.ActivityIDColumnName = "ActivityID";
adapter.ActivityProperties.AddRange(new
GenericGanttModelAdapter.ColumnMapping[] {
    new GenericGanttModelAdapter.ColumnMapping("ActivityName", "Name"),
    new GenericGanttModelAdapter.ColumnMapping("ActivityStartTime",
"StartTime"),
    new GenericGanttModelAdapter.ColumnMapping("ActivityEndTime", "EndTime")
});

// Populate the dataset
adapter.Fill(dataset);

```

The table of the **DataSet** that represents the activities now contains three rows. At this point, for example, the **DataSet** can be used with a **DataAdapter** to update a database. For details on how to achieve this, see *Accessing Data with ADO.NET* in the .NET Framework documentation.

The following C# code adds a new activity to the Gantt data model, modifies an existing activity, and deletes another activity. Then, the **DataSet** is filled again:

```

// Add a new activity
IActivity a4 = model.NewActivity();
a4.Name = "A4";
model.Activities.Add(a4);

// Modify a1
a1.Info = "Info on A1";

// And delete a3
model.Activities.Remove(a3);

// Populate the dataset
adapter.Fill(dataset);

```

The table of the **DataSet** that represents the activities now contains four rows. The following table shows the state of each row:

Row	Row State
A1	Modified
A2	Unchanged
A3	Deleted
A4	Added

```
adapter.Update(dataset);
```

***Note:** When the dataset is filled using the adapter, errors can be reported if the type of the objects in the Gantt model does not match the types of the objects in the dataset. For example, if a dataset column type is **int** whereas the type of the corresponding Gantt model property is **string**, there will be a mismatch. To fix this problem, you can listen to the **GenericGanttModelAdapter.RowFilled** event, and try to make the conversion yourself in the event handler.*

Using the Generic Gantt Model Adapter

IBM® ILOG® Gantt for .NET provides a ready-to-use Gantt model adapter through the **GenericGanttModelAdapter** class. The **GenericGanttModelAdapter** class can be configured to be used in many scenarios by mapping table and columns of a **DataSet** on the Gantt data model.

The **GenericGanttModelAdapter** class provides a straightforward mechanism to map a dataset on a Gantt data model by defining several properties that let the adapter know the dataset structure.

Activity Related Properties

The following table shows the properties related to activities:

Method	Description
ActivitiesTableName	Gets or sets the name of the table for activities.
ActivityIDColumnName	Gets or sets the name of the column that contains activity identifiers in the activities table.

ActivityParentIDColumnName	Gets or sets the name of the column that contains parent activity identifiers in the activities table.
ActivityIndexColumnName	Gets or sets the name of the column that contains the position of an activity relative to its siblings in the activities table.
ActivityProperties	Gets the collection of column mappings for activities.

Resource Related Properties

The following table shows the properties related to resources:

Method	Description
ResourcesTableName	Gets or sets the name of the table for resources.
ResourceIDColumnName	Gets or sets the name of the column that contains resource identifiers in the resources table.
ResourceParentIDColumnName	Gets or sets the name of the column that contains parent resource identifiers in the resources table.
ResourceIndexColumnName	Gets or sets the name of the column that contains the position of a resource relative to its siblings in the resources table.
ResourceProperties	Gets the collection of column mappings for resources.

Constraint Related Properties

The following table shows the properties related to constraints:

Method	Description
ConstraintsTableName	Gets or sets the name of the table for constraints.
ConstraintFromActivityIDColumnName	Gets or sets the name of the column that contains the identifier of the constraint source activity in the constraints table.

ConstraintToActivityIDColumnName	Gets or sets the name of the column that contains the identifier of the constraint destination activity in the constraints table.
ConstraintProperties	Gets the collection of column mappings for constraints.

Reservations Related Properties

The following table shows the properties related to reservations:

Method	Description
ReservationsTableName	Gets or sets the name of the table for reservations.
ReservationActivityIDColumnName	Gets or sets the name of the column that contains the identifier of the reservation activity in the reservations table.
ReservationResourceIDColumnName	Gets or sets the name of the column that contains the identifier of the reservation resource in the reservations table.
ReservationProperties	Gets the collection of column mappings for reservations.

Configuring the Adapter using the Wizard

When designing the **GenericGanttModelAdapter** in Visual Studio, you can use the wizard available from the smart tags of the adapter. The wizard has four pages, one for each Gantt model entity, that helps you configure the mapping between the **DataSet** and the Gantt data model.

Developing a Custom Gantt Model Adapter

You need to develop a custom Gantt model adapter when the generic Gantt model adapter cannot be used because the database structure cannot be mapped to the Gantt data model. This may be the case, for example, if the type of a database column cannot be converted to the type expected by the corresponding Gantt data property.

To develop a Gantt model adapter, you need to know the structure of the datasets that will be used with this adapter.

The only requirement for the **DataSet** structure is to store Gantt entities into different tables. Furthermore, it is not necessary for a Gantt model adapter to handle each kind of Gantt entities. For example, if the data located in the database does not contain constraints information, the Gantt model adapter can skip constraints related operations. To handle a specific Gantt entity, the Gantt model adapter must implement a specific method that returns the table of the dataset dedicated to this specific Gantt entity: **GetTableForActivities** for activities, **GetTableForResources** for resources, **GetTableForConstraints** for constraints, **GetTableForReservations** for reservations.

Supporting Activities

To support activities, a Gantt model adapter must implement the following methods:

Method	Description
GetTableForActivities	Returns the table of the dataset used to store activities.
GetActivityID	Retrieves the activity identifier from the specified row.
FillActivity	Is called to fill a data row using the specified activity.
UpdateActivity	Is called to update an activity using the specified data row.

In addition, the following methods can be overridden to provide optional features:

Method	Description
GetParentActivityID	Retrieves the parent activity id from the specified row. Override this method if the table for activities handles hierarchical activities, that is, if an activity can have sub-activities.
GetActivityIndex	Retrieves the activity index from the specified row. Override this method if the table for activities handles indexes for activities.

Supporting Resources

To support resources, a Gantt model adapter must implement the following methods:

Method	Description
GetTableForResources	Returns the table of the dataset used to store resources.
GetResourceID	Retrieves the resource identifier from the specified row.
FillResource	Is called to fill a data row using the specified resource.
UpdateResource	Is called to update an resource using the specified data row.

In addition, the following methods can be overridden to provide optional features:

Method	Description
GetParentResourceID	Retrieves the parent resource id from the specified row. Override this method if the table for resources handles hierarchical resources, that is, if a resource can have sub-resources.
GetResourceIndex	Retrieves the resource index from the specified row. Override this method if the table for resources handles indexes for resources.

Supporting Constraints

To support constraints, a Gantt model adapter must implement the following methods:

Method	Description
GetTableForConstraints	Returns the table of the dataset used to store constraints.
GetConstraintFromActivityID	Retrieves the source activity identifier from the specified row.
GetConstraintToActivityID	Retrieves the destination activity identifier from the specified row.

FillConstraint	Is called to fill a data row using the specified constraint.
UpdateConstraint	Is called to update a constraint using the specified data row.

Supporting Reservations

To support reservations, a Gantt model adapter must implement the following methods:

Method	Description
GetTableForReservations	Returns the table of the dataset used to store reservations.
GetReservationActivityID	Retrieves the activity identifier from the specified row.
GetReservationResourceID	Retrieves the resource identifier from the specified row.
FillReservation	Is called to fill a data row using the specified reservation.
UpdateReservation	Is called to update a reservation using the specified data row.

Using the Clipboard to Store Scheduling Data

IBM® ILOG® Gantt for .NET provides the `GanttClipboard` class for storing and retrieving scheduling data to and from the clipboard.

The **`GanttClipboard`** class is located in the `ILOG.Views.Gantt.Windows.Forms` namespace. It contains several methods to help manage basic clipboard operations, such as copy, cut, and paste. The data format used inside the clipboard is the SDXL format. The **`GanttClipboard`** class is independent of the Gantt data model implementation. Therefore, it can be used with any implementation.

In This Section

Storing Scheduling Data in the Clipboard

Explains how to store scheduling data in the clipboard.

Retrieving Scheduling Data from the Clipboard

Explains how to retrieve scheduling data previously stored in the clipboard.

Related Sections

Reading and Writing Scheduling Data Using XML

Describes how to import or export scheduling data into or from a Gantt data model using XML.

Populating Gantt Data Models

Explains the basic steps for populating a Gantt data model.

Storing Scheduling Data in the Clipboard

The `GanttClipboard` class is for storing activities and resources in the clipboard in the SDXL format. A description of the relevant methods available in the **GanttClipboard** class follows:

Method	Action
<code>CopyToClipboard(IActivity[])</code>	Copies activities to the clipboard. The specified activities are serialized in an SDXL stream and the stream is copied to the clipboard.
<code>CopyToClipboard(IResource[])</code>	Copies resources to the clipboard. The specified resources are serialized in an SDXL stream and the stream is copied to the clipboard.
<code>CutToClipboard(IActivity[])</code>	Cuts activities to the clipboard. The specified activities are serialized in an SDXL stream and the stream is copied to the clipboard. Then, the activities are removed from their model.
<code>CutToClipboard(IResource[])</code>	Cuts resources to the clipboard. The specified resources are serialized in an SDXL stream and the stream is copied to the clipboard. Then, the resources are removed from their model.

Example

The following C# code sample shows how to put some activities in the clipboard using the **CopyToClipboard** method:

```
// Create a Gantt Data Model
IGanttModel model = new SimpleGanttModel();
// Populate it
IActivity a1 = model.NewActivity();
IActivity a2 = model.NewActivity();
model.Activities.AddRange(new IActivity[] { a1, a2 });
// Then copy to the clipboard
GanttClipboard.CopyToClipboard(new IActivity[] { a1, a2 });
```

Retrieving Scheduling Data from the Clipboard

The `GanttClipboard` class allows activities, resources, and constraints to be retrieved from the clipboard. A description of the relevant methods available in the **`GanttClipboard`** class follows:

Method	Action
<code>GetActivities</code>	Retrieves the activities located in the clipboard. Note that the activities and constraints returned are not added to the model. You need to add them yourself.
<code>GetResources</code>	Retrieves the resources located in the clipboard. Note that the resources returned are not added to the model. You need to add them yourself.

Example

The following C# code sample shows how to retrieve the activities located in the clipboard to add them into a Gantt data model:

