# IBM ILOG Plant PowerOps

# 3.2

# C++ Reference Manual

**July 2009**

# Table of Contents

# Table of Contents

# Table of Contents

# Table of Contents

# Overview

| Group Summary | |
|---|---|
| optim.plant.checker | The solution checking objects. |
| optim.plant.engine | The optimization engines. |
| optim.plant.io | The input and output objects. |
| optim.plant.model | The problem object model. |
| optim.plant.solution | The solution objects. |

# Group optim.plant.checker

The solution checking objects.

| Class Summary | |
|---|---|
| IloMSChecker | The `IloMSChecker` class is used to check the validity of a model or solution. |
| IloMSCheckerMessage | The `IloMSCheckerMessage` class is used to record the errors found in a model or solution. |

Use to check solutions.

# Group optim.plant.engine

The optimization engines.

| Class Summary | |
|---|---|
| IloMSBatchingEngine | The `IloMSBatchingEngine` class is used to solve batch sizing problems to integrate planning and scheduling. |
| IloMSPlanningEngine | The `IloMSPlanningEngine` class is used to solve manufacturing planning problems represented by `IloMSModel` objects. |
| IloMSRepairAlgorithm | The `IloMSRepairAlgorithm` class is used to change a scheduling solution while enforcing (or repairing) some constraints. |
| IloMSReplicateAlgorithm | The `IloMSReplicateAlgorithm` class is used to duplicate a set of production orders. |
| IloMSSchedulingEngine | The `IloMSSchedulingEngine` class implements a scheduling engine that uses the current batching solution (`getCurrentBatchingSolution` of `IloMSModel`) to compute a scheduling solution. The solution can be accessed with `getCurrentSchedulingSolution` of `IloMSModel`. |

| Enumeration Summary | |
|---|---|
| IloMSBatchingAlgorithm | `IloMSBatchingAlgorithm` selects the batching algorithm used by the batching engine. |
| IloMSPlanningAlgorithm | `IloMSPlanningAlgorithm` sets the planning algorithm used by the planning engine. |
| IloMSRepairExtent | The enumerated type `IloMSRepairExtent` is used to identify the extent of repair with regards to constraints. |

**IBM® ILOG® Plant PowerOps contains several optimization engines**.

The planning engine uses a model containing planning data as input. This engine is typically used for mid-term planning to determine production and inventory level in **time buckets**. Constraints are satisfied globally by time bucket, not at each time unit. The output is a planning solution containing the recipes to use and in what amount per time bucket.

The batching (or lot-sizing) engine takes a model and a planning solution as input and determines how the level of production of recipes must be split into production orders (typically batches). It may also compute a pegging between these production orders in terms of material flow arcs. The output is a lot-sizing solution that is a set of production orders and material flow arcs.

The scheduling engine takes as input either a model with scheduling data or a model with a lot-sizing solution. It generates activities from production orders and tries to determine a schedule respecting the constraint at each time unit.

# Group optim.plant.io

The input and output objects.

| Class Summary | |
|---|---|
| IloMSCheckForStop | The `IloMSCheckForStop` class is used as the base class for engine stopping callback. |
| IloMSCheckForStopI | The abstract class `IloMSCheckForStopI` is used to enable you to specify a stopping condition for the scheduling engine. |
| IloMSCsvReader | The `IloMSCsvReader` class is used to read Plant PowerOps problems and solutions from CSV files. |
| IloMSCsvWriter | The `IloMSCsvWriter` class is used to write Plant PowerOps problems and solutions to CSV files. |
| IloMSDefaultCheckForStop | This `IloMSDefaultCheckForStop` class is used to enable you to stop the scheduling engine by changing the internal state of an object of this class. |
| IloMSSolutionHook | The `IloMSSolutionHook` class provides a call-back mechanism that is activated every time a new solution is found. |
| IloMSSolutionHookI | The `IloMSSolutionHookI` class provides a call-back mechanism that is activated every time a new solution is found. |

| Macro Summary | |
|---|---|
| ILOMSCHECKFORSTOP0 | This macro is provided in order to facilitate the definition of the classes `IloMSCheckForStopI` and `IloMSCheckForStop`. |

Includes classes to read and write to csv files.

# Group optim.plant.model

The problem object model.

| Class Summary | |
|---|---|
| IloMSAbstractActivity | The `IloMSAbstractActivity` class is a common superclass for both production and setup activities. |
| IloMSAbstractMaterialFlowArc | The `IloMSAbstractMaterialFlowArc` class is used to represent flows of material between stock, procurements, production orders, and customer demands. In general, such a flow of material induces a temporal constraint (precedence) between the corresponding nodes. |
| IloMSAbstractMaterialFlowNode | The `IloMSAbstractMaterialFlowNode` class is an abstract mother class for all extrema of material flow arcs; these are typically production orders, demands, and so forth. |
| IloMSAbstractProduction | The abstract class `IloMSAbstractProduction` is used to represent production orders and planned production. |
| IloMSActivity | `IloMSActivity` is used to represent production activities. |
| IloMSActivityChain | The `IloMSActivityChain` class is used to represent sequences of activities to be executed successively on the same primary resource. |
| IloMSActivityCompatibilityConstraint | The class `IloMSActivityCompatibilityConstraint` is used to represent compatibility constraints between two activities. |
| IloMSBucket | The `IloMSBucket` class is used to represent time buckets in planning. |
| IloMSBucketSequence | The `IloMSBucketSequence` class is used to represent a sequence of time buckets. |
| IloMSBucketTemplate | The `IloMSBucketTemplate` class defines a pattern or mold for bucket generation. |
| IloMSBucketTemplateSequence | The `IloMSBucketTemplateSequence` class is used to connect a set of bucket templates to a given bucket sequence. |
| IloMSCalendar | The `IloMSCalendar` class is used to represent calendars associated with the different modes of an activity. |
| IloMSCalendarInterval | `IloMSCalendarInterval` is used to represent a calendar as a set of calendar intervals. |
| IloMSDate | The class `IloMSDate` is used to contain date information. |
| IloMSDemand | The class `IloMSDemand` is used to represent the request for a certain amount of material deliverable in a time window, with an optional preferred due date. |
| IloMSDueDate | The `IloMSDueDate` class is used to represent due date objects. |
| IloMSInventoryMaxCostFunction | The class `IloMSInventoryMaxCostFunction` is used to evaluate the cost of violating the maximal inventory over time. |
| IloMSInventoryMinCostFunction | The class `IloMSInventoryMinCostFunction` is used to evaluate the cost of violating the minimal inventory (creating an inventory deficit) over time. |
| IloMSMaterial | The `IloMSMaterial` class represents Stock-Keeping Units; it is used to represent finished products, raw materials, or intermediates. |
| IloMSMaterialFamily | The `IloMSMaterialFamily` class is used to represent material families. A material may be a member of several families. Families are grouped by their type in the GUI for aggregation purpose. |

| | |
|---|---|
| IloMSMaterialFamilyCardinalityConstraint | The `IloMSMaterialFamilyCardinalityConstraint` class is used to represent cardinality constraints on the number of products that can be produced in a given time interval. |
| IloMSMaterialProduction | The class `IloMSMaterialProduction` is used to represent all production and consumption of a material. |
| IloMSMode | The `IloMSMode` class is used to represent the different ways to perform an activity. |
| IloMSModel | The `IloMSModel` class gathers the objects that define the manufacturing problem to be solved. |
| IloMSObject | The class `IloMSObject` is used to represent all types of Plant PowerOps objects. |
| IloMSOptimizationCriterion | The `IloMSOptimizationCriterion` class is used to represent optimization criteria. |
| IloMSOptimizationProfile | The `IloMSOptimizationProfile` class is used to represent optimization profiles. |
| IloMSPlannedDelivery | The class `IloMSPlannedDelivery` is used in a planning solution to represent the amount of material that exits from inventory in a period of time to satisfy a specific demand. This amount is in addition to any already-fixed pegging arcs to the demand. |
| IloMSPlannedProduction | The class `IloMSPlannedProduction` is used to represent the quantity of a recipe that is planned to be executed in a period of time. |
| IloMSPrecedence | The `IloMSPrecedence` class is used to represent precedence constraints between activities. |
| IloMSProcurement | The class `IloMSProcurement` is used to represent material procured from outside the plant or the initial stock. |
| IloMSProcurementToDemandArc | The `IloMSProcurementToDemandArc` class is used to represent flow of material between a procurement and a demand. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSProcurementToProdArc | The `IloMSProcurementToProdArc` class is used to represent flow of material between a procurement and a production order. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSProcurementToStorageArc | The `IloMSProcurementToStorageArc` class is used to represent flow of material between a procurement and stock or storage. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSProdToDemandArc | The `IloMSProdToDemandArc` class is used to represent flow of material between a production order and a demand. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSProdToProdArc | The `IloMSProdToProdArc` class is used to represent flow of material between two production orders. Note that a material flow between production orders represents a precedence constraint between them. |
| IloMSProdToStorageArc | The `IloMSProdToStorageArc` class is used to represent flow of material between a production order and stock or storage. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSProductionOrder | The class `IloMSProductionOrder` is used to represent production orders. |
| IloMSQuality | |

| | |
|---|---|
| | The class `IloMSQuality` is used to represent qualities of materials. |
| IloMSRecipe | The class `IloMSRecipe` is used to represent production recipes. |
| IloMSRecipeFamily | The `IloMSRecipeFamily` class is used to represent recipe families. A recipe may be a member of several families. Families are a way to define submodels and to perform a partial database load of the data. |
| IloMSRecipeFamilyFilter | The `IloMSRecipeFamilyFilter` class is used to filter a model to create a submodel that contains recipe families. |
| IloMSResource | The `IloMSResource` class is used to describe production resources on which activities will be performed. |
| IloMSResourceCapacityCostFunction | The class `IloMSResourceCapacityCostFunction` is used to evaluate the cost of using a resource over time. |
| IloMSResourceConstraint | The class `IloMSResourceConstraint` is used to represent a resource requirement on an activity. |
| IloMSResourceFamily | The `IloMSResourceFamily` class is used to represent resource families. A resource may be a member of several families. Families are grouped by their type in the GUI for aggregation purpose. |
| IloMSScheduledActivity | The class `IloMSScheduledActivity` is used to represent an activity as scheduled on the shop floor. |
| IloMSScope | The `IloMSScope` class represents *scopes*, used in the GUI as a means to control optimization. |
| IloMSSetupActivity | The `IloMSSetupActivity` class is used to **explicitly** represent setup activities. |
| IloMSSetupMatrix | The `IloMSSetupMatrix` class is used to store the setup time and cost incurred when two production activities follow each other on the same resource. |
| IloMSStandardKPI | The `IloMSStandardKPI` class is used to represent the Key Performance Indicators that are defined as *standard* in PPO. |
| IloMSStorageToDemandArc | The `IloMSStorageToDemandArc` class is used to represent flow of material between stock or storage and a demand. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSStorageToProdArc | The `IloMSStorageToProdArc` class is used to represent flow of material between stock or storage and a production order. Note that a material flow between two nodes represents a precedence constraint between them. |
| IloMSStorageUnit | The `IloMSStorageUnit` class represents a location or storage facility for materials. |
| IloMSUnit | The `IloMSUnit` class is used to represent units of measure. |

| **Macro Summary** | |
|---|---|
| IloMSIdentifier | `IloMSIdentifier` is used to provide a name for programming elements. |
| IloMSIntMinusInfinity | `IloMSIntMinusInfinity` represents negative infinity as `-999999999`. |
| IloMSIntPlusInfinity | `IloMSIntPlusInfinity` represents positive infinity as `999999999`. |
| IloMSNoFeature | `IloMSNoFeature` is used to represent that no feature is selected. |

| **Enumeration Summary** | |
|---|---|
| IloMSActivityCompatibilityType | `IloMSActivityCompatibilityType` is used to select a type used to enforce a compatibility constraint. |

| | |
|---|---|
| IloMSBucketPeriodUnit | `IloMSBucketPeriodUnit` is used to select the time period over which the buckets will be generated. Period unit is a calendar unit. This period unit must be greater than or equal to the bucket type. |
| IloMSBucketType | `IloMSBucketType` is used to select the type (the default duration) of buckets defined by bucket templates. |
| IloMSCheckerMessageLevel | `IloMSCheckerMessageLevel` is used to identify the severity of the checker message. |
| IloMSCleaningStatus | `IloMSCleaningStatus` identifies the cleaning status of an activity. |
| IloMSDay | The enumerated type `IloMSDay` is used to identify the day of the week. |
| IloMSDemandCompatibilityType | `IloMSDemandCompatibilityType` table places logical constraints on demand deliveries in the planning model. |
| IloMSDimension | `IloMSDimension` is used to identify the dimension type of the unit. |
| IloMSMaterialFlowNodeType | `IloMSMaterialFlowNodeType` is used to define the node type of the material flow. |
| IloMSMaterialFlowType | `IloMSMaterialFlowType` is used to define the path of material flow. |
| IloMSPeggingStrategy | `IloMSPeggingStrategy` is used to identify the pegging strategy to be used with arcs carrying a given `IloMSMaterial`. |
| IloMSPerformedStatus | The enumerated type `IloMSPerformedStatus` is used to identify the status of an activity in the scheduling solution as performed or unperformed. To let the scheduling engine decide whether or not to perform an activity, the activity must have the status `IloMSPerformedOrUnperformed`. |
| IloMSPlanningSetupModel | `IloMSPlanningSetupModel` is used to define which approximation of the setup model the planning engine must take into account for each resource in a given time interval. |
| IloMSPrecedenceType | `IloMSPrecedenceType` is used to define the type of precedence constraint between two activities. |
| IloMSRecipeFamilyStatus | `IloMSRecipeFamilyStatus` is used to give more semantics to an `IloMSRecipeFamilyFilter`. |
| IloMSRecipeType | `IloMSRecipeType` is used to provide additional semantics to a recipe. |
| IloMSRepairCapacity | The enumerated type `IloMSRepairCapacity` is used to identify the capacity of repair. |
| IloMSServiceLevelType | `IloMSServiceLevelType` sets the service level. |

| **Function Summary** | |
|---|---|
| operator<< | This operator directs output to an output stream, usually standard output. |

The problem object model gathers classes used to define an IBM® ILOG® Plant PowerOps model.