```
// Create a Gantt Data Model
IGantModel model = new SimpleGanttModel();
// Populate it
IActivity a1 = model.NewActivity();
IActivity a2 = model.NewActivity();
model.Activities.AddRange(new IActivity[] { a1, a2 });
// Copy to the clipboard
GanttClipboard.CopyToClipboard(new IActivity[] { a1, a2 });
// Clear the activities of the model
model.Activities.Clear();
// Retrieve the activities from the clipboard
IConstraint[] constraints = null;
IActivity[] activities = GanttClipboard.GetActivities(model, out constraints);
// Add them to the model
model.Activities.AddRange(activities);
```


Managing Undo/Redo in a Gantt Data Model

IBM® ILOG® Gantt for .NET provides a class that enables undo/redo operations on a Gantt data model: the `GanttModelUndoManager` class.

The **`GanttModelUndoManager`** class is located in the `ILOG.Views.Gantt.Data` namespace. An undo manager records events triggered by a Gantt data model when it is modified. The recorded event sequence is used to undo or redo the actions taken on the Gantt data model. The **`GanttModelUndoManager`** class is independent of the Gantt data model implementation. Therefore, it can be used with any implementation.

In This section

Enabling Undo/Redo

Explains how to enable undo/redo on a data model.

Disabling Undo/Redo

Explains how to disable undo/redo on a data model.

Undoing Modifications

Explains how to undo modifications.

Redoing Modifications

Explains how to redo modifications.

Grouping Modifications

Explains how to group several modifications to the model in a macro.

Enabling Undo/Redo

To enable undo/redo on a data model, create an instance of the `GanttModelUndoManager` class and connect it to the data model as follows:

```
SimpleGanttModel model = new SimpleGanttModel();
GanttModelUndoManager undoManager = new GanttModelUndoManager();
undoManager.GanttModel = model;
```

The undo manager then records all the events triggered by the model.

Disabling Undo/Redo

To disable undo/redo on a data model, disconnect the undo manager from the data model as follows:

```
undoManager.GanttModel = null;
```

The undo manager then stops recording the events triggered by the model.

Undoing Modifications

After enabling undo/redo by connecting an undo manager to a data model, use the `GanttModelUndoManager.CanUndo` property to know whether undoing a modification is possible or not. Then, call the `GanttModelUndoManager.Undo` method to undo the modification:

```
if (undoManager.CanUndo)
    undoManager.Undo();
```

Redoing Modifications

Use the `GanttModelUndoManager.CanRedo` property to know whether redoing a modification is possible or not. Then, call the `GanttModelUndoManager.Redo` method to redo the modification:

```
if (undoManager.CanRedo)
    undoManager.Redo();
```

Grouping Modifications

You can group several modifications to the model in a macro. This allows the undo manager to consider this group of modifications as a single modification. Therefore, the undo manager can undo this group of modifications in one atomic operation. To create a macro, use the following C# code:

```
undoManager.BeginMacro();  
// Modify the model  
undoManager.EndMacro();
```


Using Predefined Dialog Boxes for Editing Scheduling Data

IBM® ILOG® Gantt for .NET provides several predefined dialog boxes for editing the scheduling data.

The dialog boxes include:

- ◆ A dialog box for editing an activity.
- ◆ A dialog box for editing a resource.
- ◆ A dialog box for editing a constraint.

This section describes how to use and customize these editing functions.

In This Section

Editing an Activity Using the Predefined Dialog Box

Describes how to edit and customize through the ActivityDialog class.

Editing a Resource Using the Predefined Dialog Box

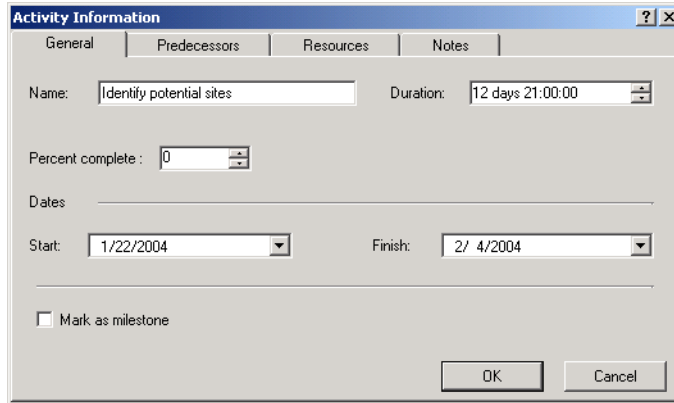
Describes how to edit a resource through the ResourceDialog class.

Editing a Constraint Using the Dialog Box

Describes how to edit and customize through the ConstraintDialog class.

Editing an Activity Using the Predefined Dialog Box

IBM® ILOG® Gantt for .NET provides a dialog box for editing the properties of an activity. This dialog box is for editing the basic information on an activity, such as the name, the start, or the end time, and also the constraints and the resources assigned to this activity (the reservations).



This dialog box has the following tab pages:

Tab Page	Description
General	Is for editing basic information on the activity such as the name, start date, end date, and duration.
Predecessors	Is for editing the predecessor constraints of the activity.
Resources	Is for editing the resources assigned to the activity (the reservations).
Notes	Is for editing notes on the activity.

You can remove the Predecessors and Resources tab pages by setting the `CanEditConstraints` and `CanEditReservations` properties to **false**.

Here is a C# code fragment that opens the dialog box on the current activity of an activity table:

```
ActivityDialog activityDialog = new ActivityDialog();

public void ShowActivityProperties(ActivityTable table)
{
    IActivity current = table.GetCurrentActivity();
```

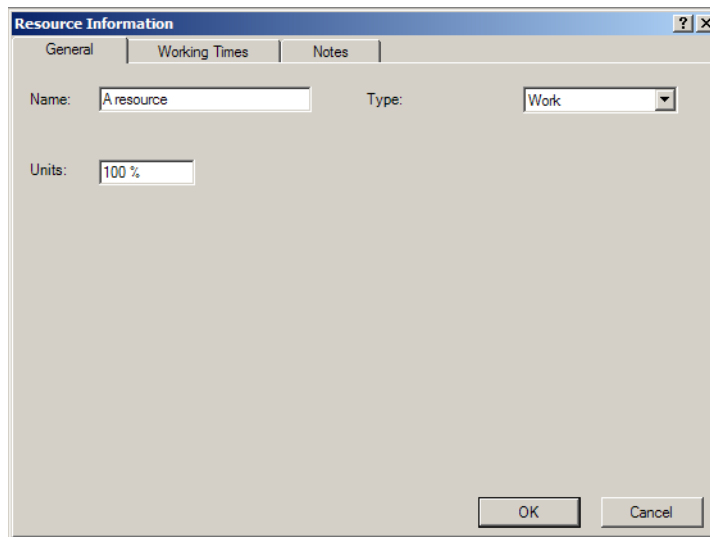
```
if (current != null)
{
    activityDialog.Activity = current;
    activityDialog.ShowDialog(this);
}
}
```

You can customize this dialog box to fit your specific needs. You can add a tab page or add controls in existing pages. For an example of customizing the dialog box, see the *Custom Gantt Sample*.

See Also *Editing a Constraint Using the Dialog Box*

Editing a Resource Using the Predefined Dialog Box

IBM® ILOG® Gantt for .NET provides a dialog box for editing the properties of a resource. This dialog box is used to edit the basic information on a resource, such as the name.

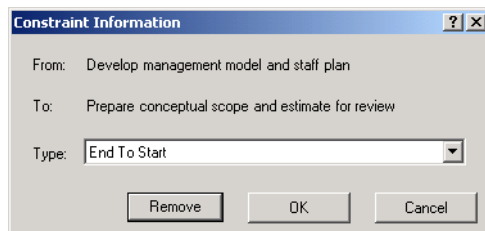


This dialog box has the following tab pages:

Tab Page	Description
General	To edit basic information on the resource, such as the name.
Working Times	To edit the calendar of the resource when the model is a Project Scheduling Model.
Notes	To edit notes on the resource.

Editing a Constraint Using the Dialog Box

IBM® ILOG® Gantt for .NET provides a predefined dialog box for editing a constraint. This dialog box is implemented by the class `ConstraintDialog`.



The `ConstraintDialog` class is used to change the type of constraint and to remove a constraint from the Gantt model. You can hide the Remove button by setting the `CanRemove` property to `false`.

To indicate which constraint is to be edited, use the `Constraint` property of the class.

Here is a small C# example that shows an event handler, which handles the Mouse-down event of an `ActivitySheet` instance and opens a constraint dialog when a constraint link is double-clicked:

```
private void OnActivitySheetMouseDown(object sender, MouseEventArgs e)
{
    ActivitySheet sheet = (ActivitySheet)sender;
    if (e.Clicks == 2)
    {
        GanttSheet.HitTestInfo info = sheet.HitTest(new Point(e.X, e.Y));
        if (info.Type == GanttSheet.HitTestType.Constraint)
        {
            constraintDialog.Constraint = info.Constraint;
            constraintDialog.ShowDialog(this);
        }
    }
    else

```

```
        {  
            // do something else  
        }  
    }  
}
```

See Also

Editing an Activity Using the Predefined Dialog Box

Printing Gantt Charts

The GanttChart and ScheduleChart classes are user interface controls designed to display projects on screen. You may need to have the projects printed on paper to distribute and exchange the projects. You may also need to print not only the visible part of the projects, but also the part that is not visible.

The Gantt package provides APIs that allow you to have the Gantt charts printed in a document as single or multiple pages.

In This Section

Introducing the GanttPrintDocument Class

Describes how to define a Gantt printable document and how to print it.

Customizing Printing

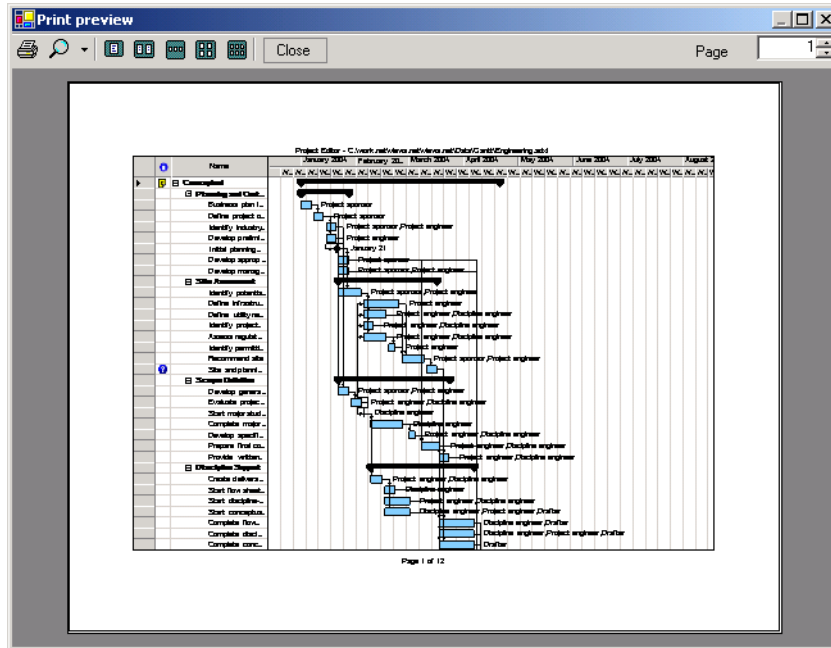
Describes the document page setup dialog box.

Introducing the GanttPrintDocument Class

To print the content of a Gantt Chart Control (GanttChart class), a Schedule Chart Control (ScheduleChart class), or a Reservation Chart Control (ReservationChart class) use the GanttPrintDocument class.

This class is defined in the ILOG.Views.Gantt.Windows.Forms.Printing namespace.

The **GanttPrintDocument** class extends the **PrintDocument** class of .NET Framework. This class allows you to use all the printing support of .NET Framework to print Gantt Chart content; for example, the Print preview control of .NET Framework.



You can create a **GanttPrintDocument** in the following way:

```
GanttChart chart = ...;
GanttPrintDocument document = new GanttPrintDocument(chart);
```

Here is a list of properties of **GanttPrintDocument** that you can customize for printing:

Property	Description
Page Settings	The size, orientation, and margins of a page.
Header and Footer	A text to add at the top or bottom of each page.
Number of Pages per Band	Gantt printing provides support for multipage printing through the PagesPerBand property. This property represents the number of pages you want printed between the start and the end date.
Repeat Table	Indicates whether the table should be printed repeatedly on every page.

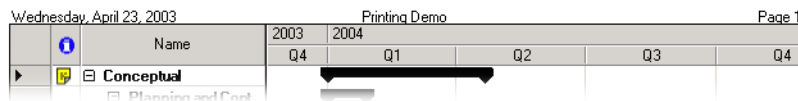
Start Date	The start date of the first printed page. This property defines the beginning of the printed data.
End Date	The end date for the last printed page in a band. This property defines the end of the printed data.
Number of Columns of the Table	Indicates the number of table columns to be printed.
Order of Page Numbering	Indicates whether pages are numbered on the right and then on the bottom or on the bottom and then on the right.

Note: All these properties are also accessible from the Gantt Page Setup dialog box described in Customizing Printing.

To manage the header and footer of pages, the ILOG.Views.Gantt.Windows.Forms.Printing namespace also contains two additional classes: the Header and Footer classes. The header and footer are common to all pages of a document and thus are set on an instance of the GanttPrintDocument class.

The **Header** and **Footer** classes are very similar. A header and a footer are defined by three text sections. Each section can have a specified font.

The following illustration shows an example of header.



Each of the three text sections of a header or footer can contain the text that you specify in the constructor of the object. For the header shown in the figure above it would be:

```
Header header =
    new Header("Wednesday, April 23, 2003", "Printing Demo", "Page 1");
```

Since the header and footer are defined on the document, you should not specify the page number as in the previous example. The **Header** and **Footer** classes provide a certain number of patterns that will be translated to values from the document when the document is printed.

Here is the list of patterns that you can use:

Pattern	Description
PagePattern	The pattern for the page number.
PagesPattern	The pattern for the number of pages in the document.
DatePattern	The pattern for the printing date.
TimePattern	The pattern for the printing time.
FileNamePattern	The pattern for the name of the file associated with the document.
DocumentPattern	The pattern for the name of the document.
AuthorPattern	The pattern for the name of the author of the document.

To create the header in the figure above:

```
Header header =
    new Header(HeaderFooter.DatePattern, "Printing Demo", "Page " +
HeaderFooter.PagePattern)
```

The following table summarizes the properties of GanttPrintDocument:

Property	Description	Default Value
DocumentName (inherited from PrintDocument)	The name of the document.	""
DefaultPageSettings (inherited from PrintDocument)	The paper size, margins, and orientation of pages.	
PrinterSettings (inherited from PrintDocument)	The printer that prints the document.	
Author	The name of the author of the document.	""
File	The name of the file that is printed.	""
Header	The header of each page.	null
Footer	The footer of each page.	null
PagesPerBand	The number of pages on which the time period is to be displayed.	2

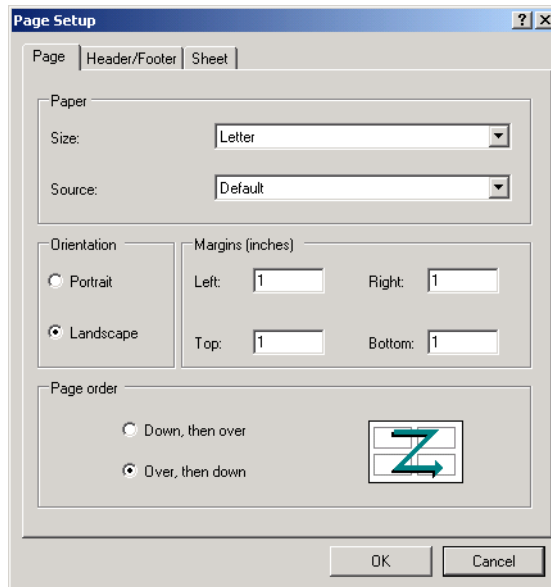
StartDate	The start-printing date.	The start date of the Gantt or schedule chart.
EndDate	The end-printing date.	The end date of the Gantt or schedule chart.
PrintAllTimeRange	Indicates whether the document chooses the printed time period automatically to print all the content of the chart.	true
RepeatTable	Indicates whether the table is printed on every page.	false
TableColumnCount	The number of columns of the table to print.	2
IsOverThenDownOrder	Indicates the page-numbering order	true

Customizing Printing

IBM® ILOG® Gantt for .NET provides a dialog box that allows you to customize the printing of a Gantt Chart or Schedule Chart control in a single window. This dialog box makes it possible to choose the paper size, orientation, and margins, just like the Windows® Forms **PageSetupDialog** class. In addition, you can specify the header and footer to be added to the chart and the properties that are specific to a Gantt Chart or Schedule Chart document, such as the start- and end-printing dates and the number of columns of the table to print.

The following illustration shows the document page setup dialog box.

Printing Gantt Charts



This dialog box is implemented by the class `GanttPageSetupDialog` in the `ILOG.Views.Windows.Forms.Printing` namespace. The properties of an instance of the `GanttPrintDocument` class can be edited in the dialog box.

To open the dialog box on a document you can do:

```
GanttPrintDocument document = new GanttPrintDocument(chart);
GanttPageSetupDialog pageSetupDialog = new GanttPageSetupDialog();
pageSetupDialog.Document = document;
pageSetupDialog.ShowDialog();
```

where `chart` is an instance of the `GanttChart`, `ScheduleChart` or `ReservationChart` class.

Storing and Displaying Working and Nonworking Times

The IBM® ILOG® Gantt for .NET library defines several classes that allow you to store, edit, and display the working and nonworking times. The following classes are involved:

- ◆ The **WorkCalendar** class allows you to store the working and nonworking times as well as exception periods.
- ◆ The **WorkingTimesEditor** is an editor that allows you to edit the content of a **WorkCalendar**.
- ◆ The **WorkingTimesDialog** is a dialog box that allows you to edit the content of a **WorkCalendar** (using a **WorkingTimesEditor**).
- ◆ The **WorkingTimesGrid** is a grid of the Gantt Chart that displays the working and nonworking periods in the background of a Gantt chart.

***Note:** The **WorkCalendar** is different from the **CalendarView** control that displays the content of a Gantt data model in a calendar format. To learn more about the **CalendarView** control see *Displaying Activities Using a Calendar View*.*

In This Section

*Using a **WorkCalendar** to Store Working and Nonworking Periods*

Storing and Displaying Working and Nonworking Times

Explains how to create a **WorkCalendar** object to store working and nonworking periods.

Navigating in a WorkCalendar

Explains how to navigate in working and nonworking periods in a **WorkCalendar**.

Editing the Content of a WorkCalendar

Explains the panels and dialog boxes that allow you to edit the content of a **WorkCalendar**.

Displaying Working and Nonworking Times in Gantt Controls

Explains how to use the **WorkingTimesGrid** to display the nonworking times in the Gantt controls.

Using a WorkCalendar to Store Working and Nonworking Periods

A **WorkCalendar** object is used to store working and nonworking times. IBM® ILOG® Gantt for .NET defines two types of **WorkCalendar**: a **WorkCalendar** can be a base calendar or a subcalendar that inherits from a base calendar. A subcalendar will inherit its specification from its base calendar and may modify the working and nonworking times specified by the base calendar. For example, a base calendar can be used to store the general working times on a project and a subcalendar can be used to specify the working times and vacations of a specific resource of the project.

To create a base calendar named `MyCalendar`, use the following C# code:

```
WorkCalendar myBaseCalendar = new WorkCalendar("MyCalendar", null);
```

To create a subcalendar of this calendar proceed as follow:

```
WorkCalendar mySubCalendar = new WorkCalendar("MySubCalendar", myBaseCalendar);
```

By default a **WorkCalendar** defines Saturday and Sunday as nonworking days, for the other days the working times are from 8 AM to 12 AM and from 1 PM to 5 PM defining 8 working hours per day.

The **WorkCalendar** class provides methods that allow you to modify these settings for a day of the week.

To set all Fridays as nonworking days you would do:

```
mySubCalendar.SetNonWorking(DayOfWeek.Friday);
```

To change the working times of all Mondays to be 7 AM - 1 PM and 3 PM - 8 PM you would do:

```
WorkingTime[] times = new WorkingTime[2];  
times[0] = new WorkingTime(TimeSpan.FromHours(7), TimeSpan.FromHours(13));
```

```
times[1] = new WorkingTime(TimeSpan.FromHours(15), TimeSpan.FromHours(20));
mySubCalendar.SetWorkingTimes(DayOfWeek.Monday, times);
```

You can also specify exceptional periods:

To specify a nonworking period from 01/01/2005 to 01/07/2005 you would do:

```
mySubCalendar.SetNonWorking(new DateTime(2005,1,1), new DateTime(2005, 1,7));
```

To specify special working hours from 7 AM to 9 AM for the period 06/01/2005 to 06/06/2005 you would do:

```
WorkingTime[] times = new WorkingTime[1];
times[0] = new WorkingTime(TimeSpan.FromHours(7), TimeSpan.FromHours(9));
mySubCalendar.SetWorkingTimes(new DateTime(2005,6,1), new DateTime(2005, 6,6),
times);
```

Navigating in a WorkCalendar

A `WorkCalendar` object does not only allow you to store the working and nonworking times. It also allows you to navigate in the working and nonworking times and perform several computation on working periods.

For example, you can get the next or previous working time from a date. Here is an example:

the following C# code computes the next and previous working time from January 1 2005.