Most classes in this group inherit from `IloMSObject`.

Plant PowerOps can be used to describe manufacturing scheduling and planning problems.

A manufacturing scheduling problem includes a description of the activities to be performed; of the different modes in which each of these activities can be performed; of the resources and types of setups required in each mode; and of a variety of additional constraints (for example, precedence constraints) to satisfy.

A solution assigns to activities the modes, start times, and end times that satisfy the constraints. In some cases, it is permissible to leave an activity in an "unperformed" mode, meaning that a mode, a start time, and an end time are chosen, but no resources are assigned to the activity. This might be used to pinpoint the need to either add capacity on a resource or subcontract some of the activities, rather than performing them too early or too late.

A "good" solution must typically respond to a number of conflicting optimization criteria. For example, a solution may have to balance criteria specifying the lowest possible manufacturing cost; the avoidance of time-consuming machine setups; and delivery of the products on time (for example, not too early to avoid storage costs, nor too late to avoid inconvenience and cost to the customer). The importance given to different optimization criteria varies depending on the environment (importance of setups, need to reduce storage, etc.) and the given context (high versus low demand). The overall (compound) optimization criterion is defined as a linear combination of the individual criteria. The weight of each criterion is provided as part of the model.

Plant PowerOps offers an API that provides more than just activities and resources; the notions of recipes, production orders, materials and demand are also available.

A recipe is the description of a generic process that transforms some input materials into output material(s). A recipe consists of a set of prototype activities, linked by precedence and possibly other constraints. Each prototype activity is executable in one or several modes, and consumes and produces given materials in given quantities.

Production orders (or batches) are obtained by instantiating recipes. A production order specifies the batch size and the recipe to use. For each order, production activities and associated constraints are generated by instantiating the prototype activities of the recipe, for a batch of the given size.

# Group optim.plant.solution

The solution objects.

| Class Summary | |
|---|---|
| IloMSBatchingSolution | The `IloMSBatchingSolution` class is used to represent a batching solution. |
| IloMSPlanningSolution | The `IloMSPlanningSolution` class is used to represent partial and complete planning solutions found by a planning engine. |
| IloMSSchedulingSolution | The `IloMSSchedulingSolution` class is used to represent partial and complete solutions to the scheduling problem defined in the corresponding instance of `IloMSModel`. |

**IBM® ILOG® Plant PowerOps solution management.**

Plant PowerOps contains several engines that each create a solution object when their method `solve()` returns true. Solutions are both created and used as input by PPO engines.

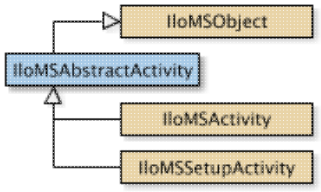
For instance, the planning engine creates a planning solution, and the batching engine creates a batching solution; but the batching engine requires a planning solution as input.

**Notion of initial solution:** Scheduling Solutions can be used as a starting point of an optimization and may be repaired if they are not feasible.

**Notion of frozen solution:** A frozen solution contains elements that optimization will not alter; for example, the start time of an activity must not be changed.

# Class IloMSAbstractActivity

**Definition file:** ilplant/absactivity.h
**Library:** plant




The `IloMSAbstractActivity` class is a common superclass for both production and setup activities.
An activity can be performed in one or different modes.

Due dates can be attached either to the start time or to the end time of an activity. Earliness and tardiness costs will be incurred when an activity does not start or end at the given due date. When several due dates apply to the same activity, the earliness and tardiness costs of the activity are defined as the maximal earliness and tardiness costs over all the given due dates.

Precedence constraints are used to relate the start and end times of two activities and impose minimal and maximal delays between them.

All the methods of the `IloMSAbstractActivity` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSMode, IloMSDueDate, IloMSPrecedence

| Method Summary | |
|---|---|
| public IloMSActivityChain | getActivityChain() const |
| public IloMSActivityCompatibilityConstraint | getActivityCompatibilityConstraint(IloInt index) const |
| public IloMSCleaningStatus | getCleaningStatus() const |
| public IloMSDueDate | getDueDate(IloInt index) const |
| public IloMSPrecedence | getIncomingPrecedence(IloInt index) const |
| public IloMSMode | getMode(IloInt index) const |
| public IloMSMode | getModePrototype(IloInt generatedIndex) const |
| public IloInt | getNumberOfActivityCompatibilityConstraints() const |
| public IloInt | getNumberOfDueDates() const |
| public IloInt | getNumberOfIncomingPrecedences() const |
| public IloInt | getNumberOfModes() const |
| public IloInt | getNumberOfOutgoingPrecedences() const |
| public IloMSPrecedence | getOutgoingPrecedence(IloInt index) const |
| public IloMSProductionOrder | getProductionOrder() const |
| public IloMSRecipe | getRecipe() const |
| public IloBool | hasActivityChain() const |
| public IloBool | hasProductionOrder() const |
| public IloBool | hasRecipe() const |

| | |
|---|---|
| public IloBool | isActivityPrototype() const |
| public IloBool | isProductionActivity() const |
| public IloBool | isSetupActivity() const |
| public void | setCleaningStatus(IloMSCleaningStatus status) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public IloMSActivity | toProductionActivity() const |
| public IloMSSetupActivity | toSetupActivity() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloMSActivityChain` **`getActivityChain`**`() const`

This method returns the activity chain containing the invoking activity. An exception is thrown if the activity is not in an activity chain.

`public IloMSActivityCompatibilityConstraint`
**`getActivityCompatibilityConstraint`**`(IloInt index) const`

This method returns the activity compatibility constraint of the invoking activity with the given `index`. An exception is thrown if the given `index` is out of bounds.

`public IloMSCleaningStatus` **`getCleaningStatus`**`() const`

This method returns the cleaning status of the invoking activity.

`public IloMSDueDate` **`getDueDate`**`(IloInt index) const`

This method returns the due date of the invoking activity with the given `index`. An exception is thrown if the given `index` is out of bounds.

`public IloMSPrecedence` **`getIncomingPrecedence`**`(IloInt index) const`

This method returns the incoming precedence of the invoking activity with the given `index`. An exception is thrown if the given `index` is out of bounds.

`public IloMSMode` **`getMode`**`(IloInt index) const`

This method returns the mode of the invoking activity with the given `index`. On a generated activity from a production order, it returns the generated mode. On an activity prototype from a recipe, it returns the mode

prototype. An exception is thrown if the given `index` is out of bounds.

```
public IloMSMode getModePrototype(IloInt generatedIndex) const
```

This method returns the mode prototype corresponding to the generated mode specified by `generatedIndex`. This method can be called from a generated activity or a prototype.

```
public IloInt getNumberOfActivityCompatibilityConstraints() const
```

This method returns the number of activity compatibility constraints associated with the invoking activity.

```
public IloInt getNumberOfDueDates() const
```

This method returns the number of due dates of the invoking activity.

```
public IloInt getNumberOfIncomingPrecedences() const
```

This method returns the number of incoming precedence constraints of the invoking activity; that is, the number of precedence constraints for which the invoking activity is the "successor" activity.

```
public IloInt getNumberOfModes() const
```

This method returns the number of modes of the invoking activity.

```
public IloInt getNumberOfOutgoingPrecedences() const
```

This method returns the number of outgoing precedence constraints of the invoking activity; that is, the number of precedence constraints for which the invoking activity is the "predecessor" activity.

```
public IloMSPrecedence getOutgoingPrecedence(IloInt index) const
```

This method returns the outgoing precedence of the invoking activity with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSProductionOrder getProductionOrder() const
```

This method returns the production order to which the invoking activity belongs. An exception is thrown if the invoking activity does not belong to any production order.

```
public IloMSRecipe getRecipe() const
```

This method returns the recipe to which the invoking activity (a prototype activity) belongs. An exception> is thrown if the invoking activity is not a prototype activity.

```
public IloBool hasActivityChain() const
```

This method returns true if the invoking activity is part of an activity chain.

```
public IloBool hasProductionOrder() const
```

This method returns true if the invoking activity belongs to a production order.

```
public IloBool hasRecipe() const
```

This method returns true if the invoking activity is a prototype activity.

```
public IloBool isActivityPrototype() const
```

This method returns true if the invoking activity is an activity prototype.

```
public IloBool isProductionActivity() const
```

This method returns true if the invoking activity is an instance of the class `IloMSActivity`.

```
public IloBool isSetupActivity() const
```

This method returns true if the invoking activity is an instance of the class `IloMSSetupActivity`.

```
public void setCleaningStatus(IloMSCleaningStatus status)
```

This method sets the cleaning status of the invoking activity.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking activity. An exception is thrown if the given identifier is already used.

```
public IloMSActivity toProductionActivity() const
```

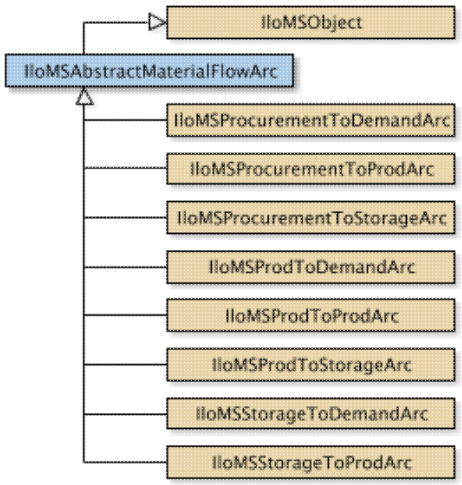
This method casts the invoking activity to an `IloMSActivity`.

```
public IloMSSetupActivity toSetupActivity() const
```

This method casts the invoking activity to an `IloMSSetupActivity`.

# Class IloMSAbstractMaterialFlowArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSAbstractMaterialFlowArc` class is used to represent flows of material between stock, procurements, production orders, and customer demands. In general, such a flow of material induces a temporal constraint (precedence) between the corresponding nodes.
Eight types of material flow arcs are distinguished. A stock-to-production arc means that some material initially in stock will be used as an input to a given production order. A stock-to-demand arc means that some material initially in stock will be used to (partially or totally) satisfy a demand. A procurement-to-production arc means that some material provided by a given procurement will be used as an input to a given production order. A procurement-to-demand arc means that some material provided by a given procurement will be used to (partially or totally) satisfy a demand. A procurement-to-stock arc means that some material provided by a given procurement will remain in stock. A production-to-production arc means that some material produced by a given production order will be used as an input to another given production order. A production-to-demand to demand arc means that some material produced by a given production order will be used to (partially or totally) satisfy a demand. A production-to-storage arc means that some material produced by a given production order will remain in stock.

A material flow arc specifies the quantity of material that is moved between the two corresponding nodes. Firm minimal and maximal quantities can also be specified. When the firm minimal quantity is not 0, PPO is not allowed to remove the arc and the quantity on the arc is not allowed to go below the firm minimal quantity. When the firm maximal quantity is not infinite, PPO is not allowed to extend the quantity on the arc above the given maximal quantity.

All the methods of the `IloMSAbstractMaterialFlowArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public IloInt | getEndConsumptionTime() const |
| public IloInt | getEndProductionTime() const |
| public IloNum | getFirmQuantityMax() const |
| public IloNum | getFirmQuantityMin() const |
| public IloMSMaterial | getMaterial() const |
| public IloMSMaterialFlowType | getMaterialFlowType() const |
| public const char * | getMaterialFlowTypeName() const |

| | |
|---|---|
| public IloMSAbstractMaterialFlowNode | getPredecessor() const |
| public IloNum | getQuantity() const |
| public IloNum | getQuantityInDisplayUnit() const |
| public IloInt | getStartConsumptionTime() const |
| public IloInt | getStartProductionTime() const |
| public IloMSAbstractMaterialFlowNode | getSuccessor() const |
| public IloBool | isFirm() const |
| public void | setFirm(IloBool firm) |
| public void | setFirmQuantityMax(IloNum firmQuantityMax) |
| public void | setFirmQuantityMin(IloNum firmQuantityMin) |
| public void | setQuantity(IloNum quantity) |
| public IloMSProcurementToDemandArc | toProcurementToDemandArc() const |
| public IloMSProcurementToProdArc | toProcurementToProdArc() const |
| public IloMSProcurementToStorageArc | toProcurementToStorageArc() const |
| public IloMSProdToDemandArc | toProdToDemandArc() const |
| public IloMSProdToProdArc | toProdToProdArc() const |
| public IloMSProdToStorageArc | toProdToStorageArc() const |
| public IloMSStorageToDemandArc | toStorageToDemandArc() const |
| public IloMSStorageToProdArc | toStorageToProdArc() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public IloInt **getEndConsumptionTime**() const

internal

public IloInt **getEndProductionTime**() const

internal

public IloNum **getFirmQuantityMax**() const

This method returns the maximal quantity carried by this arc.

public IloNum **getFirmQuantityMin**() const

This method returns the minimal quantity carried by this arc.

public IloMSMaterial **getMaterial**() const

This method returns the material along the arc.

```
public IloMSMaterialFlowType getMaterialFlowType() const
```

This method returns the type of the material flow arc.

```
public const char * getMaterialFlowTypeName() const
```

This method returns the type of the material flow arc as a readable string.

```
public IloMSAbstractMaterialFlowNode getPredecessor() const
```

This method returns the node (procurement, production order, or null) which provides the material flowing along the arc.

```
public IloNum getQuantity() const
```

This method returns the quantity of material along the arc.

```
public IloNum getQuantityInDisplayUnit() const
```

This method returns the quantity of material along the arc, converted in the display unit of the material.

```
public IloInt getStartConsumptionTime() const
```

internal

```
public IloInt getStartProductionTime() const
```

internal

```
public IloMSAbstractMaterialFlowNode getSuccessor() const
```

This method returns the node (production order, demand, or null) which consumes the material flowing along the arc.

```
public IloBool isFirm() const
```

This method returns true if the firm minimal quantity carried by this arc is strictly positive, i.e., if the arc cannot be removed.

```
public void setFirm(IloBool firm)
```

If the given Boolean is true, this method sets the firm minimal quantity and the firm maximal quantity of the arc to its current quantity. Otherwise (i.e., if the given Boolean is false), this method resets the firm minimal quantity and the firm maximal quantity of the arc to 0 and infinity.

`public void` **`setFirmQuantityMax`**`(IloNum firmQuantityMax)`

This method sets the maximal quantity carried by this arc.

`public void` **`setFirmQuantityMin`**`(IloNum firmQuantityMin)`

This method sets the minimal quantity carried by this arc.

`public void` **`setQuantity`**`(IloNum quantity)`

This method sets the quantity of material along the arc.

`public IloMSProcurementToDemandArc` **`toProcurementToDemandArc`**`() const`

This method casts the invoking arc to an `IloMSProcurementToDemandArc`.

`public IloMSProcurementToProdArc` **`toProcurementToProdArc`**`() const`

This method casts the invoking arc to an `IloMSProcurementToProdArc`.

`public IloMSProcurementToStorageArc` **`toProcurementToStorageArc`**`() const`

This method casts the invoking arc to an `IloMSProcurementToStorageArc`.

`public IloMSProdToDemandArc` **`toProdToDemandArc`**`() const`

This method casts the invoking arc to an `IloMSProdToDemandArc`.

`public IloMSProdToProdArc` **`toProdToProdArc`**`() const`

This method casts the invoking arc to an `IloMSProdToProdArc`.

`public IloMSProdToStorageArc` **`toProdToStorageArc`**`() const`

This method casts the invoking arc to an `IloMSProdToStorageArc`.

`public IloMSStorageToDemandArc` **`toStorageToDemandArc`**`() const`

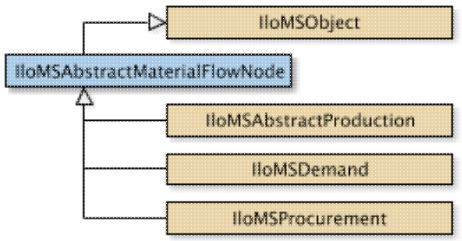
This method casts the invoking arc to an `IloMSStorageToDemandArc`.

`public IloMSStorageToProdArc` **`toStorageToProdArc`**`() const`

This method casts the invoking arc to an `IloMSStorageToProdArc`.

# Class IloMSAbstractMaterialFlowNode

**Definition file:** ilplant/matflownode.h
**Library:** plant




The `IloMSAbstractMaterialFlowNode` class is an abstract mother class for all extrema of material flow arcs; these are typically production orders, demands, and so forth.

**See Also:** IloMSAbstractMaterialFlowNode, IloMSAbstractMaterialFlowArc, IloMSProductionOrder, IloMSDemand

| Method Summary | |
|---|---|
| public const char * | getCategory() const |
| public IloMSMaterialFlowNodeType | getMaterialFlowNodeType() const |
| public IloBool | isFirm() const |
| public void | setCategory(const char * category) |
| public void | setFirm(IloBool firm) |
| public IloMSDemand | toDemand() |
| public IloMSProductionOrder | toProductionOrder() |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public const char * getCategory() const
```

This method returns the status or category of a counterpart or counter-component in an external system, such as SAP-R/3-APO.

```
public IloMSMaterialFlowNodeType getMaterialFlowNodeType() const
```

This method returns the type of node.

```
public IloBool isFirm() const
```

This method returns true if the node is firm.

```
public void setCategory(const char * category)
```

This method sets the status or category of a counterpart or counter-component in an external system, such as SAP-R/3-APO.

```
public void setFirm(IloBool firm)
```

This method makes the node firm.

```
public IloMSDemand toDemand()
```

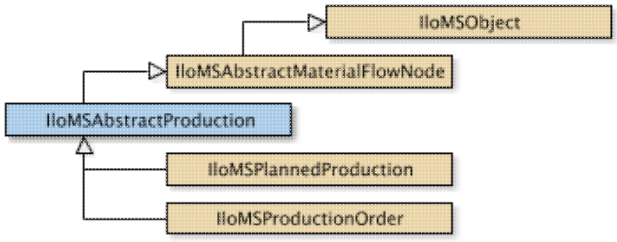
This method casts the invoking node into a demand object.

```
public IloMSProductionOrder toProductionOrder()
```

This method casts the invoking node into a production order.

# Class IloMSAbstractProduction

**Definition file:** ilplant/absproduction.h
**Library:** plant




The abstract class `IloMSAbstractProduction` is used to represent production orders and planned production.
It represents a production using the recipe of a process. The batch size is used to compute the quantity of material produced or consumed. Batch size is also used to adjust the processing time of generated activities (if a variable processing time has been specified on the activity prototypes of the recipe).

**See Also:** IloMSRecipe, IloMSDemand, IloMSAbstractMaterialFlowArc, IloMSPlannedProduction, IloMSProductionOrder

| Method Summary | |
|---|---|
| public void | addPlannedMode(IloMSMode modePrototype) |
| public IloNum | getBatchSize() const |
| public IloInt | getNumberOfPlannedModes() const |
| public IloMSMode | getPlannedMode(IloInt activityPrototypeIndex) const |
| public IloInt | getPlannedTimeMax() const |
| public IloInt | getPlannedTimeMin() const |
| public IloMSRecipe | getRecipe() const |
| public IloBool | isFirm() const |
| public void | setBatchSize(IloNum batchSize) |
| public void | setFirm(IloBool firm) |
| public void | setPlannedTimeMax(IloInt value) |
| public void | setPlannedTimeMin(IloInt value) |

| Inherited Methods from `IloMSAbstractMaterialFlowNode` |
|---|
| getCategory, getMaterialFlowNodeType, isFirm, setCategory, setFirm, toDemand, toProductionOrder |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public void addPlannedMode(IloMSMode modePrototype)
```

This method adds a reference to the mode prototype in the original recipe. This mode prototype corresponds to a decision of the planning engine with respect to which resource was used among the alternatives defined in the original recipe.

```
public IloNum getBatchSize() const
```

This method returns the batch size, that is, the quantity of recipe to be executed by this production order.

```
public IloInt getNumberOfPlannedModes() const
```

This method returns the number of mode prototypes corresponding to the decisions of the planning engine, with respect to which resource was used among the alternatives defined in the original recipe.

```
public IloMSMode getPlannedMode(IloInt activityPrototypeIndex) const
```

This method returns the mode of#-th activity prototype. This mode prototype corresponds to a decision of the planning engine with respect to which resource is to be used among the alternatives defined in the original recipe.

```
public IloInt getPlannedTimeMax() const
```

This accessor returns an approximate maximal time at which the production order is supposed to be done. The time is expressed in the time unit of the model, from the date origin of the model.

```
public IloInt getPlannedTimeMin() const
```

This accessor returns an approximate minimal time at which the production order could be done. The time is expressed in the time unit of the model, from the date origin of the model.

```
public IloMSRecipe getRecipe() const
```

This accessor returns the recipe implemented by the invoking production order.

```
public IloBool isFirm() const
```

This method returns the status of the invoking object (firm batch size min is strictly positive).

```
public void setBatchSize(IloNum batchSize)
```

This method sets the batch size, that is, the quantity of recipe to be executed by this production order.

```
public void setFirm(IloBool firm)
```

This method sets the firm batch size min and max to the current batch size.

A firm production order must be scheduled in order to be taken into account in the planning module (a scheduled activity must exist for each of its activities). The planning module will not try to move it. The planning module will treat a firm order as a fixed order, as opposed to the scheduling module which may try to move it.

```
public void setPlannedTimeMax(IloInt value)
```

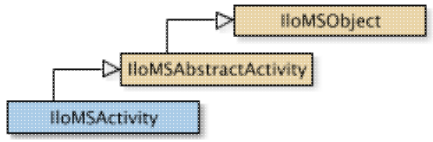
This accessor sets an approximate maximal time at which the production order is supposed to be done. The time is expressed in the time unit of the model, from the date origin of the model.

```
public void setPlannedTimeMin(IloInt value)
```

This accessor sets an approximate minimal time at which the production order could be done. The time is expressed in the time unit of the model, from the date origin of the model.

# Class IloMSActivity

**Definition file:** ilplant/activity.h
**Library:** plant




`IloMSActivity` is used to represent production activities.
The `IloMSActivity` class inherits from the class `IloMSAbstractActivity`.

**Activity types and phases**

There are three basic activity types in PPO: Production, cleanup, and setup activities. `IloMSActivity` represents the first two types (cleanup activities are production activities with a status set to cleaning with `IloMSAbstractActivity::setCleaningStatus`). Setup activities are instances of the class `IloMSSetupActivity`.

An activity typically has three phases in PPO: As a prototype, an instance, and as a scheduled activity. A prototype activity is a template or mold of an activity, represented through the recipes and modes in the data model. A prototype is not an actual specific activity instance, but rather a model of an activity that typically is performed many times in a plant process. During optimization, the production orders are created and this process uses the activity prototypes as a template to create the actual, explicit instances of activities. Each activity instance must be performed at some point in the schedule in order to meet the demand of the production orders. Optimization schedules each of these instances to a particular time slot, and the results can be viewed in the **Scheduled Activities** data table in the GUI.

All the methods of the `IloMSActivity` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**Setup states on production activities**

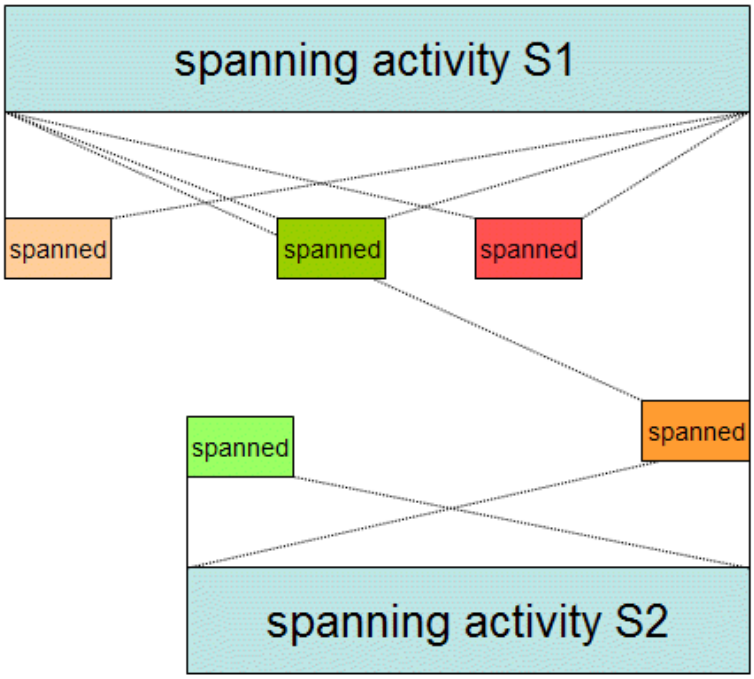
A setup state per setup feature can be associated with each production activity. A setup time and a setup cost will be incurred between any two production activities directly following each other on the same unary resource. This setup time and setup cost depend on the setup states of the two activities. By default, the setup state of an activity is set to the special value -1, meaning the setup time and the setup cost before or after the activity is equal to 0.

**Spanning over multiple activities**

An activity `s` can span over a set of activities `{a}`. The activity `s` is called the spanning activity and its time bounds are computed as a propagation of the time bounds of its spanned activities. More precisely, it means that the start time of the spanning activity is equal to the earliest of the start times of the spanned activities, and that the end time of the spanning activity is equal to the latest of the end times of the spanned activities.

The main difference between precedence constraints and spanning constraints is that with precedence constraints, one can specify that an activity covers a set of activities but if there is no known order among the activities then one cannot constrain the covering activity so that:

- The start time of the covering activity is equal to the start of the earliest covered activity
- The end time of the covering activity is equal to the end time of the latest covered activity.




**Modifying activity names**

You can define multiline names using **n** to start a new line. To properly see multiline names in the Gantt, change the Activity and Resource Chart Row Height in the Options menu.

By default, activities generated from activity prototypes have an automatic long name based on a concatenation of the production order name and the activity prototype name, separated by a dot. To keep the original name of the activity prototype, you must set the setting `bShortName` to true. When the `bShortName` setting is true, the activity label from the activity prototypes interprets the following special characters as shown:

- ^a = abbreviated main product name to 3 first letters (NOT SUPPORTING MULTIBYTE CHARACTERS)
- ^b = abbreviated main ingredient name to 3 first letters (NOT SUPPORTING MULTIBYTE CHARACTERS)
- ^i = main product identifier
- ^j = main ingredient identifier
- ^m = main ingredient name
- ^n = order index
- ^o = order identifier
- ^p = main product name
- ^q = quantity of main product or batch size; if missing round up as an integer
- ^r = recipe name, or if missing then the recipe identifier, or if missing the recipe id
- ^s = batch size

See the method `putSetting` of `IloMSOptimizationProfile` for more information.

**See Also:** IloMSAbstractActivity

| Method Summary | |
|---|---|
| `public void` | `addSpannedActivity(IloMSActivity activity, IloBool onStart, IloBool onEnd)` |
| `public void` | `addSpannedActivity(IloMSActivity activity)` |
| `public ILOMSDEPRECATED IloInt` | `getEndMaxIfPerformed()` |
| `public ILOMSDEPRECATED IloInt` | `getEndMinIfPerformed()` |
| `public IloInt` | `getNumberOfSpannedActivities() const` |

| | |
|---|---|
| public IloMSPerformedStatus | getPerformedStatus() const |
| public IloMSSetupActivity | getSetupActivity() const |
| public IloMSIdentifier | getSetupState() const |
| public IloMSIdentifier | getSetupState(IloMSIdentifier feature) const |
| public IloMSActivity | getSpannedActivity(IloInt index) const |
| public ILOMSDEPRECATED IloInt | getStartMaxIfPerformed() |
| public ILOMSDEPRECATED IloInt | getStartMinIfPerformed() |
| public IloBool | hasSetupActivity() const |
| public IloBool | hasSetupState() const |
| public IloBool | hasSetupState(IloMSIdentifier feature) const |
| public IloBool | isSpanningOnEnd(IloInt index) const |
| public IloBool | isSpanningOnStart(IloInt index) const |
| public IloMSSetupActivity | newSetupActivity() |
| public void | removeSetupState(IloMSIdentifier feature) |
| public ILOMSDEPRECATED void | setEndMaxIfPerformed(IloInt value) |
| public ILOMSDEPRECATED void | setEndMinIfPerformed(IloInt value) |
| public void | setPerformedStatus(IloMSPerformedStatus perfStatus) |
| public void | setSetupState(IloMSIdentifier setupState) |
| public void | setSetupState(IloMSIdentifier feature, IloMSIdentifier setupState) |
| public ILOMSDEPRECATED void | setStartMaxIfPerformed(IloInt value) |
| public ILOMSDEPRECATED void | setStartMinIfPerformed(IloInt value) |

| **Inherited Methods from `IloMSAbstractActivity`** |
|---|
| getActivityChain, getActivityCompatibilityConstraint, getCleaningStatus, getDueDate, getIncomingPrecedence, getMode, getModePrototype, getNumberOfActivityCompatibilityConstraints, getNumberOfDueDates, getNumberOfIncomingPrecedences, getNumberOfModes, getNumberOfOutgoingPrecedences, getOutgoingPrecedence, getProductionOrder, getRecipe, hasActivityChain, hasProductionOrder, hasRecipe, isActivityPrototype, isProductionActivity, isSetupActivity, setCleaningStatus, setIdentifier, toProductionActivity, toSetupActivity |

| **Inherited Methods from `IloMSObject`** |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public void **addSpannedActivity**(IloMSActivity activity, IloBool onStart, IloBool onEnd)

This method adds a spanned activity to the invoking activity. The spanning constraint may be enforced only on start or only on end, or on both.