```
WorkCalendar calendar = new WorkCalendar("MyCalendar", null);
DateTime date = new DateTime(2005,1,1);

DateTime next = calendar.NextWorkingTime(date);
DateTime previous = calendar.PreviousWorkingTime(date);

Console.WriteLine("Next working time is " + next);
Console.WriteLine("Previous working time is " + previous);
```

This code gives you Monday January, 3, 2005 at 8 AM as the next working time and Friday December, 31 2004 at 5 PM as the previous working time.

Similarly, the `WorkCalendar` defines methods to navigate to next nonworking times (`NextNonWorkingTime`).

You may also compute the amount of work between two dates:

```
DateTime fromDate = new DateTime(2005,1,5);
DateTime toDate = new DateTime(2005,1,12);

TimeSpan work = calendar.WorkBetween(fromDate, toDate);
```

This fragment of C# code returns a duration of 40 hours. In the period from 05 to 12 January 2005, there are 5 working days of 8 hours, resulting in 40 hours of work.

Storing and Displaying Working and Nonworking Times

Finally, the **WorkCalendar** object provides methods to add or remove a work duration from a date to compute another date. The following C# code illustrates how to add or remove a period of work to and from a date:

```
TimeSpan work = TimeSpan.FromDays(88);
DateTime date = new DateTime(2004, 1, 1, 8, 0, 0);
Console.WriteLine(date);
date = calendar.Add(date, work);
Console.WriteLine(date);
date = calendar.Remove(date, work);
Console.WriteLine(date);
```

This C# code fragment will print:

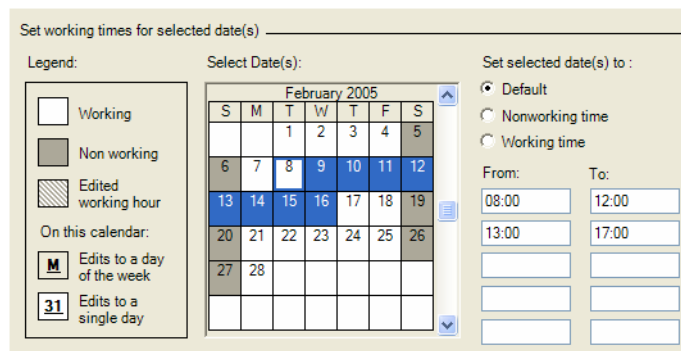
```
1/1/2004 8:00:00 AM
1/4/2005 5:00:00 PM
1/1/2004 8:00:00 AM
```

Editing the Content of a WorkCalendar

The content of a **WorkCalendar** can be modified by code. However, IBM® ILOG® Gantt for .NET class library also provides classes that allow a final user of your application to modify the content of a **WorkCalendar**: the **WorkingTimesDialog** and **WorkingTimesEditor** classes.

The **WorkingTimesEditor** and **WorkingTimesDialog** are very similar. The **WorkingTimesEditor** is a panel that you can use inside your own dialog box, and the **WorkingTimesDialog** is a dialog box that simply contains a **WorkingTimesEditor**.

The figure below represents a **WorkingTimesEditor**.



Here is a C# code fragment that shows how to use a **WorkingTimesDialog**.

```
WorkCalendar calendar = new WorkCalendar();

WorkingTimesDialog dialog = new WorkingTimesDialog();
```



```

dialog.Calendar = calendar;
if (dialog.ShowDialog() == DialogResult.OK) {
    // OK was chosen, validates the modifications.
    dialog.ApplyCalendarModification();
}

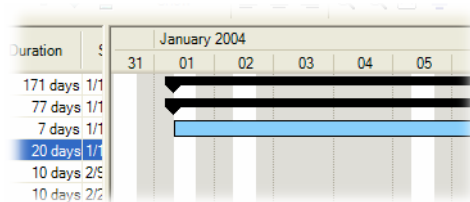
```

Displaying Working and Nonworking Times in Gantt Controls

The nonworking periods defined by an instance of the `WorkCalendar` class can be displayed in the Gantt controls by means of a grid. The grid class displaying the content of a **WorkCalendar** is defined by the `WorkingTimesGrid` class.

When creating a `GanttChart`, a `ScheduleChart` or a `LoadChart` control, these controls already contain an instance of **WorkingTimesGrid** in their collection of grids. By default, this grid displays the content of the standard calendar.

In the following figure, behind the Gantt bars you can see the nonworking period displayed in gray:



Note: The `WorkingTimesGrid` is always up-to-date with the calendar that it displays. When the calendar changes, the grid is automatically refreshed.

The following C# code fragment shows how to change the calendar displayed by the grid of a `GanttChart` control.

```

WorkCalendar myCalendar = new WorkCalendar();
GanttChart gantt = new GanttChart();
WorkingTimesGrid grid = (WorkingTimesGrid)gantt.TimeGrids[1];
grid.Calendar = myCalendar;

```

Note: This code assumes that a `GanttChart` control has by default two time grids, the second one being the `WorkingTimesGrid`.

Storing and Displaying Working and Nonworking Times

This mechanism is the same for all the Gantt controls displaying time grids, such as the **ScheduleChart**, **GanttSheet**, **LoadChart** classes and so on.

You can also display the nonworking days in a CalendarView control. For this control, you can simply use the **Calendar** property. Only the days defined as nonworking will be displayed in a different color.

Creating Project Scheduling Applications with IBM ILOG Gantt for .NET

IBM® ILOG® Gantt for .NET allows you to create applications that require project scheduling capabilities through a specific Gantt Data Model class: the `ProjectSchedulingModel`.

The **`ProjectSchedulingModel`** class stores information about your project and uses this information to calculate and maintain the schedule of your project. The **`ProjectSchedulingModel`** computes the schedule immediately. As soon as you have entered information about your project, you can learn about the scheduled start date of activities and the project target date.

In the `<install-dir>\Samples\Applications\GanttEditor` directory, you will find a fully-featured application sample that shows how to use the **`ProjectSchedulingModel`** in a real project scheduling application.

The new option, IBM ILOG Gantt for .NET - Project Management, provides a data model and algorithms for scheduling projects, leveling resources, and computing the critical paths. This option is particularly interesting for developing rapidly project management solutions that can be deployed as desktop applications or over the Web.

Specific licenses are required for developing and deploying applications that embed IBM ILOG Gantt for .NET - Project Management. When the ILM license key does not allow this option, the Gantt chart displays a warning message that indicates the license violation.

In This Section

Programming with the ProjectSchedulingModel

Explains classes involved in the project scheduling Gantt model.

How the Project Scheduling Model Computes the Schedule of a Project

Explains how the schedule of a project is computed.

Resource Leveling in ProjectSchedulingModel

Explains how the project scheduling model removes resources overallocations.

Displaying the Critical Path of a Project Scheduling Model

Explains how to display the critical path of the project in a Gantt Chart.

Saving and Reading a ProjectSchedulingModel to an XML File

Explains how to read and write a project to XML.

Programming with the ProjectSchedulingModel

IBM® ILOG® Gantt for .NET provides a specific data model class that allows you to create project scheduling applications: the ProjectSchedulingModel class.

Like the SimpleGanttModel class, the ProjectSchedulingModel class is an implementation of the IGanttModel interface and both classes hold data that can be displayed in the Gantt controls of IBM ILOG Gantt for .NET library. The **ProjectSchedulingModel** adds project scheduling capabilities to the **SimpleGanttModel** and thus should be used for project scheduling applications.

In This Section

The ProjectSchedulingModel Class

Introduces the **ProjectSchedulingModel** class.

Activities in the ProjectSchedulingModel

Introduces the SchedulingActivity class.

Controlling When the Project Schedule is Recomputed

Explains how to control when the schedule of the project is recomputed.

Displaying and Editing Content of a ProjectSchedulingModel in a Gantt Control

Explains how to display and edit a **ProjectSchedulingModel** in Gantt controls.

See Also

How the Project Scheduling Model Computes the Schedule of a Project | Resource Leveling in ProjectSchedulingModel

The ProjectSchedulingModel Class

The ProjectSchedulingModel class is an extension of the SimpleGanttModel class that adds project scheduling capabilities. Just like the **SimpleGanttModel** described in the section *Introducing the Gantt Data Model In-Memory Implementation*, the **ProjectSchedulingModel** defines the scheduling information that can be edited and displayed by any WinForms and WebForms controls of the IBM® ILOG® Gantt for .NET library.

For adding, removing and accessing activities, resources, constraints and reservations, the **ProjectSchedulingModel** works the same way as the **SimpleGanttModel**, but for each modification of the model, the **ProjectSchedulingModel** re-computes a working schedule of the project. For example, when if you change the duration of an activity in the model, the model re-computes a schedule and this may have impact on the start time of all the successors activities and on the project scheduled end date.

The following types are involved in the Project Scheduling Data model:

Type	Description
ProjectSchedulingModel	An implementation of IGanttModel that defines the project scheduling data model that uses the classes listed below.
SchedulingActivity	Represents an activity or a task that must be completed in the project.
SchedulingResource	Represents a resource that can be allocated to an activity to make its completion possible.
SchedulingConstraint	Represents an activity-to-activity scheduling constraint.
SchedulingReservation	Represents the allocation of a resource to an activity.

When creating a **ProjectSchedulingModel**, you must specify a starting date for the project; this is done through the StartDate property of the **ProjectSchedulingModel** class. The project is scheduled from the start date, so activities that do not have predecessors or other scheduling constraints will be scheduled at the project start date. The project end date will be automatically computed and is available through the EndDate property.

By default, the **ProjectSchedulingModel** schedules activities using the working times on the project calendar (see WorkCalendar class). The project calendar can be specified using the Calendar property. When creating a **ProjectSchedulingModel**, the model has a standard calendar that defines Saturday and Sunday as nonworking days and working times from

8AM to 12AM and from 1PM to 5 PM. Specific calendars can also be specified for resources and activities, read the section *Calendars in the Project Scheduling Model* for more details.

The following C# code fragment creates a **ProjectSchedulingModel** and adds one activity. Finally, the scheduled start time of the activity and the project target date are displayed.

```
ProjectSchedulingModel project = new ProjectSchedulingModel();
project.StartDate = new DateTime(2005, 1, 1);
SchedulingActivity activity = project.NewActivity() as SchedulingActivity;
activity.Duration = TimeSpan.FromHours(8);
project.Activities.Add(activity);

Console.WriteLine("scheduled start time : " + activity.StartTime);
Console.WriteLine("project's target date : " + project.EndDate);
```

This C# code prints the following information on the console:

```
scheduled start time : 1/3/2005 8:00:00 AM
project's target date : 1/3/2005 5:00:00 PM
```

This shows that the activity has been scheduled to start on January 3, 2005 at 8 AM, even though the project start date is January 1, this is because January 1, 2005 is a Saturday.

Activities in the ProjectSchedulingModel

Activities in the **ProjectSchedulingModel** are defined by the **SchedulingActivity** class. Since the **ProjectSchedulingModel** automatically computes the schedule of the activity you should not specify a start date for activities. For an activity, it is only mandatory to specify a duration. The schedule of activities will be computed by the model.

The **Duration** property of the activity represents the amount of work needed to complete the activity. For example, if you are using the default standard calendar, a duration of 8 hours represents in fact one day of elapsed duration since in the default calendar each working day has 8 hour of work (The default calendar has working times from 8 AM to 12PM and from 1PM to 5PM).

Here is a small piece of C# code that creates a **ProjectSchedulingModel** with a start date of January 5, 2005 and adds an activity with duration of 8 hours.