```
public void addSpannedActivity(IloMSActivity activity)
```

This method adds a spanned activity to the invoking activity. The spanning constraint is enforced both on start and on end.

```
public ILOMSDEPRECATED IloInt getEndMaxIfPerformed()
```

This deprecated method returns the maximal end time imposed on the generated activity when it is performed.

```
public ILOMSDEPRECATED IloInt getEndMinIfPerformed()
```

This deprecated method returns the minimal end time imposed on the generated activity when it is performed.

```
public IloInt getNumberOfSpannedActivities() const
```

This method returns the number of spanned activities covered by the invoking activity.

```
public IloMSPerformedStatus getPerformedStatus() const
```

This method returns the performed status for the activity. When the value returned is `IloMSPerformedOrUnperformed`, it means that the scheduling engine must decide to set this activity to either performed or unperformed in the solution. The decision taken by the scheduling engine can be accessed by using the appropriate method in the scheduling solution.

**See Also:** IloMSPerformedStatus, IloMSSchedulingSolution

```
public IloMSSetupActivity getSetupActivity() const
```

This method returns the explicit setup activity associated with the invoking production activity. A null reference is returned if the invoking production activity has no explicit setup activity.

```
public IloMSIdentifier getSetupState() const
```

This method returns the setup state of the invoking activity. It is equivalent to calling `getSetupState(IloMSNoFeature)`.

```
public IloMSIdentifier getSetupState(IloMSIdentifier feature) const
```

This method returns the setup state of the invoking activity for the given setup `feature`.

```
public IloMSActivity getSpannedActivity(IloInt index) const
```

This method returns the spanned activity covered by the invoking activity associated with `index`.

```
public ILOMSDEPRECATED IloInt getStartMaxIfPerformed()
```

This deprecated method returns the maximal start time imposed on the generated activity when it is performed.

```
public ILOMSDEPRECATED IloInt getStartMinIfPerformed()
```

This deprecated method returns the minimal start time imposed on the generated activity when it is performed.

```
public IloBool hasSetupActivity() const
```

This method returns true if the invoking activity has an explicit setup activity, and false otherwise.

```
public IloBool hasSetupState() const
```

This method returns true if a setup state is defined for the invoking activity.

```
public IloBool hasSetupState(IloMSIdentifier feature) const
```

This method returns true if a setup state is defined for the invoking activity for the given setup `feature`.

```
public IloBool isSpanningOnEnd(IloInt index) const
```

This method returns true if the spanning constraint is enforced on end with the indexed spanned activity.

```
public IloBool isSpanningOnStart(IloInt index) const
```

This method returns true if the spanning constraint is enforced on start with the indexed spanned activity.

```
public IloMSSetupActivity newSetupActivity()
```

This method creates an explicit setup activity for the invoking production activity.

```
public void removeSetupState(IloMSIdentifier feature)
```

This method removes the setup state required by the invoking activity for the given setup feature.

```
public ILOMSDEPRECATED void setEndMaxIfPerformed(IloInt value)
```

This deprecated method sets the maximal end time if the activity is performed. If it is unperformed, its end time is not constrained. This can be used, for example, to specify that the activity has to be unperformed once it reaches a certain tardiness threshold. This method has no effect on activity prototypes; use only with generated activities.

```
public ILOMSDEPRECATED void setEndMinIfPerformed(IloInt value)
```

This deprecated method sets the minimal end time if the activity is performed. If it is unperformed, its end time is not constrained. This can be used, for example, to specify that the activity has to be unperformed once it reaches a certain earliness threshold. This method has no effect on activity prototypes; use only with generated activities.

```
public void setPerformedStatus(IloMSPerformedStatus perfStatus)
```

This method allows setting the performed status for the activity. When the value provided as argument is `IloMSPerformedOrUnperformed`, it means that the scheduling engine must decide to set this activity to either performed or unperformed. The decision taken by the scheduling engine can be accessed by using the appropriate method in the scheduling solution.

**See Also:** IloMSPerformedStatus, IloMSSchedulingSolution

```
public void setSetupState(IloMSIdentifier setupState)
```

This method sets the setup state of the invoking activity. It is equivalent to calling `setSetupState(IloMSNoFeature, setupState)`.

```
public void setSetupState(IloMSIdentifier feature, IloMSIdentifier setupState)
```

This method sets the setup state of the invoking activity for the given setup feature. The parameter `setupState` is the setup state of the invoking activity.

```
public ILOMSDEPRECATED void setStartMaxIfPerformed(IloInt value)
```

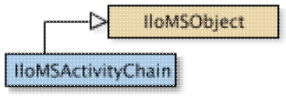
This deprecated method sets the maximal start time if the activity is performed. If it is unperformed, its start time is not constrained. This method can be used, for example, to specify that the activity has to be unperformed once it reaches a certain tardiness threshold. This method has no effect on activity prototypes; use only with generated activities.

```
public ILOMSDEPRECATED void setStartMinIfPerformed(IloInt value)
```

This deprecated method sets the minimal start time if the activity is performed. If it is unperformed, its start time is not constrained. This can be used, for example, to specify that the activity has to be unperformed once it reaches a certain earliness threshold. This method has no effect on activity prototypes; use only with generated activities.

# Class IloMSActivityChain

**Definition file:** ilplant/actchain.h
**Library:** plant




The `IloMSActivityChain` class is used to represent sequences of activities to be executed successively on the same primary resource.
In a manufacturing environment, it is often the case that several activities contributing to the production of the same lot must be performed on the same primary resource, without being interrupted by other activities contributing to the production of other lots. Such is the case, for example, if the process consists of installing a product on a machine and performing several treatments on the product without uninstalling it. The concept of activity chain is introduced for this purpose. Roughly, an activity chain is a sequence of production activities with the same setup state, that must be performed successively on the same primary resource.

An activity chain must meet all the following criteria: 1) The activities must be scheduled in one consistent order on the same primary resource. 2) The activities must be production activities. No setup activities are allowed in the chain. 3) The activities must all have the same setup state. 4) The required capacity on the primary resource must be the same for all activities (A1,A2,A3 will form a rectangle on the primary resource). 5) The resource is used from the beginning to the end of the chain.

The manufacturing scheduling problem would be less complex if an activity chain could be replaced by a unique activity to be performed on the primary resource under consideration. However, there are many cases in which representing the individual activities and linking them in an activity chain might be necessary. For example, consider the case where each individual activity requires different scarce secondary resources that need to be taken into account to build a realistic schedule. Another example is if some of the individual activities cannot be interrupted by "breaks" (such as lunch breaks), but breaks are allowed within the activity chain (that is, between the individual activities).

All the methods of the `IloMSActivityChain` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public void | appendActivity(IloMSActivity activity) |
| public IloBool | canAppend(IloMSActivity activity) const |
| public IloMSActivity | getActivity(IloInt index) const |
| public IloInt | getNumberOfActivities() const |
| public IloInt | getPosition(IloMSActivity activity) const |
| public void | setIdentifier(IloMSIdentifier identifier) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`appendActivity`**`(IloMSActivity activity)`

This method adds a new `activity` at the end of the activity chain. An exception is thrown if the invoking chain already contains an activity with a different setup state.

```
public IloBool canAppend(IloMSActivity activity) const
```

This method returns false if any of the following conditions are true: The invoking activity chain already contains an activity with a different setup state; the invoking chain already contains an activity with a different cleaning status; the activity to be appended has a setup activity and it is not the first activity in the chain.

```
public IloMSActivity getActivity(IloInt index) const
```

This method returns the activity of the invoking chain with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloInt getNumberOfActivities() const
```

This method returns the total number of activities in the invoking chain.

```
public IloInt getPosition(IloMSActivity activity) const
```

This method returns the position of the `activity` into the activity chain

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking activity chain. An exception is thrown if the given `identifier` is already used.

# Class IloMSActivityCompatibilityConstraint

**Definition file:** ilplant/actcompcst.h
**Library:** plant


The class `IloMSActivityCompatibilityConstraint` is used to represent compatibility constraints between two activities.
This constraint enables you to enforce that two given activities will be executed in compatible modes. For example you may want to impose them to execute on the same line or on the same primary resource or on connected primary resources. You may also impose compatibility among the performance statuses such that, for example, if a first activity is performed then a second activity must be unperformed. The actual constraint enforced depends on the type of the `IloMSActivityCompatibilityConstraint` and can be chosen using the enumerated type `IloMSActivityCompatibilityType`. Note that some compatibility types are bidirectional (for example, `IloMSConnectedPrimaryResources`, `IloMSSameLineId`, `IloMSSamePrimaryResourceAndCapacity`); in this case the order of the two activities is not important. Some other compatibility types are not bidirectional (for example, `IloMSPerformedImpliesPerformed`); in this case the order of the two activities matters.

All the methods of the `IloMSActivityCompatibilityConstraint` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSActivity, IloMSMode, IloMSResource, IloMSActivityCompatibilityType

| Method Summary | |
|---|---|
| public IloMSAbstractActivity | getFirstActivity() const |
| public IloMSAbstractActivity | getSecondActivity() const |
| public IloMSActivityCompatibilityType | getType() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloMSAbstractActivity getFirstActivity() const
```

This method returns the first activity affected by the invoking compatibility constraint.

```
public IloMSAbstractActivity getSecondActivity() const
```

This method returns the second activity affected by the invoking compatibility constraint.

```
public IloMSActivityCompatibilityType getType() const
```

This method returns the type of the invoking activity compatibility constraint.

# Class IloMSBatchingEngine

**Definition file:** ilplant/batchengine.h
**Library:** plant


The `IloMSBatchingEngine` class is used to solve batch sizing problems to integrate planning and scheduling. The batching engine is responsible for splitting the production computed by the planning engine (as recipe levels) into production orders (batch-sizing), and is responsible for creating material flow arcs (pegging) that will be used by the scheduling engine. It takes as input an `IloMSPlanningSolution` and creates an `IloMSBatchingSolution` with instances of `IloMSProductionOrder` and `IloMSAbstractMaterialFlowArc`.

**See Also:** IloMSPlanningSolution, IloMSRecipe, IloMSProductionOrder, IloMSAbstractMaterialFlowArc, IloMSPlanningEngine, IloMSSchedulingEngine, IloMSBatchingSolution

| Constructor Summary | |
|---|---|
| `public` | `IloMSBatchingEngine(IloMSModel model, IloMSPlanningSolution planningSolution)` |

| Method Summary | |
|---|---|
| `public IloBool` | `solve()` |

## Constructors

`public` **`IloMSBatchingEngine`**`(IloMSModel model, IloMSPlanningSolution planningSolution)`

This method creates a batching engine for the given `IloMSModel` object, taking a `planningSolution` as input.

## Methods

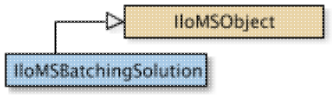
`public IloBool` **`solve`**`()`

This method solves a batching problem. It splits the production levels computed by the planning solution into various production orders and may compute pegging arcs. It must respect the minimal and maximal batch sizes defined in the recipes. It returns true if a solution is found to batch sizing; false otherwise.

**See Also:** IloMSProductionOrder, IloMSAbstractMaterialFlowArc, IloMSRecipe

# Class IloMSBatchingSolution

**Definition file:** ilplant/batchsolution.h
**Library:** plant




The `IloMSBatchingSolution` class is used to represent a batching solution.
All methods of the `IloMSBatchingSolution` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public IloMSAbstractActivity | getAbstractActivityByIdentifier(IloMSIdentifier identifier) const |
| public IloMSActivity | getActivity(IloInt index) const |
| public IloMSActivityChain | getActivityChain(IloInt i) const |
| public IloMSChecker | getChecker() |
| public IloMSSchedulingSolution | getCurrentSchedulingSolution() const |
| public IloMSAbstractMaterialFlowArc | getIncomingMaterialFlowArc(IloMSAbstractMaterialFlowNode node, IloInt index) const |
| public IloMSSchedulingSolution | getInitialSchedulingSolution() const |
| public IloMSAbstractMaterialFlowArc | getMaterialFlowArc(IloInt index) const |
| public IloInt | getNumberOfActivities() const |
| public IloInt | getNumberOfActivityChains() const |
| public IloInt | getNumberOfIncomingMaterialFlowArcs(IloMSAbstractMaterialFlo node) const |
| public IloInt | getNumberOfMaterialFlowArcs() const |
| public IloInt | getNumberOfOutgoingMaterialFlowArcs(IloMSAbstractMaterialFlo node) const |
| public IloInt | getNumberOfProductionOrders(IloMSRecipe recipe) const |
| public IloInt | getNumberOfProductionOrders() const |
| public IloMSAbstractMaterialFlowArc | getOutgoingMaterialFlowArc(IloMSAbstractMaterialFlowNode node, IloInt index) const |
| public IloMSProductionOrder | getProductionOrder(IloMSRecipe recipe, IloInt index) const |
| public IloMSProductionOrder | getProductionOrder(IloInt index) const |
| public IloBool | hasInitialSchedulingSolution() const |
| public IloMSProcurementToDemandArc | newProcurementToDemandArc(IloMSMaterial material, IloNum quantity, IloMSProcurement pred, IloMSDemand succ) |
| public IloMSProcurementToProdArc | newProcurementToProdArc(IloMSMaterial material, IloNum quantity, IloMSProcurement pred, IloMSProductionOrder succ) |
| public IloMSProcurementToStorageArc | newProcurementToStorageArc(IloMSMaterial material, IloNum quantity, IloMSProcurement |

| | |
|---|---|
| | pred) |
| public IloMSProdToDemandArc | newProdToDemandArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder pred, IloMSDemand succ) |
| public IloMSProdToProdArc | newProdToProdArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder pred, IloMSProductionOrder succ) |
| public IloMSProdToStorageArc | newProdToStorageArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder pred) |
| public IloMSProductionOrder | newProductionOrder(IloMSRecipe recipe) |
| public IloMSStorageToDemandArc | newStorageToDemandArc(IloMSMaterial material, IloNum quantity, IloMSDemand succ) |
| public IloMSStorageToProdArc | newStorageToProdArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder succ) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInitialSchedulingSolution(IloMSSchedulingSolution solution) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public IloMSAbstractActivity **getAbstractActivityByIdentifier**(IloMSIdentifier identifier) const

This method returns the abstract activity with the given identifier in the invoking batching solution.

public IloMSActivity **getActivity**(IloInt index) const

This method returns the activity at the given index in the invoking batching solution.

public IloMSActivityChain **getActivityChain**(IloInt i) const

This method returns the *nth* activity chain owned by this solution.

public IloMSChecker **getChecker**()

This method returns the checker of the invoking scheduling solution.

public IloMSSchedulingSolution **getCurrentSchedulingSolution**() const

This method returns the current scheduling solution; note that values returned before solving are no longer valid after solving with success.

```
public IloMSAbstractMaterialFlowArc
getIncomingMaterialFlowArc(IloMSAbstractMaterialFlowNode node, IloInt index) const
```

This method returns the incoming material flow arc of the given material flow node with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSSchedulingSolution getInitialSchedulingSolution() const
```

This method returns the initial scheduling solution.

```
public IloMSAbstractMaterialFlowArc getMaterialFlowArc(IloInt index) const
```

This method returns the material flow arc with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloInt getNumberOfActivities() const
```

This method returns the number of activities in the invoking batching solution.

```
public IloInt getNumberOfActivityChains() const
```

This method returns the number of activity chains owned by this solution.

```
public IloInt getNumberOfIncomingMaterialFlowArcs(IloMSAbstractMaterialFlowNode
node) const
```

This method returns the number of incoming material flow arcs of the given material flow node in the invoking batching solution; that is, the number of material flow arcs for which the given material flow node is the "successor" material flow node.

```
public IloInt getNumberOfMaterialFlowArcs() const
```

This function returns the number of existing material flow arcs.

```
public IloInt getNumberOfOutgoingMaterialFlowArcs(IloMSAbstractMaterialFlowNode
node) const
```

This method returns the number of outgoing material flow arcs of the given material flow node in the invoking batching solution; that is, the number of material flow arcs for which the given material flow node is the "predecessor" material flow node.

```
public IloInt getNumberOfProductionOrders(IloMSRecipe recipe) const
```

This method returns the number of production orders of `recipe` in the invoking batching solution.

```
public IloInt getNumberOfProductionOrders() const
```

This method returns the number of production orders in the invoking batching solution. If the solution was created on a model that already contained some production orders, these will be counted as well.

```
public IloMSAbstractMaterialFlowArc
getOutgoingMaterialFlowArc(IloMSAbstractMaterialFlowNode node, IloInt index) const
```

This method returns the outgoing material flow arc of the given material flow node with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSProductionOrder getProductionOrder(IloMSRecipe recipe, IloInt index)
const
```

This method returns the production orders of the given `recipe` with the given `index` in the invoking batching solution.

```
public IloMSProductionOrder getProductionOrder(IloInt index) const
```

This method returns the production orders with the given `index` in the invoking batching solution.

```
public IloBool hasInitialSchedulingSolution() const
```

This method returns true if there is an initial scheduling solution.

```
public IloMSProcurementToDemandArc newProcurementToDemandArc(IloMSMaterial
material, IloNum quantity, IloMSProcurement pred, IloMSDemand succ)
```

This method creates and returns a new material flow arc between the procurement and the demand.

The argument `quantity` is indicative in this instance and is not taken into account in the optimization.

```
public IloMSProcurementToProdArc newProcurementToProdArc(IloMSMaterial material,
IloNum quantity, IloMSProcurement pred, IloMSProductionOrder succ)
```

This method creates and returns a new material flow arc between the procurement and the production order.

```
public IloMSProcurementToStorageArc newProcurementToStorageArc(IloMSMaterial
material, IloNum quantity, IloMSProcurement pred)
```

This method creates and returns a new material flow arc between the procurement and the stock.

```
public IloMSProdToDemandArc newProdToDemandArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder pred, IloMSDemand succ)
```

This method creates and returns a new material flow arc between the predecessor production order `pred` and the successor demand `succ`.

The `quantity` is in this version indicative and not taken into account in the optimization.

```
public IloMSProdToProdArc newProdToProdArc(IloMSMaterial material, IloNum quantity,
IloMSProductionOrder pred, IloMSProductionOrder succ)
```

This method creates and returns a new material flow arc between the predecessor production order `pred` and the successor production order `succ`. This introduces an implicit precedence constraint between a producer activity of `pred` producing the `material`, and a consumer activity of `succ` consuming the `material`.

```
public IloMSProdToStorageArc newProdToStorageArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder pred)
```

This method creates and returns a new material flow arc between the predecessor production order `pred` and the stock.

```
public IloMSProductionOrder newProductionOrder(IloMSRecipe recipe)
```

This function creates and returns a new production order on the invoking batching solution. The production order implements the `recipe` and the default batch size is 1.

```
public IloMSStorageToDemandArc newStorageToDemandArc(IloMSMaterial material, IloNum
quantity, IloMSDemand succ)
```

This method creates and returns a new material flow arc between the stock and the demand `succ`.

```
public IloMSStorageToProdArc newStorageToProdArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder succ)
```

This method creates and returns a new material flow arc between the stock and the production order `succ`.

```
public void setIdentifier(IloMSIdentifier identifier)
```

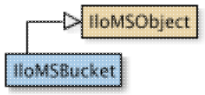
This modifier associates an `identifier` with the invoking batching solution. An exception is thrown if the given `identifier` is already used.

```
public void setInitialSchedulingSolution(IloMSSchedulingSolution solution)
```

This method specifies an initial scheduling solution.

# Class IloMSBucket

**Definition file:** ilplant/bucket.h
**Library:** plant




The `IloMSBucket` class is used to represent time buckets in planning.
Typical time buckets are months, weeks, or days, and are used by the planning engine to globally check constraints (as opposed to the scheduling engine which checks constraints at each time unit). Each bucket has a start time and an end time. The start time is included in the bucket and the end time is excluded.

Two consecutive buckets are usually such that the end time of the first bucket equals the start time of the second bucket. Time buckets are owned by a bucket sequence.

**See Also:** IloMSBucketSequence

| Method Summary | |
|---|---|
| public IloNum | getBucketRatio(IloInt start, IloInt end) const |
| public IloMSBucketSequence | getBucketSequence() |
| public IloInt | getDuration() const |
| public IloInt | getEndTime() const |
| public IloInt | getStartTime() const |
| public void | setEndTime(IloInt endTime) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setStartTime(IloInt startTime) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloNum` **`getBucketRatio`**`(IloInt start, IloInt end) const`

This method returns the intersection duration between the bucket and the interval `[start;end)` divided by `(end - start)`.

`public IloMSBucketSequence` **`getBucketSequence`**`()`

This method returns the bucket sequence that owns this bucket.

`public IloInt` **`getDuration`**`() const`

This method returns the duration of the bucket.

```
public IloInt getEndTime() const
```

This method returns the end time of the bucket.

```
public IloInt getStartTime() const
```

This method returns the start time of the bucket.

```
public void setEndTime(IloInt endTime)
```

This method sets the end time of the invoking bucket.

```
public void setIdentifier(IloMSIdentifier identifier)
```

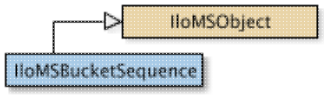
This modifier associates an identifier with the invoking bucket.

```
public void setStartTime(IloInt startTime)
```

This method sets the start time of the invoking bucket.

# Class IloMSBucketSequence

**Definition file:** ilplant/bucketsequence.h
**Library:** plant




The `IloMSBucketSequence` class is used to represent a sequence of time buckets.
Time buckets are stored in a bucket sequence in chronological order. A model can have several bucket sequences. For example, the planning module can use daily buckets, while the GUI displays weekly buckets.

Note that the bucket sequence used for optimization must be more detailed than GUI display bucket sequences. Also, the time boundaries set by the optimized bucket sequence must be respected everywhere; in other words, the buckets in any other sequence must exactly overlap the optimized bucket sequence. For example, one week of the display bucket sequence exactly overlaps seven days of the optimized bucket sequence.

**See Also:** IloMSModel, IloMSBucket

| Method Summary | |
|---|---|
| public void | add(IloMSBucket bucket) |
| public IloMSBucket | findBucket(IloInt time) const |
| public IloInt | findBucketIndex(IloInt time) const |
| public IloMSBucket | getBucket(IloInt index) const |
| public IloInt | getNumberOfBuckets() const |
| public void | setIdentifier(IloMSIdentifier identifier) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`add`**`(IloMSBucket bucket)`

This method adds the `bucket` to the invoking sequence in chronological order.

`public IloMSBucket` **`findBucket`**`(IloInt time) const`

This method finds the bucket that matches the `time` passed as argument.

`public IloInt` **`findBucketIndex`**`(IloInt time) const`

This method finds the bucket index that matches the `time` passed as argument.

`public IloMSBucket` **`getBucket`**`(IloInt index) const`

This method returns the bucket for the `index`.

`public IloInt` **`getNumberOfBuckets`**`() const`

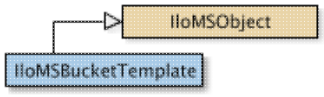
This method returns the number of buckets.

`public void` **`setIdentifier`**`(IloMSIdentifier identifier)`

This modifier associates an `identifier` with the invoking bucket.

# Class IloMSBucketTemplate

**Definition file:** ilplant/buckettemplate.h
**Library:** plant




The `IloMSBucketTemplate` class defines a pattern or mold for bucket generation.

**See Also:** IloMSModel, IloMSBucket, IloMSBucketSequence, IloMSBucketTemplateSequence

| Method Summary | |
|---|---|
| public IloInt | getBucketRank() const |
| public IloMSBucketSequence | getBucketSequence() const |
| public IloMSBucketTemplateSequence | getBucketTemplateSequence() const |
| public IloMSBucketType | getBucketType() const |
| public IloInt | getNumberOfPeriods() const |
| public IloMSBucketPeriodUnit | getPeriodUnit() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloInt` **`getBucketRank`**`() const`

This method returns the rank of this bucket template.

`public IloMSBucketSequence` **`getBucketSequence`**`() const`

This method returns the bucket sequence.

`public IloMSBucketTemplateSequence` **`getBucketTemplateSequence`**`() const`

This method returns the bucket template sequence.

`public IloMSBucketType` **`getBucketType`**`() const`

This method returns the type of this bucket template.