```
ProjectSchedulingModel model = new ProjectSchedulingModel();
model.StartDate = new DateTime(2005,1,5);
SchedulingActivity activity = model.NewActivity() as SchedulingActivity;
activity.Duration = TimeSpan.FromHours(8);
model.Activities.Add(activity);
```

Once the activity is added in the model, the following properties of the **SchedulingActivity** class will be computed by the **ProjectSchedulingModel**:

Property Name	Property Type	Description
StartTime	DateTime	The scheduled start time of the activity.
EndTime	DateTime	The scheduled end time of the activity.
EarlyStart	DateTime	The earliest date the activity can start based on the predecessor and successors of the activity and other scheduling constraints.
EarlyFinish	DateTime	The earliest date the activity can finish based on the predecessors and successors of the activity and other scheduling constraints.
LateStart	DateTime	The latest date the activity can start based on the predecessors and successors of the activity and other scheduling constraints.
LateFinish	DateTime	The latest date the activity can finish based on the predecessors and successors of the activity and other scheduling constraints.
TotalSlack	TimeSpan	The amount of time the activity can be delayed without delaying the project's end date.
FreeSlack	TimeSpan	The amount of time the activity can be delayed without delaying any successor activity.
Critical	Boolean	Indicates whether the activity is critical or not.

Note: You should not specify the **StartTime** or **EndTime** properties of a **SchedulingActivity**. Those properties are automatically computed. If you set the value of the **StartTime** property, a constraint of type "Start No Earlier Than" will be set on the activity. If you specify the **EndTime** property, a constraint of type "Finish No Earlier Than" will be set on the activity.

To learn more about constraints and other properties of **SchedulingActivity** that allow you to control the schedule of an activity and how the schedule of the project is computed see the section *How the Project Scheduling Model Computes the Schedule of a Project*.

When the automatic resource leveling is turned on, the following properties of the **SchedulingActivity** will also be computed:

Property Name	Property Type	Description
PreleveledStart	DateTime	Represents the start time of the activity before the resource leveling was computed.
PreleveledEnd	DateTime	Represents the end time of the activity before the resource leveling was computed.
LevelingDelay	TimeSpan	The amount of time the activity is delayed from its early start date (EarlyStart) in order to remove resource overallocations.

To learn more about resource leveling read the section *Resource Leveling in ProjectSchedulingModel*.

You can also specify that an activity has already started by specifying an actual start time (**SchedulingActivity.ActualStartTime** property) and the percentage of completion (**SchedulingActivity.WorkComplete** property). That information is taken into account by the model when scheduling the activity. Activities that have already started will always be scheduled at their actual start time and will not be delayed by the resource leveling algorithm.

Controlling When the Project Schedule is Recomputed

Since every modification of the model leads to a re-computation of the schedule of the project, it is important to be able to control when the schedule is re-computed in order to avoid unnecessary re-computations.

For example, if you want to change the duration of several activities, you do not want the model to re-compute the schedule several times.

The *ProjectSchedulingModel* provides the *BeginScheduleSession* and *EndScheduleSession* methods that allow you to group several modifications so that only one single schedule will be computed.

Here is the typical C# code that you would write:

```
try {
    model.BeginScheduleSession();

    ... change several things in the model...
} finally {
    model.EndScheduleSession();
}
```

The **ProjectSchedulingModel** class also provides two events, the `BeginSchedule` and `EndSchedule` events, that are fired when the scheduling starts and finishes.

Control

Displaying and Editing Content of a ProjectSchedulingModel in a Gantt

Since the `ProjectSchedulingModel` is an implementation of the `IGanttModel` interface, it can be displayed in all the WinForms and WebForms controls provided with IBM ILOG Gantt for .NET: `GanttChart`, `ScheduleChart`, `CalendarView`, and so on.

This section gives some specific information that you might need to know when using a **ProjectSchedulingModel** with the Gantt controls.

Note: In the `<install-dir>\Samples\Applications\GanttEditor` directory, you will find a fully-featured application sample that shows how to use the **ProjectSchedulingModel** and almost all the WinForms controls of IBM ILOG Gantt for .NET.

Displaying the Working Times in the Grid of a Gantt Control

As described in *Using Time Grids and Date Indicators*, the Gantt chart controls that display time-based information, such as the `GanttChart` control, can display a grid that represents working and nonworking times. The displayed grid is an instance of the `WorkingTimesGrid` class and can display the working and nonworking times defined in a particular calendar (class `WorkCalendar`).

When using a **ProjectSchedulingModel**, several calendars may be involved: the project calendar, a resource calendar or an activity calendar. You can decide which calendar to display in the grid of the Gantt control.

Here is a small example that shows how to change the calendar displayed in a **GanttChart** control to the calendar of a specific resource:

```
void ChangeDisplayedCalendar(GanttChart ganttChart, SchedulingResource resource)
{
    foreach (TimeGrid grid in ganttChart.TimeGrids)
        if (grid is WorkingTimesGrid)
            ((WorkingTimesGrid)grid).Calendar = resource.Calendar;
}
```

For more information on calendars in the **ProjectSchedulingModel**, see *Calendars in the Project Scheduling Model*.

Specifying How Work Durations are Entered and Displayed

For the properties that represent a duration of work, such as the **Duration** or **TotalSlack** property of the **SchedulingActivity** class, you can control how they are displayed and entered in the controls.

The **MinutesPerDay** property of the **ProjectSchedulingModel** class represents the number of minutes of work in one day. The default value is 480 minutes (8 hours). When you enter 1 day for the **Duration** property in the Gantt table, 8 hours will be set to the **Duration** property.

Similarly, the **MinutesPerWeek** property represents the number of minutes of work in a week; the default value is 2400 minutes (40 hours). If you enter 1 week for the duration of an activity in the Gantt table, 40 hours will be set the to **Duration** property.

Finally, the **DaysPerMonth** property works the same as the two other properties, the default value is 20. When you enter a duration of 1 month in a Gantt table, the result will be a value of 20***MinutesPerDay** minutes in the **Duration** property of the activity.

How the Project Scheduling Model Computes the Schedule of a Project

As you build a project plan, the **ProjectSchedulingModel** class calculates and creates a working schedule based on information you provide about the activities to be done and the people who work on them.

The **ProjectSchedulingModel** schedules the project from the information you specify about the project itself: the individual activities (class **SchedulingActivity**) required to complete the project and, if necessary, the resources (class **SchedulingResource**) needed to complete those activities. If anything about your project changes after you create your schedule, you can update the activities or resources and the **ProjectSchedulingModel** adjusts the schedule for you.

For each activity, you enter duration, dependencies between activities (class `SchedulingConstraint`), and activity constraints (type `ActivityConstraintType`), then the **ProjectSchedulingModel** calculates the start and end date for each activity. You can enter resources in your project and then assign them to activities to indicate which resource is responsible for completing each activity (class `SchedulingReservation`). If you enter resources, task schedules are further refined according to resource availability and working times entered on calendars (class `WorkCalendar`). Other elements, such as lead time and lag time on dependencies can affect the scheduling. Understanding the effects of these elements can help you maintain and adjust the project schedule.

In This section

How Project Start Date Affects the Schedule

Explains the `StartDate` property of the **ProjectSchedulingModel**.

How Constraint Links Affect the Schedule

Explains the impact of **SchedulingConstraint** on the schedule.

How Constraints on Activities Affect the Schedule

Explains specific constraints for activities.

Calendars in the Project Scheduling Model

Explains how calendars affect the schedule.

See Also

Programming with the ProjectSchedulingModel

How Project Start Date Affects the Schedule

The `ProjectSchedulingModel` schedules your project based upon the project start date. The start date of the project can be specified by the `StartDate` property of the **ProjectSchedulingModel**. When adding an activity in the model, the activity is initially scheduled at the project start date. Later on, if you add predecessors to the activity or if you set other constraints on the activity, the schedule start time of the activity will be computed accordingly. The project end date is computed like the latest end date of all activities in the model and can be retrieved using the `EndDate` property of the **ProjectSchedulingModel** class.

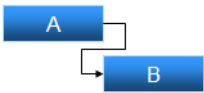

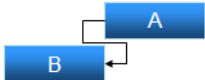

You may change the project start date at any time and the **ProjectSchedulingModel** will recompute a new schedule based on this new start date.

How Constraint Links Affect the Schedule

When a new activity is inserted in the `ProjectSchedulingModel`, the activity has no predecessor constraint and will be scheduled to start at the project start date.

Later on, predecessor constraints can be added between activities using the `SchedulingConstraint` class that defines precedence constraints between activities.

The different types of precedence constraints defined by the `ConstraintType` enumeration are the following:

Predecessor Constraint Type	Example	Description
EndToStart		The activity B cannot start until the activity A is finished.
StartToStart		The activity B cannot start until the activity A is started.
StartToEnd		The activity B cannot finish until the activity A starts.
EndToEnd		The activity B cannot finish until the activity A finishes.

Lead and Lag Time

The `SchedulingConstraint` class also allows you to specify a lead or a lag time for the constraint.

A lag time is a delay between the end of an activity and the start of its successor, a lead time is an overlap between the two activities, so that the successor starts before the end of the predecessor.



To specify a lag or lead time in the **SchedulingConstraint** use the **Lag** and **LagFormat** property of the **SchedulingConstraint** class. The lead and lag time can be expressed in various formats: it can be defined as a work duration or an elapsed duration. The duration may be directly specified or defined as a percentage of the duration of the predecessor activity. For example, to create an EndToStart constraint between activity A and B with a lag expressed as 50% of the elapsed duration of A (assuming that the variable *a* and *b* are instances of the **SchedulingActivity** class), you would do:

```
SchedulingConstraint c = model.NewConstraint(a, b, ConstraintType.EndToStart);
c.Lag = 50;
c.LagFormat = LagFormat.EllapsedPercentage;
model.Constraints.Add(c);
```

To express a lead time, you would give a negative value to the **Lag** property.

How Constraints on Activities Affect the Schedule

The schedule start date of an activity is mainly controlled by the dependencies between activities, as explained in *How Constraint Links Affect the Schedule*.

However, the scheduled start date of an activity can also be controlled using constraints on activity. Constraints on activities are set and retrieved through the **Constraint** property of the **SchedulingActivity** class. The different types of constraints that can be specified on an activity are defined by the **ActivityConstraintType** enumeration. By default when creating an activity, the constraint on the activity is set to **AsSoonAsPossible**. This means that the activity will be scheduled as soon as possible. For example, if the activity has several end-to-start predecessors, the task will be scheduled as soon as the predecessors are finished.

For the **AsSoonAsPossible** and **AsLateAsPossible** constraints, it is not necessary to specify a constraint date. For the other types of constraints, a constraint date must be specified in the **ConstraintDate** property of the **SchedulingActivity** class.

Here are the different types of constraints and how they affect the schedule of an activity:

Constraint type	Description
AsSoonAsPossible	The ProjectSchedulingModel schedules the activity to start as soon as it can. This is the default constraint type. No specific date constraint is added to the activity. Activities with this constraint that have no predecessors will be scheduled at the project start date.
AsLateAsPossible	The ProjectSchedulingModel schedules the task to start as late as it can, based on the constraints on predecessors. No specific date constraint is added to the activity.
FinishNoLaterThan	The ProjectSchedulingModel schedules the task to finish no later than the date specified by the ConstraintDate property of the activity. The activity can be scheduled to finish before or at the specified date. With such a constraint, the activity may be scheduled to start before the date allowed by predecessor constraints and thus violate the constraint imposed by predecessors.
StartNoLaterThan	The ProjectSchedulingModel schedules the task to start no later than the date specified by the ConstraintDate property of the activity. The activity can be scheduled to start before or at the specified date. With such a constraint, the activity may be scheduled to start before the date allowed by predecessor constraints and thus violate the constraint imposed by predecessors.
StartNoEarlierThan	The ProjectSchedulingModel schedules the task to start no earlier than the date specified by the ConstraintDate property of the activity. The activity can be scheduled to start after or at the specified date. Such a constraint will be automatically set on the activity, when you modify the StartTime property of the activity.
FinishNoEarlierThan	The ProjectSchedulingModel schedules the task to finish no earlier than the date specified by the ConstraintDate property of the activity. The activity can be schedule to finish after or at the specified date. Such a constraint will be automatically set on the activity, when you modify the EndTime property of the activity.

Constraint type	Description
StartOn	The ProjectSchedulingModel schedules the task to start at the date specified by the ConstraintDate property of the activity without taking into account other scheduling constraints.
FinishOn	The ProjectSchedulingModel schedules the task to end at the date specified by the ConstraintDate property of the activity without taking into account other scheduling constraints.

Calendars in the Project Scheduling Model

The **ProjectSchedulingModel** uses calendars to define the working and nonworking periods of the project such as holidays and weekends. Calendars are defined by the **WorkCalendar** class.

The **ProjectSchedulingModel** defines several types of calendars:

- ◆ The project calendar defines the default working and nonworking times for the project and can be set using the **Calendar** property of the **ProjectSchedulingModel**. If no calendar is defined for resources working on an activity or for the activity itself, an activity will be scheduled within the working times defined by the project calendar.
- ◆ The resource calendar defines specific working and nonworking periods for a resource and can be retrieved using the **Calendar** property of the **SchedulingResource** class. The work assigned to the resource will be scheduled within the working time of the resource calendar.

*Note: Resource calendar only apply to work resource not material resource (see the **SchedulingResource.Type** property).*

- ◆ The activity calendar defines a specific calendar for an activity. This calendar can be retrieved using the **Calendar** property of the **SchedulingActivity** class. When a calendar is assigned to an activity, the activity will be scheduled within the working times of this calendar without taking into consideration the calendar of resources that may be assigned to the activity.

When creating a **ProjectSchedulingModel**, the model has a standard calendar that defines Saturday and Sunday as nonworking days and working times from 8AM to 12AM and from 1PM to 5 PM.

*Note: The **ProjectSchedulingModel** holds a list of base calendars in its **BaseCalendars** property. This collection contains the base calendars that can be used for the project calendar or for activity calendars. The resource calendars must be a subcalendar of one of the calendars in the base calendars collection.*

To learn more about Calendars see *Storing and Displaying Working and Nonworking Times*.

Resource Leveling in **ProjectSchedulingModel**

Resource leveling is the process of removing overallocation of resources. A resource is overallocated when too much work is assigned to it.

The **ProjectSchedulingModel** can remove overallocations automatically by delaying activities so that resources have enough time to work on the activity. When delaying, the **ProjectSchedulingModel** ensures that all the scheduling constraints are still valid. The resource leveling process may delay the project end date.

Overallocations of resource can be removed automatically by the resource leveling algorithm of the **ProjectSchedulingModel** and can also be done manually by specifying delays for each activity. To turn on or off the automatic resource leveling of the model use the **AutomaticResourceLeveling** property of the **ProjectSchedulingModel** class.

When the resource leveling is automatic, the **ProjectSchedulingModel** uses a heuristic to determine which activity should be delayed first. This heuristic examines the following properties of the activity:

- ◆ late start
- ◆ total slack
- ◆ duration
- ◆ priority
- ◆ ID

The delays computed automatically are stored in the **LevelingDelay** property of the **SchedulingActivity** class.

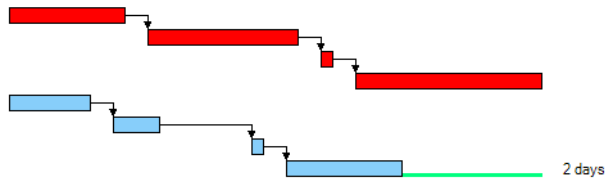
The automatic resource leveling can be started using no leveling delays or can use the current values stored in the **LevelingDelay** property of each activity. This is controlled by the **ClearDelaysBeforeLeveling** property of the **ProjectSchedulingModel** class. When the **ClearDelaysBeforeLeveling** property is set to `true`, all the delays will be reset to zero before the resource leveling starts.

The resource leveling process may be time consuming, that is why the leveling algorithm fires events about the progress of the algorithm so that some feedback can be given to a final user. This is done through the `LevelingProgress` event of the **ProjectSchedulingModel** class.

In some cases the resource leveling algorithm may not be able to find a solution that removes all the resource overallocation. For example, if two activities that are causing an overallocation for a resource have a "StartOn" constraint, they cannot be delayed and the overallocation cannot be removed. In this case, the **LevelingProgress** event also gives information about which resource is problematic at which date and allows continuing or stopping the algorithm.

Displaying the Critical Path of a Project Scheduling Model

The critical path is the set of activities that are critical. In some cases this set of activities draws a path from the beginning to the end of the project. An activity is considered to be critical if any delay of this activity will delay the project end date.



In the project displayed above, the first 4 activities are critical. None of this activity has slack and therefore the sequence drives the project end date, while the last 4 activities have slack, each activity can be delayed by 2 days before the project end date is affected. These are not critical.

The `ProjectSchedulingModel` computes for each activity the slack available for this activity before the project end date is delayed. This information is stored in the `TotalSlack` property of the `SchedulingActivity` class. When the activity has no slack, the activity is considered as critical and the `Critical` property is set to `true`. However, you can decide when an activity becomes critical by changing the `CriticalSlackThreshold` property of the **ProjectSchedulingModel** class. This can be useful if you need to be alerted before a task becomes really critical.

The **TotalSlack** of each activity is based on the comparison of the `EarlyStart`, `EarlyFinish`, `LateStart` and `LateFinish` dates of the activity, each of these depending on dates of predecessor and successor activities. The **EarlyStart** of an activity is the earliest date this activity can start based on predecessor links and other scheduling constraints. The


EarlyFinish is the earliest date the activity can finish, the **LateStart** is the latest date the activity can start and the **LateFinish** the latest date the activity can finish.

The **TotalSlack** is computed as the minimum value between the **LateFinish** minus the **EarlyFinish**, and the **LateStart** minus the **EarlyStart** dates.

The **ProjectSchedulingModel** also computes the FreeSlack property. The **FreeSlack** is the delay that can be applied to the activity before the activity delays one of its successor activity.

In order to display the critical path in the GanttChart control you can simply use two different bar styles: one bar style for noncritical activities and another one for critical activities where for example bars will be displayed as red:



Name	Appearance	Style For ...	Start	End
Activity		Normal && !Critical	StartTime	EndTime
Critical Activity		Normal && Critical	StartTime	EndTime
Progress		Normal	StartTime	CompletedThru

```
ActivityBarStyle normalBarStyle = new ActivityBarStyle();
normalBarStyle.FromProperty = "StartTime";
normalBarStyle.ToProperty = "EndTime";
normalBarStyle.Bar.BorderColor = Color.Blue;
normalBarStyle.Bar.Color = Color.Blue;
normalBarStyle.Name = "Activity";
normalBarStyle.StyleFor = "Normal && !Critical";
```

```
ActivityBarStyle criticalBarStyle = new ActivityBarStyle();
criticalBarStyle.FromProperty = "StartTime";
criticalBarStyle.ToProperty = "EndTime";
criticalBarStyle.Bar.BorderColor = Color.Red;
criticalBarStyle.Bar.Color = Color.Red;
criticalBarStyle.Name = "Critical Activity";
criticalBarStyle.StyleFor = "Normal && Critical";
```

You can find an example of a displayed critical path in the GanttEditor sample located in the `<install-dir>\Samples\Applications\GanttEditor` directory.

For more information on activity bar style see *Representing Activity Bars in Gantt Sheets*.

Saving and Reading a ProjectSchedulingModel to an XML File

In order to save and read the content of a ProjectSchedulingModel to and from an XML file, you will use the ProjectSchedulingModelSerializer class. This class extends the GanttModelXmlSerializer class in order to store information that is specific to the ProjectSchedulingModel class such as the start date of the project or calendars.

Since the **ProjectSchedulingModelSerializer** is a subclass of the **GanttModelXmlSerializer** you can follow the procedures described in *Reading and Writing Scheduling Data Using XML* to read and write data or to extend it.

*Note: The **ProjectSchedulingModelSerializer** can also read XML files generated by the **GanttModelXmlSerializer**, but since the XML files generated by the **GanttModelXmlSerializer** do not contain all the information required to correctly fill the **ProjectSchedulingModel**, some information will be computed by the reader. In this case, the model consider that the calendar is the standard calendar; the start date of the model is computed as the smallest start time of all activities, the work duration of activities are computed from the start and end time of activities using the standard calendar.*

Localizing a Gantt Application

IBM® ILOG® Gantt for .NET is internationalized. All messages, resources and dialog boxes of IBM ILOG Gantt for .NET are localized for English and French language. If you need to localize for another language, IBM ILOG Gantt for .NET provides the resource files and tools that will allow you to localize the library for a particular culture. The library uses the culture information that you have specified in the Control Panel to display dates and numbers. All the controls of IBM ILOG Gantt for .NET can also be used in right-to-left mode for Arabic and other languages that are written right-to-left.

In order to create a localized version of IBM ILOG Gantt for .NET you must create assemblies (dlls) that contain the culture-dependant resources of the library. Those assemblies are called *satellite assemblies*. IBM ILOG Gantt for .NET provides the tools that will help you create satellite assemblies for a particular culture.

In This Section

Creating a Localization Project

Explains how to use the localization tool.

Translating the Resource Files

Describes the different types of resource files and explains how to translate them.

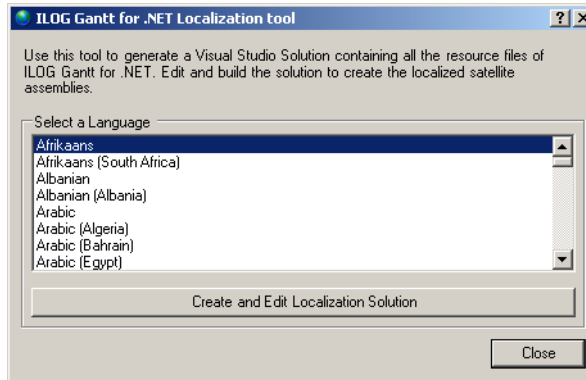
Creating the Satellite Assemblies

Explains how to create the satellite assemblies.

Creating a Localization Project

The IBM® ILOG® Gantt for .NET localization tool allows you to start localizing the library for a particular culture. The `LocalizationTool.exe` file can be found in the directory `<install-dir>\bin`.

Launch the `LocalizationTool.exe` and choose a language for the localization.



When you press the button `Create and Edit Localization Solution`, the tool opens a new Visual Studio .NET solution.

The tool creates a new solution for building the satellite assemblies and creates also a new directory with a copy of all the resource files that you may need to localize. This new directory is called: `<install-dir>\localization\Resources-<culture name>`.

For example, if you choose to localize for Spanish, a new directory named `resources-es` will be created. This new directory contains a copy of all the resource files of IBM ILOG Gantt for .NET and a solution named `Localization-es.sln`.

The solution created by the Localization tool contains a project for each of the IBM ILOG Gantt for .NET satellite assemblies. Each project contains the resource files (`.resx` files) that need to be translated for the culture you have chosen.

Translating the Resource Files

The projects contain two types of resource files:

- ◆ resource files that contain global resource for the library. Those files are named `Resources.resx`.

- ◆ resource files that correspond to a predefined dialog box of IBM ILOG Gantt for .NET. For example, the `ActivityDialog.resx` file corresponds to the ActivityDialog dialog box.

You can use Visual Studio .NET to open the resource files and do the translation, but you can also use the .NET Framework tool `winres.exe` to do the translation of the dialog box resources. Refer to the .NET framework documentation for more information about `winres.exe`.

Creating the Satellite Assemblies

When you have modified the resource files for your culture, you can build the solution created by the localization tool. When you build the solution, you create the satellite assemblies in the `<install-dir>\bin\<culture name>` directory. It is important to note that these assemblies are not fully operational. They will not be recognized as satellite assemblies of IBM® ILOG® Gantt for .NET until they are signed with the IBM ILOG Gantt for .NET private key. In order to sign those assemblies you must send them to the technical Support where they will be signed with the IBM ILOG Gantt for .NET private key. You can test your satellite assemblies before they are signed by registering the assemblies for "Verification skipping" using the .NET framework tool named `SN.exe`.

In a Visual Studio .NET command prompt do:

```
sn -Vr <assembly>
```

for each satellite assembly.

Displaying a GanttChart in an AJAX Web Application

IBM® ILOG® Gantt for .NET comes with a set of classes that enables to build AJAX-enabled Web applications that provide rich client-side user interaction and a better user experience. This framework is based on the ASP.NET 2.0 AJAX 1.0 Extension. This extension is built into .NET Framework 3.5 and is also available as a separate download for applications targeting .NET Framework 2.0. In this case, it requires to be installed on both the development and deployment machines.

The IBM ILOG Gantt for .NET AJAX components support the following browsers:

- ◆ Microsoft Internet Explorer 6
- ◆ Microsoft Internet Explorer 7
- ◆ Firefox 2.0
- ◆ Firefox 3.x

In This Section

Overview of the IBM ILOG Gantt for .NET AJAX Framework

Briefly introduces the IBM ILOG Gantt for .NET AJAX Framework controls.

Adding AJAX Capabilities to IBM ILOG Gantt for .NET Web Controls

Explains how to use predefined AJAX extenders.

Customizing the AJAX Extenders

Explains how to customize default behaviors.

Interacting with the Client Control

Explains how to interact with the client control.

Providing Contextual Information to the Client Control

Explains how provide contextual information to the client control.

Overview of the IBM ILOG Gantt for .NET AJAX Framework

The IBM® ILOG® Gantt for .NET AJAX Framework consists of a set of server-side ASP.NET components and JavaScript components based on the Microsoft ASP.NET AJAX extension. This extension provides AJAX capabilities to ASP.NET applications by means of server-side ASP.NET Web controls and JavaScript classes. It enables postback partial-refresh thanks to the **UpdatePanel** control, an asynchronous communication layer, and provides an object-oriented JavaScript library. For more information about the Microsoft ASP.NET AJAX extension, see <http://ajax.asp.net>.

The IBM ILOG Gantt for .NET AJAX Framework allows you to add rich client-side user interactions support to IBM ILOG Gantt for .NET Web controls (that is, editing capabilities like moving activities or creating constraints, time scale panning) by means of the following new controls:

- ◆ GanttChartExtender: adds AJAX functionalities to a GanttChart Web control, like constraint creation, activities editing, time scale panning and zooming.
- ◆ ScheduleChartExtender: adds AJAX functionalities to a ScheduleChart Web control, like editing activities, re-parenting activities or panning and zooming time scale.
- ◆ ReservationChartExtender: adds AJAX functionalities to a ReservationChart Web control, like editing activities or panning and zooming time scale.

Adding AJAX Capabilities to IBM ILOG Gantt for .NET Web Controls

Adding AJAX capabilities to the IBM® ILOG® Gantt for .NET Web controls is the purpose of the GanttChart, ScheduleChart and ReservationChart control extenders defined respectively by the GantChartExtender, ScheduleChartExtender and ReservationChartExtender classes of the ILOG.Views.Gantt.Web.UI namespace.

AJAX extenders are provided as a set of server and client classes and are defined by the HierarchyChartExtender abstract class of the **ILOG.Views.Gantt.Web.UI** namespace. This is the base class for hierarchy chart AJAX extender implementations.

The **HierarchyChartExtender** class extends the ASP.NET AJAX **ExtenderControl** class to add client-side behaviors to an associated HierarchyChart Web control. It defines the required JavaScript resources and services to implement hierarchy chart client interactions.

The client-side behaviors are handled by a JavaScript implementation of the extender control that is running in the browser and initialized when the page is loaded. This JavaScript class (ILOG.Views.Gantt.Web.UI.HierarchyChartBehavior) is the base class for hierarchy chart AJAX extender client implementation.

In order to be used, an AJAX extender must be set on a hierarchy chart. When an AJAX extender is set, all input events coming to the **HierarchyChart** client-side representation in the browser are forwarded to the extender JavaScript control to process the events. The AJAX **HierarchyChart** extender is set declaratively in the .aspx file by specifying its **TargetControlID** to the hierarchy chart identifier value.

The predefined AJAX extenders available in the IBM ILOG Gantt for .NET library are:

- ◆ **GanttChart** extender
- ◆ **ScheduleChart** extender
- ◆ **ReservationChart** extender

The following example shows how to add AJAX functionalities to a **GanttChart** by means of a **GanttChartExtender** control:

The .aspx file:

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <cc1:GanttChart ID="GanttChart1" runat="server"
          Width="765px" Height="378px"
          FirstVisibleTime="01/18/2008 17:04:00"
          VerticalAlignment="Center" />
        <cc1:GanttChartExtender ID="SelectExtender1" runat="server"
          TargetControlID="GanttChart1" />
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
```

The code-behind file:

```
public partial class _Default : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) {
            GanttModelXmlSerializer ganttSerializer =
                new GanttModelXmlSerializer();
            SimpleGanttModel ganttModel = new SimpleGanttModel();
            ganttSerializer.SetFileName(ganttModel, MapPath("~/demo.sdxl"));
            GanttChart1.GanttModel = ganttModel;
        }
    }
}
```

```

        GanttChart1.ZoomToFit();
        Page.Session["GanttModel"] = ganttModel;
    }
}

protected override void OnInit(EventArgs e) {
    SimpleGanttModel ganttModel =
        (SimpleGanttModel)Page.Session["GanttModel"];
    if (ganttModel != null) {
        GanttChart1.GanttModel = ganttModel;
    }
}
}

```

Built-in Interactions

The **HierarchyChartExtender** class supports the following prebuilt interactions that are common to all chart extenders:

◆ Selecting an activity

- Single selection: click the corresponding row in the table or in the Gantt sheet.
- Interval selection: click the starting row and with the Shift key pressed click the ending row in the table or the Gantt sheet.
- Toggle selection: to toggle the selection state of an activity, keep the CTRL key pressed and click the corresponding row in the table or the Gantt sheet.
- Multiple Rows selection: to select multiple rows at once, click the starting row and drag it to the ending row.

When an activity is selected, a semi-transparent filled rectangle is displayed over the selected row. The rendering properties of this rectangle can be customized in many ways. For more information, see *Customizing the Client Selection Rendering*.

When a multiple rows selection is performed, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnRowsSelection` property to **true**.

◆ Modifying the start time and end time of an editable activity

- Modifying the start time of an activity: either click the center of the activity bar or the start of the activity bar and drag the ghost to the desired start time.
- Modifying the end time of an activity: click the end of the activity bar and drag the ghost to the desired end time.

This interaction can be disabled by setting the `CanEditActivities` property to **false**. Before an activity is modified the `BeforeEditActivity` event is raised that allows you to cancel the modification or alert the user in some particular cases.

When an activity is edited, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnEditActivity` property to **true**.

◆ Zooming the time scale

- Zoom-in the scale unit: to zoom-in a time scale unit by 2, double-click the time scale unit.
- Zoom-out the time scale: with the Shift key pressed double-click the time scale.
- Zoom-in a specific time interval: with the CTRL key pressed, click the time scale and drag it to define the new time interval.
- Zoom-out a specific time interval: with the Shift and CTRL keys pressed, click the time scale and drag it to define the new time interval.

This interaction can be disabled by setting the `CanZoomTimeScale` property to **false**.

When the time scale is zoomed, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnTimeScale` property to **true**.

◆ Panning the time scale

- Pan the time scale: to translate the visible time interval, click the time scale and drag it mouse to the desired time location.

When the time scale is panned, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnTimeScale` property to **true**.

GanttChart Extender

The `GanttChart` extender is defined by the `GanttChartExtender` class and its client control implementation by the `ILOG.Views.Gantt.Web.UI.GanttChartBehavior` JavaScript class.

It allows you to select and edit activities displayed in a **GanttChart**, as well as to scroll the visible area and to pan and zoom the time scale in the same way these actions are performed in the Windows® Forms **GanttChart**.

In addition to the interactions described in *Built-in Interactions*, the **GanttChart** extender supports the following operations:

◆ Creating a constraint

To create a constraint between two activities, keep simultaneously the CTRL key and the left mouse button pressed on an activity bar and drag it over another activity bar.

The creation of constraints can be disabled by setting the `CanCreateConstraint` property to **false**. Before a constraint is really created, the `BeforeCreateConstraint` event is raised. It allows you to cancel the creation or alert the user in some particular cases.

When a constraint is created, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnCreateConstraint` property to **true**.

- ◆ Providing contextual information to the client

The **GanttChartExtender** class allows you to provide contextual information to the client asynchronously when a selection occurs by means of the `QuerySelectionData` event. For more information on how to handle the **QuerySelectionData** event, see *Providing Contextual Information to the Client Control*.

ScheduleChart Extender

The `ScheduleChart` extender is defined by the `ScheduleChartExtender` class and its client control implementation by the **ILOG.Views.Gantt.Web.UI.ScheduleChartBehavior** JavaScript class.

It allows you to select and edit activities displayed in a **ScheduleChart**, as well as to scroll the visible area and to pan and zoom the time scale in the same way these actions are performed in the Windows® Forms **ScheduleChart**.

In addition to the interactions described in *Built-in Interactions*, the **ScheduleChart** extender supports the following operations:

- ◆ Editing a reservation
 - Move a reservation to a new resource: keep simultaneously the Shift key and the left mouse button pressed over the reservation and drag it over the new resource row. Moving a reservation can be disabled by setting the `CanMoveReservation` property to **false**.
 - Copy a reservation to a new resource: to copy a reservation to a new resource, keep simultaneously the CTRL key and the left mouse button pressed on the reservation, then drag it over the new resource row. Copying a reservation can be disabled by setting the `CanCopyReservation` property to **false**.