`public IloInt` **`getNumberOfPeriods`**`() const`

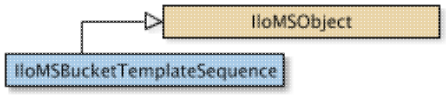
This method returns the number of periods of this bucket template.

```
public IloMSBucketPeriodUnit getPeriodUnit() const
```

This method returns the period unit of this bucket template.

# Class IloMSBucketTemplateSequence

**Definition file:** ilplant/buckettemplatesequence.h
**Library:** plant




The `IloMSBucketTemplateSequence` class is used to connect a set of bucket templates to a given bucket sequence.

**See Also:** IloMSModel, IloMSBucket, IloMSBucketTemplate, IloMSBucketSequence

| Method Summary | |
|---|---|
| `public IloMSBucketSequence` | `getBucketSequence() const` |
| `public IloMSBucketTemplate` | `getBucketTemplate(IloInt index) const` |
| `public IloInt` | `getNumberOfBucketTemplates() const` |
| `public void` | `setBucketSequence(IloMSBucketSequence bucketSequence)` |
| `public void` | `setIdentifier(IloMSIdentifier identifier)` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSBucketSequence` **`getBucketSequence`**`() const`

This method returns the current bucket sequence.

`public IloMSBucketTemplate` **`getBucketTemplate`**`(IloInt index) const`

This method returns the bucket template at the `index` position.

`public IloInt` **`getNumberOfBucketTemplates`**`() const`

This method returns the number of bucket templates.

`public void` **`setBucketSequence`**`(IloMSBucketSequence bucketSequence)`

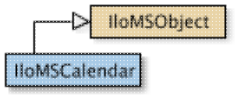
This method associates a bucket sequence to the current object.

`public void` **`setIdentifier`**`(IloMSIdentifier identifier)`

This modifier sets the `identifier` of the current object.

# Class IloMSCalendar

**Definition file:** ilplant/calendar.h
**Library:** plant




The `IloMSCalendar` class is used to represent calendars associated with the different modes of an activity. A calendar consists of a set of time intervals representing breaks, shifts and/or productivity intervals. Calendars can be associated with different activity modes.

Breaks are time intervals when activities cannot be executed because, for example, a resource is not available. A maximal break duration is associated with each activity mode; this means that an activity cannot be interrupted if the break is longer than the maximal break duration. For example, if the maximal break duration is eight hours, the activity cannot be interrupted by a break longer than eight hours. If the break maximal duration is zero, then the activity is not breakable.

A productivity interval specifies the speed at which the activity executes over time. It consists of a list of time intervals to which a strictly positive floating point number called "productivity" is attached. (By default, the productivity is equal to 1.0.) The productivity is used to relate the processing time and the duration of the activity. When the productivity is 0.5, two duration units are needed to execute one unit of processing time. When the productivity is 2.0, one duration unit is needed to execute two units of processing time. When the productivity is greater than 1.0, the processing time of the mode will usually not be fixed; that is, the difference between the maximal processing time and the minimal processing time will usually exceed "productivity - 1.0" to allow an appropriate rounding of both the processing time and the duration of the activity. For example, if in a solution the productivity is 2.0 from the start time to the end time of the activity, and if the duration is 10, then the processing time in this solution is 20. If the minimal and maximal processing times are both equal to 19, the engine regards such a solution as invalid.

The shift intervals specify that some points in time are shift changes. Some activities cannot overlap a shift change, and this is controlled by the shift breakable flag. If the shift breakable flag is set to true, then the activity can be executed during more than one shift (that is, it can overlap a shift change).

If a calendar has no break interval, then it is assumed there are no breaks. If a calendar has no productivity intervals, it is assumed that the productivity is 1.0 at all times. If a calendar has no shift intervals, it is assumed that there are no shifts (or equivalently that all activities executed in the calendar are breakable by shifts).

All the methods of the `IloMSCalendar` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSResource, IloMSMode, IloMSCalendarInterval

| Method Summary | |
|---|---|
| public IloNum | getEnergy(IloInt capacity, IloInt start, IloInt end) |
| public IloMSCalendarInterval | getInterval(IloInt index) const |
| public IloMSCalendarInterval | getNextBreak(IloInt time) const |
| public IloMSCalendarInterval | getNextEndOfShift(IloInt time) const |
| public IloInt | getNumberOfIntervals() const |
| public IloBool | isAdditive() |
| public void | removeInterval(IloMSCalendarInterval interval) |
| public void | setAdditive(IloBool additive) |
| public void | setIdentifier(IloMSIdentifier identifier) |

| Inherited Methods from `IloMSObject` |
| --- |
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

```
public IloNum getEnergy(IloInt capacity, IloInt start, IloInt end)
```

This method returns the energy available on the invoking calendar between `start` and `end`. The energy is computed on the basis of the breaks, capacity and productivity of the invoking calendar. The value `capacity` is used as default for the time intervals on which the capacity is not defined.

```
public IloMSCalendarInterval getInterval(IloInt index) const
```

This method returns the calendar interval with the given `index`.

```
public IloMSCalendarInterval getNextBreak(IloInt time) const
```

This method returns the next calendar interval which is a break, and is after the specified `time`. It returns a null object when no candidate calendar interval is found.

```
public IloMSCalendarInterval getNextEndOfShift(IloInt time) const
```

This method returns the next calendar interval with an end which is an end of shift, and is after the specified `time`. It returns a null object when no candidate calendar interval is found.

```
public IloInt getNumberOfIntervals() const
```

This method returns the number of calendar intervals of the invoking calendar.

```
public IloBool isAdditive()
```

Returns true if and only if the invoking calendar is additive. If the calendar is additive, then each calendar interval provides capacity over a given time interval; if several intervals overlap, then the corresponding capacities shall be added to determine the actual capacity of the resource at a given time. If the calendar is not additive, then each calendar interval limits the capacity available over a given time interval; if several intervals overlap, the most constraining limit applies.

```
public void removeInterval(IloMSCalendarInterval interval)
```

This method removes the given calendar interval from the invoking calendar.

```
public void setAdditive(IloBool additive)
```

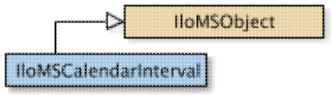
States whether the invoking calendar is additive or not.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking calendar.

# Class IloMSCalendarInterval

**Definition file:** ilplant/calendar.h
**Library:** plant




`IloMSCalendarInterval` is used to represent a calendar as a set of calendar intervals.
The `IloMSCalendarInterval` class is used to represent a calendar as a set of calendar intervals, each interval having its own capacity, productivity, break, and end of shift properties. Note that none of these properties is compulsory as each has a default value. The default capacity is the capacity of the resource to which the calendar is attached; the default productivity is 1.0; and a calendar interval is neither a break nor an end of shift by default.

Calendar intervals can be periodic or aperiodic (without recurring regular intervals). A periodic interval of periodicity **p** > 0 (zero) is implicitly repeated every **p** units of time from a given "period start time" to a given "period end time". An aperiodic interval (with periodicity 0 or zero) occurs only once, from its start time to its end time (although on the GUI Calendars view, you can copy the interval using the "Repeat every" GUI function).

All the methods of the `IloMSCalendarInterval` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSCalendar

| Method Summary | |
|---|---|
| public IloInt | getCapacity() const |
| public IloInt | getEndTime() const |
| public IloInt | getPeriodEndTime() const |
| public IloInt | getPeriodicity() const |
| public IloInt | getPeriodStartTime() const |
| public IloNum | getProductivity() const |
| public IloInt | getStartTime() const |
| public IloBool | hasCapacity() const |
| public IloBool | isBreak() const |
| public IloBool | isEndOfShift() const |
| public void | setBreak(IloBool isBreak) |
| public void | setCapacity(IloInt capacity) |
| public void | setEndOfShift(IloBool isEndOfShift) |
| public void | setEndTime(IloInt time) |
| public void | setPeriodEndTime(IloInt periodEndTime) |
| public void | setPeriodicity(IloInt periodicity) |
| public void | setPeriodStartTime(IloInt periodStartTime) |
| public void | setProductivity(IloNum productivity) |
| public void | setStartTime(IloInt time) |
| public void | unsetCapacity() |

| Inherited Methods from `IloMSObject` |
|---|

display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString

## Methods

public IloInt **getCapacity**() const

This method returns the capacity of the invoking calendar interval. An exception is thrown if the invoking calendar interval does not have a capacity limitation.

public IloInt **getEndTime**() const

This method returns the end time of the invoking calendar interval.

public IloInt **getPeriodEndTime**() const

This method returns the period end time of the invoking calendar interval.

public IloInt **getPeriodicity**() const

This method returns the periodicity of the invoking calendar interval (0 or zero if the interval is not periodic).

public IloInt **getPeriodStartTime**() const

This method returns the period start time of the invoking calendar interval.

public IloNum **getProductivity**() const

This method returns the productivity of the invoking calendar interval. An exception is thrown if the invoking calendar interval includes no productivity information.

public IloInt **getStartTime**() const

This method returns the start time of the invoking calendar interval.

public IloBool **hasCapacity**() const

This method returns true if the invoking interval includes a capacity limitation and false otherwise.

public IloBool **isBreak**() const

This method returns true if the invoking interval is marked as a break and false otherwise.

```
public IloBool isEndOfShift() const
```

This method returns true if the invoking interval is marked as an end of shift and false otherwise.

```
public void setBreak(IloBool isBreak)
```

This modifier states whether the invoking calendar interval is a break.

```
public void setCapacity(IloInt capacity)
```

This modifier sets the capacity of the invoking calendar interval. An exception is thrown if the given `capacity` is strictly negative.

```
public void setEndOfShift(IloBool isEndOfShift)
```

This modifier states whether the invoking calendar interval is an end of shift.

```
public void setEndTime(IloInt time)
```

This modifier sets the end time of the invoking calendar interval.

```
public void setPeriodEndTime(IloInt periodEndTime)
```

This modifier sets the period end time of the invoking calendar interval. An exception is thrown if the given `periodEndTime` is not greater than or equal to the end time of the invoking interval.

```
public void setPeriodicity(IloInt periodicity)
```

This modifier sets the periodicity of the invoking calendar interval. An exception is thrown if the given `periodicity` is strictly negative.

```
public void setPeriodStartTime(IloInt periodStartTime)
```

This modifier sets the period start time of the invoking calendar interval. An exception is thrown if the given `periodStartTime` is not smaller than or equal to the start time of the invoking interval.

```
public void setProductivity(IloNum productivity)
```

This modifier sets the productivity of the invoking calendar interval. An exception is thrown if the given `productivity` is strictly negative.

```
public void setStartTime(IloInt time)
```

This modifier sets the start time of the invoking calendar interval.

```
public void unsetCapacity()
```

This modifier unsets the capacity of the invoking calendar interval. The capacity of a resource over this interval is then equal to the nominal capacity of this resource.

# Class IloMSChecker

**Definition file:** ilplant/checker.h
**Library:** plant


The `IloMSChecker` class is used to check the validity of a model or solution.
All methods of the `IloMSChecker` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSCheckerMessageLevel` | `check()` |
| `public void` | `clear()` |
| `public IloMSCheckerMessageLevel` | `getLevel(const char * type) const` |
| `public IloMSCheckerMessageI *` | `getMessage(IloMSCheckerMessageLevel level, IloInt index) const` |
| `public IloMSCheckerMessageI *` | `getMessage(IloInt index) const` |
| `public IloInt` | `getNumberOfMessages(IloMSCheckerMessageLevel level) const` |
| `public IloInt` | `getNumberOfMessages()` |
| `public IloMSCheckerMessageI *` | `newMessage(const char * messageType)` |
| `public void` | `setLevel(const char * type, IloMSCheckerMessageLevel level)` |

## Methods

`public IloMSCheckerMessageLevel` **`check`**`()`

This method returns `IloMSCheckerOk (IloMSCheckerMessageLevel)` if no error is detected.

**See Also:** IloMSCheckerMessageLevel

`public void` **`clear`**`()`

This method clears the violation messages registered at the last `check()` call.

`public IloMSCheckerMessageLevel` **`getLevel`**`(const char * type) const`

This method returns the level associated with the given message type.

**See Also:** IloMSCheckerMessageLevel

`public IloMSCheckerMessageI *` **`getMessage`**`(IloMSCheckerMessageLevel level, IloInt index) const`

This method returns the message associated with `level` and `index`.

**See Also:** IloMSCheckerMessageLevel

```
public IloMSCheckerMessageI * getMessage(IloInt index) const
```

This method returns the message at `index`.

```
public IloInt getNumberOfMessages(IloMSCheckerMessageLevel level) const
```

This method returns the number of messages at `level`.

**See Also:** IloMSCheckerMessageLevel

```
public IloInt getNumberOfMessages()
```

This method returns the number of messages.

```
public IloMSCheckerMessageI * newMessage(const char * messageType)
```

This method creates a new message. The message type must be known to the checker (see `IloMSChecker.setLevel(type,level)`).

```
public void setLevel(const char * type, IloMSCheckerMessageLevel level)
```

This message sets the level associated with the given message type.

**See Also:** IloMSCheckerMessageLevel

# Class IloMSCheckerMessage

**Definition file:** ilplant/checker.h
**Library:** plant


The `IloMSCheckerMessage` class is used to record the errors found in a model or solution.
All methods of the `IloMSCheckerMessage` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public IloInt | getEndTime() const |
| public const char * | getErrorString() const |
| public IloMSObject | getInvolvedObject(IloInt index) const |
| public const char * | getInvolvedString(IloInt index) const |
| public IloMSCheckerMessageLevel | getLevel() const |
| public IloInt | getNumberOfInvolvedObjects() const |
| public IloInt | getNumberOfInvolvedStrings() const |
| public IloInt | getStartTime() const |
| public const char * | getType() const |
| public void | setEndTime(IloInt endTime) |
| public void | setStartTime(IloInt startTime) |

## Methods

`public IloInt` **`getEndTime`**`() const`

This method returns the time at which the occurrence of the reported issue ends. It may be equal to `IloMSIntPlusInfinity`.

`public const char *` **`getErrorString`**`() const`

This method returns the default error string for the message.

`public IloMSObject` **`getInvolvedObject`**`(IloInt index) const`

This method returns the PPO object involved with the given `index`. An exception is thrown if the given `index` is out of bounds.

`public const char *` **`getInvolvedString`**`(IloInt index) const`

This method returns the string involved with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSCheckerMessageLevel getLevel() const
```

This method returns the level of the message.

**See Also:** IloMSCheckerMessageLevel

```
public IloInt getNumberOfInvolvedObjects() const
```

This method returns the number of objects involved in composing the message.

```
public IloInt getNumberOfInvolvedStrings() const
```

This method returns the number of strings involved in composing the message. It can be used to change the message display.

```
public IloInt getStartTime() const
```

This method returns the time at which the occurrence of the reported issue starts. It may be equal to `IloMSIntMinusInfinity`.

```
public const char * getType() const
```

This method returns the type of the message.

```
public void setEndTime(IloInt endTime)
```

This method sets the time at which the occurrence of the reported issue ends. It can be used to compose a custom message.

```
public void setStartTime(IloInt startTime)
```

This method sets the time at which the occurrence of the reported issue starts. It can be used to compose a custom message.

# Class IloMSCheckForStop

**Definition file:** ilplant/control.h
**Library:** plant


The `IloMSCheckForStop` class is used as the base class for engine stopping callback.
The classes `IloMSCheckForStopI` and `IloMSCheckForStop` enable you to write code that will regularly be executed by the scheduling engine or the planning engine of Plant PowerOps. The defined `IloMSCheckForStop` object must be set to the scheduling engine `IloMSSchedulingEngine` or planning engine `IloMSPlanningEngine` in order for it to be considered.

The virtual method `stop` of the implementation class must be redefined. The scheduling and planning engines guarantee that this function will be called regularly. The engine will stop as soon as the return value of the `stop` function is true.

**See Also:** IloMSDefaultCheckForStop, IloMSPlanningEngine, IloMSSchedulingEngine

| Method Summary | |
|---|---|
| `public IloBool` | `stop() const` |

## Methods

```
public IloBool stop() const
```

This method calls the virtual method of the implementation class `IloMSCheckForStopI`.

# Class IloMSCheckForStopI

**Definition file:** ilplant/control.h
**Library:** plant


The abstract class `IloMSCheckForStopI` is used to enable you to specify a stopping condition for the scheduling engine.
The `IloMSCheckForStopI` and `IloMSCheckForStop` classes enable you to write code that will regularly be executed by the scheduling engine of Plant PowerOps. The defined `IloMSCheckForStop` object must be set to the scheduling engine `IloMSSchedulingEngine` in order for it to be considered.

The virtual method `stop` must be redefined. The scheduling engine guarantees that this function will be called regularly. The engine will stop as soon as the return value of the `stop` function is true.

**See Also:** IloMSSchedulingEngine, IloMSCheckForStop, ILOMSCHECKFORSTOP0, IloMSDefaultCheckForStop

| Constructor Summary | |
|---|---|
| `public` | `IloMSCheckForStopI()` |

| Method Summary | |
|---|---|
| `public virtual void` | `reset()` |
| `public virtual IloBool` | `stop() const` |

## Constructors

`public` **`IloMSCheckForStopI`**`()`

This constructor creates a new instance of the class.

## Methods

`public virtual void` **`reset`**`()`

This virtual method is called to reset the stop value to zero.

`public virtual IloBool` **`stop`**`() const`

This virtual method is called regularly by the scheduling engine. The scheduling engine will stop as soon as this function returns the value true.

# Class IloMSCsvReader

**Definition file:** ilplant/csvreader.h
**Library:** plant


The `IloMSCsvReader` class is used to read Plant PowerOps problems and solutions from CSV files.

| Constructor Summary | |
|---|---|
| `public` | `IloMSCsvReader(IloMSModel plant)` |

| Method Summary | |
|---|---|
| `public IloMSBatchingSolution` | `readBatchingSolution(const char * filename)` |
| `public IloBool` | `readModel(const char * filename)` |
| `public IloMSPlanningSolution` | `readPlanningSolution(const char * filename)` |
| `public IloMSSchedulingSolution` | `readSchedulingSolution(const char * filename)` |
| `public void` | `readSettings(const char * filename)` |

## Constructors

`public` **`IloMSCsvReader`**`(IloMSModel plant)`

This constructor creates a CSV reader for the given `IloMSModel` object.

## Methods

`public IloMSBatchingSolution` **`readBatchingSolution`**`(const char * filename)`

This method reads a batching solution from a file in CSV format. The parameter `filename` is the name of the input file.

`public IloBool` **`readModel`**`(const char * filename)`

This method reads the problem description from the given CSV file. It returns true if no error is found in the file; false otherwise. The parameter `filename` is the name of the input file. An `IloTableNotFoundException` is thrown if the `ACTIVITY` or the `MODE` table is empty.

`public IloMSPlanningSolution` **`readPlanningSolution`**`(const char * filename)`

This method reads a planning solution from a file in CSV format. The parameter `filename` is the name of the input file.

`public IloMSSchedulingSolution` **`readSchedulingSolution`**`(const char * filename)`

This method reads a scheduling solution from a file in CSV format. The parameter `filename` is the name of the input file.

```
public void readSettings(const char * filename)
```

This method reads the settings from a file in CSV format. The old settings are overridden with the new values. Settings that are not specified in the input file but set on the `IloMSModel` will be kept. The parameter `filename` is the name of the input file.

# Class IloMSCsvWriter

**Definition file:** ilplant/csvreader.h
**Library:** plant


The `IloMSCsvWriter` class is used to write Plant PowerOps problems and solutions to CSV files.

| Constructor Summary | |
|---|---|
| `public` | `IloMSCsvWriter(IloMSModel plant)` |

| Method Summary | |
|---|---|
| `public void` | `writeModel(const char * filename, IloBool includeSettings=IloFalse)` |
| `public void` | `writePlanningSolution(const char * filename, IloMSPlanningSolution solution)` |
| `public void` | `writeSettings(const char * filename)` |

## Constructors

`public` **`IloMSCsvWriter`**`(IloMSModel plant)`

This constructor creates a CSV writer for the given `IloMSModel` object.

## Methods

`public void` **`writeModel`**`(const char * filename, IloBool includeSettings=IloFalse)`

This method writes the Plant PowerOps problem description into the given CSV file.

The parameter `filename` is the name of the output file. The parameter `includeSettings` can be used to specify whether the settings will be included in the output file or not.

`public void` **`writePlanningSolution`**`(const char * filename, IloMSPlanningSolution solution)`

This method writes a planning solution from a file in CSV format. The parameter `filename` is the name of the output file.

`public void` **`writeSettings`**`(const char * filename)`

This method writes the settings to a file in CSV format. The parameter `filename` is the name of the output file.

# Class IloMSDate

**Definition file:** ilplant/date.h
**Library:** plant


The class `IloMSDate` is used to contain date information.
This class contains year, month, and day information, as well as the optional hour, minute and seconds. It is always interpreted with respect to Universal Time UTC (previously called GMT).

| Constructor and Destructor Summary | |
|---|---|
| public | `IloMSDate(IloInt year, IloInt month, IloInt dayInMonth, IloInt hour=0, IloInt minute=0, IloInt seconds=0)` |
| public | `IloMSDate(const char * dateString)` |

| Method Summary | |
|---|---|
| public IloInt | `getDayOfMonth() const` |
| public IloMSDay | `getDayOfWeek() const` |
| public IloInt | `getDayOfYear() const` |
| public IloInt | `getHours() const` |
| public IloInt | `getMinutes() const` |
| public IloInt | `getMonth() const` |
| public IloInt | `getSeconds() const` |
| public IloInt | `getYear() const` |

## Constructors and Destructors

```
public IloMSDate(IloInt year, IloInt month, IloInt dayInMonth, IloInt hour=0,
IloInt minute=0, IloInt seconds=0)
```

This constructor creates a date by passing the year, month, and other time units as integers.

```
public IloMSDate(const char * dateString)
```

This constructor creates a date by passing a date string which complies to the ISO 8601 date format of "YYYY-MM-DD HH::MM::SS."

## Methods

```
public IloInt getDayOfMonth() const
```

This method returns the day in the month.

```
public IloMSDay getDayOfWeek() const
```

This method returns the day in the week as an enumerated value: `IloMSSunday`, `IloMSMonday`, and so forth.

`public IloInt` **`getDayOfYear`**`() const`

This method returns the day in the year, for example, `32` for Feb, 1st.

`public IloInt` **`getHours`**`() const`

This method returns the hours.

`public IloInt` **`getMinutes`**`() const`

This method returns the minutes.

`public IloInt` **`getMonth`**`() const`

This method returns the month in the year.

`public IloInt` **`getSeconds`**`() const`

This method returns seconds.

`public IloInt` **`getYear`**`() const`

This method returns the year.

# Class IloMSDefaultCheckForStop

**Definition file:** ilplant/control.h
**Library:** plant


This `IloMSDefaultCheckForStop` class is used to enable you to stop the scheduling engine by changing the internal state of an object of this class.
The `IloMSDefaultCheckForStop` class enables you to stop the engine by changing the internal state of an object of this class via the method `stopIt()`. An object of the class `IloMSDefaultCheckForStop` must be set to the scheduling or planning engine in order for it to be considered. You can create one by calling `newDefaultCheckForStop()` on `IloMSModel`.

An instance of `IloMSDefaultCheckForStop` can be in stopping state or in non-stopping state. The function `stop` returns false if the object is in non-stopping state. It returns true otherwise. The methods `stopIt` and `reset` enable controlling the internal state of the object.

This class redefines the virtual method `stop` of the base (implementation) class `IloMSCheckForStopI`.

**See Also:** IloMSModel, IloMSSchedulingEngine, IloMSPlanningEngine, IloMSCheckForStop

| Method Summary | |
|---|---|
| `public void` | `reset()` |
| `public void` | `stopIt()` |

| Inherited Methods from `IloMSCheckForStop` |
|---|
| `stop` |

## Methods

`public void` **`reset`**`()`

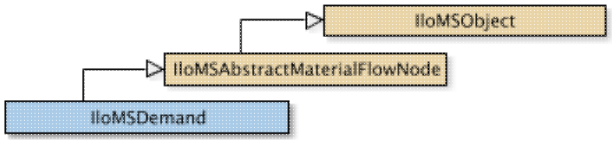
This method resets the state of the object to non-stopping state.

`public void` **`stopIt`**`()`

This method sets the state of the object to stopping state.

# Class IloMSDemand

**Definition file:** ilplant/demand.h
**Library:** plant




The class `IloMSDemand` is used to represent the request for a certain amount of material deliverable in a time window, with an optional preferred due date.
You may use `IloMSDemand` to represent a forecast for demand or actual known customer demands.

For each demand, you can define a due date, an earliest delivery start time, and a latest delivery end time. The due date is the ideal time to deliver the demand; earliness or tardiness penalties may be incurred if the delivery occurs before or after the due date.

Note that a demand may be delivered in several subdeliveries, hence part may be tardy and part may be early. Each subdelivery can be considered as an activity of duration at least 1. For each of these subdeliveries, the inventory of the demanded material is decremented at the subdelivery end time; tardiness is incurred if the subdelivery end time strictly exceeds the due date; earliness is incurred if the subdelivery end time strictly precedes the due date. If the earliest delivery start time is defined, none of the subdeliveries can start before this time. If the latest delivery end time is defined, none of the subdeliveries can end after this time. Note that the earliest delivery start time must be strictly lower than the latest delivery end time, otherwise it is not possible to deliver.

Revenues and nondelivery costs are also associated with demands. For each unit of material that is delivered, the revenue is obtained. For each unit of material that is not delivered, the nondelivery cost is incurred. For example, suppose D is a demand for 100 units of material M that provides a revenue per unit of 100 and a nondelivery cost per unit of 50. Suppose that 70 units are delivered; then the total revenue for this demand is 7000, and the total nondelivery cost 1500.

It is possible to set the nondelivery cost to infinity, thereby making a delivery mandatory; however, do this **only** when you are certain that the demand can be met. Otherwise, PPO may conclude that your problem has no solution.

If neither the revenue nor the nondelivery cost of a demand is defined, or if both are set to 0, then there is no incentive to satisfy the demand and it is very likely that PPO will not satisfy it (except of course if the demand is already pegged with existing procurements or production orders). The model checker of PPO issues a message if both revenue and nondelivery cost are undefined or if both are set to 0.

| Method Summary | |
|---|---|
| public IloInt | getDeliveryEndMax() const |
| public IloInt | getDeliveryStartMin() const |
| public IloMSDueDate | getDueDate() const |
| public IloMSMaterial | getMaterial() const |
| public IloInt | getMaxNumberOfPeggingArcs() const |
| public IloNum | getNonDeliveryVariableCost() const |
| public IloNum | getQuantity() const |
| public IloNum | getQuantityInDisplayUnit() const |
| public IloInt | getRemainingShelfLife() const |
| public IloNum | getRevenue() const |

| | |
|---|---|
| public IloMSStorageUnit | getStorageUnit() const |
| public IloBool | hasDueDate() const |
| public IloBool | hasRemainingShelfLife() const |
| public IloBool | hasStorageUnit() const |
| public IloBool | isFirm() const |
| public IloBool | isPromised() const |
| public void | setDeliveryCompulsory() |
| public void | setDeliveryEndMax(IloInt timeMax) |
| public void | setDeliveryStartMin(IloInt timeMin) |
| public void | setDueDate(IloMSDueDate dueDate) |
| public void | setFirm(IloBool firm) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setMaterial(IloMSMaterial material) |
| public void | setMaxNumberOfPeggingArcs(IloInt maxNumber) |
| public void | setNonDeliveryVariableCost(IloNum nonDeliveryCost) |
| public void | setPromised(IloBool promised) |
| public void | setQuantity(IloNum quantity) |
| public void | setQuantityInDisplayUnit(IloNum quantityInDisplayUnit) |
| public void | setRemainingShelfLife(IloInt value) |
| public void | setRevenue(IloNum revenue) |
| public void | setStorageUnit(IloMSStorageUnit storageUnit) |

| Inherited Methods from **IloMSAbstractMaterialFlowNode** |
|---|
| getCategory, getMaterialFlowNodeType, isFirm, setCategory, setFirm, toDemand, toProductionOrder |

| Inherited Methods from **IloMSObject** |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public IloInt **getDeliveryEndMax**() const

This method returns the latest time at which the delivery of the demand can end. A demand may be satisfied in several subdeliveries, each of which shall be considered as an activity of duration >= 1. None of these subdeliveries can end after the value returned by this method.

public IloInt **getDeliveryStartMin**() const

This method returns the earliest time at which the delivery of the demand can start. A demand may be satisfied in several subdeliveries, each of which shall be considered as an activity of duration >= 1. None of these subdeliveries can start before the value returned by this method.