Before a reservation is really moved or copied, the `BeforeMoveReservation` event is raised. It allows you to cancel the edition or alert the user in some particular cases.

When a reservation is moved or copied, the command is executed through an asynchronous callback by default. You can force a post back by setting the `PostBackOnEditActivity` property to **true**.

- ◆ Providing contextual information to the client

The **ScheduleChartExtender** class allows you to provide contextual information to the client asynchronously when a selection occurs by means of the `QuerySelectionData` event. For more information on how to handle the **QuerySelectionData** event, see *Providing Contextual Information to the Client Control*.

ReservationChart Extender

The `ReservationChart` extender is defined by the `ReservationChartExtender` class and its client control implementation by the

ILOG.Views.Gantt.Web.UI.ReservationChartBehavior JavaScript class.

It allows you to select and edit activities displayed in a **ReservationChart**, as well as to scroll the visible area and to pan and zoom the time scale in the same way these actions are performed in the Windows® Forms **ReservationChart**.

In addition to the interactions described in *Built-in Interactions*, the **ReservationChart** extender supports the following operations:

- ◆ Providing contextual information to the client

The **ReservationChartExtender** class allows you to provide contextual information to the client asynchronously when a selection occurs by means of the `QuerySelectionData` event. For more information on how to handle the `QuerySelectionData` event, see *Providing Contextual Information to the Client Control*.

Customizing the AJAX Extenders

The `HierarchyChartExtender` class provides several levels of customization that allows you to modify the default behaviors.

Customizing the Client Selection Rendering

The rendering attributes of the client-side interaction ghost can be modified by means of the following **HierarchyChartExtender** properties:

- ◆ `GhostColor`: represents the color of the interaction ghost.
- ◆ `GhostStrokeWidth`: represents the stroke width of the interaction ghost.
- ◆ `GhostFillOn`: indicates whether the ghost is filled with the value of the **GhostColor** property.
- ◆ `GhostOpacity`: the opacity of the ghost color.

Changing the Internal View State Cache for Asynchronous Callback

By default, all interactions are processed via asynchronous ASP.NET callbacks. Callbacks have been introduced in ASP.NET 2.0 to add support for asynchronous request to ASP pages and controls. The benefit of these callbacks is that they are integrated to the Page lifecycle, but in a slightly modified version: the view state is not transmitted during a callback. Therefore, if one of the data stored in the view state is modified during an asynchronous callback on the server, the new value does not persist in the view state and will be overridden in the next post back. To solve this issue, the **HierarchyChartExtender** class uses an internal caching policy to maintain such data that need to be kept in sync with the view state during callback. This caching policy is defined by the **HierarchyChartExtender.IAsyncCachePolicy** interface and the default implementation uses the Page session as the data cache. Custom **IAsyncCachePolicy** implementation can be specified by means of the **AsyncCachePolicy** property of the **HierarchyChartExtender** class.

Interacting with the Client Control

As an **ExtenderControl** subclass, the **HierarchyChartExtender** class is extended on the client by a client control JavaScript implementation. In the Microsoft® JavaScript Library terminology, this client control is called **Behavior**. To access a behavior associated with an ASP.NET control, the Microsoft® JavaScript library defines the **Sys.UI.Behavior.getBehaviorByName(element, name)** and **Sys.UI.Behavior.getBehaviorsByType(element, type)** methods.

The following example shows how to get the JavaScript object associated with a **GanttChartExtender**:

```
function pageLoad() {  
    var ganttChartElement = $get('GanttChart1');  
    if (ganttChartElement) {  
        var ganttChartBehavior =  
            Sys.UI.Behavior.getBehaviorByName(ganttChartElement,  
                                             'GanttChartBehavior');  
    }  
}
```

The **ILOG.Views.Gantt.Web.UI.HierarchyChartBehavior** class defines several events that programmers can subscribe to perform specific actions. These events are:

- ◆ **timeout**: occurs on an image loading time out.
- ◆ **callbackReceived**: occurs when the response of an asynchronous callback is received.
- ◆ **doCallback**: occurs when a postback is about to be triggered by this extender.
- ◆ **customPropertiesLoaded**: occurs when the selection custom properties have been loaded.

- ◆ **imageLoaded**: occurs when the control image has been loaded.

In the Microsoft® JavaScript Library terminology, events are defined as methods of the class prototype. The accessor methods are named with `add_` and `remove_` prefixes followed by the event name.

The following example shows how to subscribe to the **callbackReceived** event of a `GanttChartBehavior` instance:

```
function pageLoad() {
    var ganttChartElement = $get('GanttChart1');
    if (ganttChartElement) {
        var ganttChartBehavior =
            Sys.UI.Behavior.getBehaviorByName(ganttChartElement,
                                              'GanttChartBehavior');
        if (ganttChartBehavior)
            ganttChartBehavior.add_callbackReceived(OnCallbackReceived);
    }
}

function OnCallbackReceived(sender, e) {
    alert('callback response received');
}
```

Providing Contextual Information to the Client Control

This section illustrates how to provide custom properties to the client control when a selection occurs. In this section the `GanttChartExtender` control is used as an example, but the illustrated methodology also applies to the `ScheduleChartExtender` and `ReservationChartExtender` classes.

When a selection occurs on the client, the **QuerySelectionData** server-side event is fired to enable the user to provide custom data to the client control. Then, once the response is received on the client, the client-side **customPropertiesLoaded** event is fired making the custom properties available to the user.

To obtain this behavior, you have to implement the following instructions:

- ◆ Subscribe to the `QuerySelectionData` event of the `GanttChartExtender` class. This event is raised on the server to fetch information related to the current selection in order to post them back to the client. Data must be provided as a key-value pair where the key is the name of the property and the value is the value of the property. On the client, the dictionary is represented as a JavaScript object, each property in the dictionary being represented as a field of this object and accessible via the value of the property key.
- ◆ Subscribe to the **customPropertiesLoaded** event of the JavaScript `GanttChartExtender` class. This event is raised when the response to a selection is received and the data is accessible from the event args parameter of the handler.

The following example shows how to provide custom data to the client control: the **QuerySelectionData** event of the **GanttChartExtender** class is used to display in the HTML page the progress value of the selected activity.

The .aspx file:

```
<head runat="server">
  <title>Untitled Page</title>
  <script type="text/javascript">

function pageLoad() {
  var ganttChartElement = $get('GanttChart1');
  if (ganttChartElement) {
    var ganttChartBehavior =
      Sys.UI.Behavior.getBehaviorByName(ganttChartElement,
        'GanttChartBehavior');

    if (ganttChartBehavior)
      ganttChartBehavior.add_customPropertiesLoaded(OnPropertiesLoaded);
  }
}

function OnPropertiesLoaded(sender, properties) {
  // custom properties are stored in the customProperties field
  // of the event parameter as an array of object. The array
  // is sized according to the selection count.
  if (properties && properties.length > 0) {
    // get the progress property that has been put during the
    // QuerySelectionData event processing.
    var progress = properties[0].progress;
    // display the property value in the span placeholder.
    var elt = $get('placeholder');
    elt.innerText = 'Progress: ' + (progress * 100) + '%';
  }
}
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <cc1:GanttChart ID="GanttChart1" runat="server"
        Width="765px" Height="378px"
        FirstVisibleTime="01/18/2008 17:04:00" ShowTooltips="True"
        VerticalAlignment="Center" />
      <cc1:GanttChartExtender ID="SelectExtender1" runat="server"
        TargetControlID="GanttChart1"
        OnQuerySelectionData="OnQuerySelectionData"/>
    </div>
    <div>
      <span id='placeholder'></span>
    </div>
  </form>
</body>
</html>
```

The code-behind file:

```
public partial class _Default : System.Web.UI.Page {
```

```

protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
        GanttModelXmlSerializer ganttSerializer =
            new GanttModelXmlSerializer();
        SimpleGanttModel ganttModel = new SimpleGanttModel();
        ganttSerializer.SetFileName(ganttModel, MapPath("~/demo.sdxl"));
        GanttChart1.GanttModel = ganttModel;
        GanttChart1.ZoomToFit();
        Page.Session["GanttModel"] = ganttModel;
    }
}

protected override void OnInit(EventArgs e) {
    SimpleGanttModel ganttModel =
        (SimpleGanttModel)Page.Session["GanttModel"];
    if (ganttModel != null) {
        GanttChart1.GanttModel = ganttModel;
    }
}

protected void OnQuerySelectionData(object sender,
    QuerySelectionDataEventArgs<IActivity> args)
{
    IActivity activity = args.SelectedObject;
    float progress = activity.WorkComplete;
    // add the WorkComplete property value to the custom properties
    // list under the 'progress' name
    args.Properties.Add("progress", progress);
}
}

```


Index

A

- activity
 - deserializing **150**
 - events **34**
 - serializing **148**
- activity bar
 - defining styles **106**
 - dialog box **109**
 - styling **100**
- ADO.NET **155**
- appearance of
 - activity bars **50, 101, 125**
 - Gantt chart **47**
 - Gantt sheet **124**
 - Gantt table **61**
 - load chart **82**
 - schedule chart **47**
 - table columns **50, 63, 65**
 - time scale **135**
- auto format **47**

C

- clipboard **167**
- constraint
 - deserializing **150**
 - events **37**
 - managing **17**
 - serializing **148**

- controls
 - load chart **81**
 - time scale **134**
- customizing
 - printing **185**
 - time scale rows **137**

D

- data model
 - ADO.NET **156**
 - catching events **34, 36, 37, 39**
 - customizing **21**
 - in-memory implementation **11**
 - interfaces **10**
 - listening to **33**
 - populating **12**
 - using the clipboard **167**
- data node
 - collapsed **52**
 - displayed **52**
 - expanded **51**
 - hidden **52**
 - visible **51**
- deserializing **149**
- dialog box
 - control for styling activity bars **109**
 - customizing for editing a constraint **178**
 - editing the columns of Gantt table **70**
 - Gantt document setup **185**

E

editing

- columns of Gantt table **70**
- constraint **178**
- values in the table **66**

event

- activity **34**
- constraint **37**
- reservation **39**
- resource **36**

G

Gantt

- controls **57**
- data model **9**

I

in-memory **12**

- extending the implementation **22**

interface

- implementing **27**

M

modification

- grouping **173**
- redoing **172**
- undoing **172**

P

printing **181**

R

reservation

- deserializing **151**
- serializing **149**

resource

- deserializing **150**
- events **36**
- serializing **148**

rows

- customizing time scale **72**
- expanding or collapsing **51, 71**
- hiding or showing **51**
- using predefined behavior **71**

S

satellite assemblies **211**

schedule entity

- accessing **28**
- creating **28**
- implementing **32**
- maintaining up-to-date relationships **30**
- triggering events **30**

schedule information

- displaying **45**
- displaying in a Gantt sheet **123**
- displaying in a Gantt table **61**

SDXL

- customizing **151**
- deserializing **149**
- overview **146**
- serializing **147**

serializing **147**

T

time scale

- customizing **136**
- introducing the class **134**
- modifying the appearance **135**
- using predefined behavior **136**

U

undo/redo

- disabling **172**
- enabling **172**

Using the Localization Tool **212**