```
public IloMSDueDate getDueDate() const
```

This method returns the associated due date of the demand object. An exception is thrown if no due date is attached to this demand.

```
public IloMSMaterial getMaterial() const
```

This method returns the requested material.

```
public IloInt getMaxNumberOfPeggingArcs() const
```

This method returns the maximum number of incoming (consumption) pegging arcs.

```
public IloNum getNonDeliveryVariableCost() const
```

This method returns the cost of nondelivery for each unit of demand.

```
public IloNum getQuantity() const
```

This method returns the quantity of the requested material, expressed in the primary unit of the material.

```
public IloNum getQuantityInDisplayUnit() const
```

This method returns the quantity of the procured material, in the material display unit.

```
public IloInt getRemainingShelfLife() const
```

This method returns the minimum remaining shelf life of the material required by this demand.

```
public IloNum getRevenue() const
```

This method returns the revenue per unit of satisfied demand.

```
public IloMSStorageUnit getStorageUnit() const
```

This method returns the storage unit from which the material must be shipped.

```
public IloBool hasDueDate() const
```

This method returns true if the demand has an associated due date demand object.

```
public IloBool hasRemainingShelfLife() const
```

This method returns true if a minimum remaining shelf life is required by this demand.

```
public IloBool hasStorageUnit() const
```

This method returns true if the material requested must be shipped from a specific storage unit.

```
public IloBool isFirm() const
```

This method returns true if the node is firm.

```
public IloBool isPromised() const
```

This method returns the status of the invoking object with respect to the "Available-To-Promise" calculation. The promised demand is used only in the ATP calculation for master production scheduling. Note that a promised status is not enough to make the demand delivery mandatory for the optimizer: An infinite nondelivery cost must be set in order to make the delivery of this demand a hard constraint.

```
public void setDeliveryCompulsory()
```

This method sets the cost of nondelivery for each unit of demand to infinity.

```
public void setDeliveryEndMax(IloInt timeMax)
```

This method sets the latest time at which the delivery of the demand can end. A demand may be satisfied in several subdeliveries, each of which shall be considered as an activity of duration >= 1. None of these subdeliveries can end after `timeMax`.

```
public void setDeliveryStartMin(IloInt timeMin)
```

This method sets the earliest time at which the delivery of the demand can start. A demand may be satisfied in several subdeliveries, each of which shall be considered as an activity of duration >= 1. None of these subdeliveries can start before `timeMin`.

```
public void setDueDate(IloMSDueDate dueDate)
```

This method associates a due date with the demand object.

```
public void setFirm(IloBool firm)
```

This method makes the node firm.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking demand. An exception is thrown if the given `identifier` is already used.

```
public void setMaterial(IloMSMaterial material)
```

This method sets the material requested by this demand.

```
public void setMaxNumberOfPeggingArcs(IloInt maxNumber)
```

This method sets the maximum number of incoming (consumption) pegging arcs.

```
public void setNonDeliveryVariableCost(IloNum nonDeliveryCost)
```

This method sets the cost of nondelivery for each unit of demand.

```
public void setPromised(IloBool promised)
```

This method sets the status of the invoking object with respect to the "Available-To-Promise" (ATP) calculation. The promised demand is used only in the ATP calculation for master production scheduling. Note that a promised status is not enough to make the demand delivery mandatory for the optimizer: An infinite nondelivery cost must be set in order to make the delivery of this demand a hard constraint.

```
public void setQuantity(IloNum quantity)
```

This method sets the quantity of material requested by this demand, expressed in the primary unit of the material.

```
public void setQuantityInDisplayUnit(IloNum quantityInDisplayUnit)
```

This method sets the quantity of material requested by this demand. The provided quantity is expressed in the display unit of the material.

```
public void setRemainingShelfLife(IloInt value)
```

This method sets the minimum remaining shelf life of the material required by this demand.

```
public void setRevenue(IloNum revenue)
```

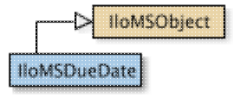
This method sets the revenue per unit of satisfied demand.

```
public void setStorageUnit(IloMSStorageUnit storageUnit)
```

This method sets the storage unit from which the material must be shipped.

# Class IloMSDueDate

**Definition file:** ilplant/duedate.h
**Library:** plant




The `IloMSDueDate` class is used to represent due date objects.
A due date object can be attached either to a demand or an activity. If attached to an activity, the due date can be the activity start time, end time, or to a time point between the start and the end of the activity.

Four cost penalties are defined for a given due date. The *earliness fixed cost* is a fixed price to be paid if the activity is early. The *tardiness fixed cost* is a fixed price to be paid if the activity is tardy. The *earliness variable cost* is the price to be paid (in addition to the earliness fixed cost) per time unit that the activity is early. The *tardiness variable cost* is the price to be paid (in addition to the tardiness fixed cost) per time unit that the activity is tardy.

For example, suppose the due date for the end time of an activity is 1000, the earliness fixed cost is 10, the earliness variable cost is 2.0, the tardiness fixed cost is 200, and the tardiness variable cost is 5.0. If the activity ends at time 995, the earliness cost is 10 + 2.0 * (1000 - 995) = 20.0, and the tardiness cost is zero. If the activity ends at time 1010, the tardiness cost is 200 + 5.0 * (1010 - 1000) = 250.0, and the earliness cost is zero. And, of course, if the activity ends exactly at time 1000, both the earliness and the tardiness costs are zero.

Note that if the due date is attached to a demand, then both variable and fixed costs are multiplied by the quantity of material that is delivered before or after the due date.

By default, the earliness and tardiness costs are equal to 0. The `setEarlinessFixedCost`, `setTardinessFixedCost`, `setEarlinessVariableCost`, and `setTardinessVariableCost` methods must be used to override this default. Otherwise, there is no penalty for delivering the demand (or performing the activity) early if the earliness fixed and variable costs are null, and there is no penalty for delivering the demand late if the tardiness fixed and variable costs are null.

All the methods of the `IloMSDueDate` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSDemand, IloMSAbstractActivity

| Method Summary | |
|---|---|
| `public IloInt` | `getDueTime() const` |
| `public IloNum` | `getEarlinessCost(IloInt startTime, IloInt endTime) const` |
| `public IloNum` | `getEarlinessFixedCost() const` |
| `public IloNum` | `getEarlinessVariableCost() const` |
| `public IloNum` | `getStartEndCoefficient() const` |
| `public IloNum` | `getTardinessCost(IloInt startTime, IloInt endTime) const` |
| `public IloNum` | `getTardinessFixedCost() const` |
| `public IloNum` | `getTardinessVariableCost() const` |
| `public void` | `setEarlinessFixedCost(IloNum cost)` |
| `public void` | `setEarlinessVariableCost(IloNum cost)` |
| `public void` | `setStartEndCoefficient(IloNum coefficient)` |
| `public void` | `setTardinessFixedCost(IloNum cost)` |

| public void | setTardinessVariableCost(IloNum cost) |
|---|---|

| Inherited Methods from IloMSObject |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public IloInt **getDueTime**() const

This method returns the time at which either the demand or the start or the end of the activity is due.

public IloNum **getEarlinessCost**(IloInt startTime, IloInt endTime) const

This method returns the value of the earliness cost incurred by an activity starting at the given `startTime` and ending at the given `endTime`.

public IloNum **getEarlinessFixedCost**() const

This method returns the fixed cost used to compute the earliness cost if early.

public IloNum **getEarlinessVariableCost**() const

This method returns the variable cost used to compute the earliness cost if early.

public IloNum **getStartEndCoefficient**() const

This method returns a coefficient stating if the due date applies to the start (0.0) or to the end (1.0) of the **activity**. Intermediate values can be used to specify intermediate points between the start time and the end time of the activity. For example, 0.4 means that the due date applies to start + 0.4 (end - start) = 0.6 start + 0.4 end. The default returned value is equal to 1.0. This method applies **only** to due dates associated with activities; it does not apply to due dates associated with demands.

public IloNum **getTardinessCost**(IloInt startTime, IloInt endTime) const

This method returns the value of the tardiness cost incurred by an activity starting at the given `startTime` and ending at the given `endTime`.

public IloNum **getTardinessFixedCost**() const

This method returns the fixed cost used to compute the tardiness cost if tardy.

public IloNum **getTardinessVariableCost**() const

This method returns the variable cost used to compute the tardiness cost if tardy.

```
public void setEarlinessFixedCost(IloNum cost)
```

This method sets the earliness fixed cost of the invoking due date object to the given `cost`.

If the due date is attached to a demand, this fixed cost will be multiplied by the quantity of material delivered early.

An exception is thrown if the given `cost` is strictly negative.

```
public void setEarlinessVariableCost(IloNum cost)
```

This method sets the earliness variable cost of the invoking due date object to the given `cost`.

If the due date is attached to a demand, this variable cost will be multiplied by the quantity of material delivered early and by the difference between the due time and the material delivery time. If the due date is attached to an activity, this variable cost will be multiplied by the difference between the due time and (start + StartEndCofficient * (end - start)).

An exception is thrown if the given `cost` is strictly negative.

```
public void setStartEndCoefficient(IloNum coefficient)
```

This method sets the coefficient stating if the due date applies to the start (0.0) or to the end (1.0) of the activity. Default is 1.0. This method applies **only** to due dates associated with activities; it does not apply to due dates associated with demands.

An exception is thrown if the given `coefficient` is not between 0.0 and 1.0.

```
public void setTardinessFixedCost(IloNum cost)
```

This method sets the tardiness fixed cost of the invoking due date object to the given `cost`.

If the due date is attached to a demand, this fixed cost will be multiplied by the quantity of material delivered late.

An exception is thrown if the given `cost` is strictly negative.

```
public void setTardinessVariableCost(IloNum cost)
```

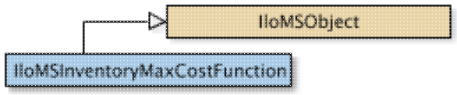
This method sets the tardiness variable cost of the invoking due date object to the given `cost`.

If the due date is attached to a demand, this variable cost will be multiplied by the quantity of material delivered late and by the difference between the material delivery time and the due time. If the due date is attached to an activity, this variable cost will be multiplied by the difference between the due time and (start + StartEndCofficient * (end - start)).

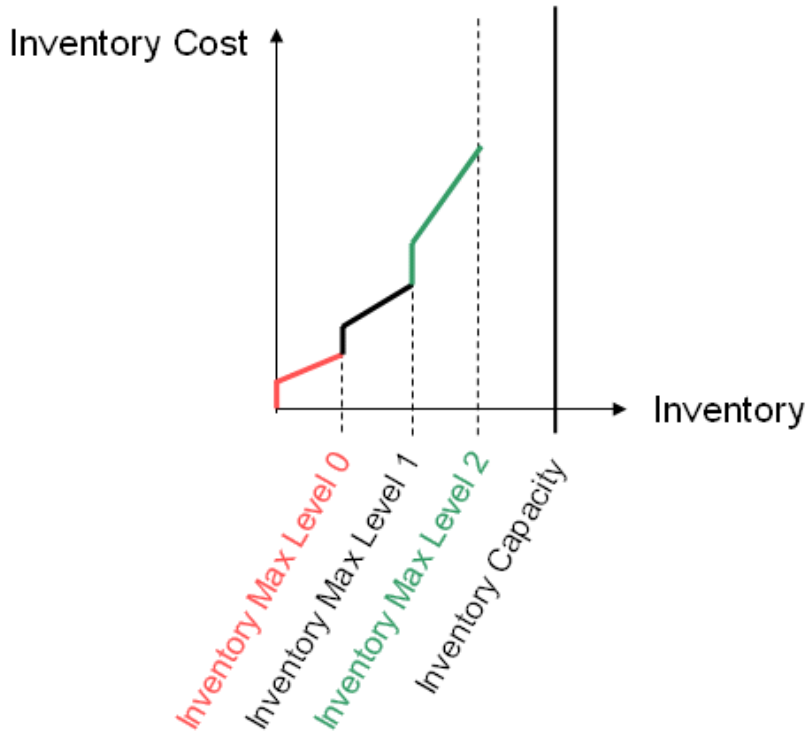
An exception is thrown if the given `cost` is strictly negative.

# Class IloMSInventoryMaxCostFunction

**Definition file:** ilplant/costfunctions.h
**Library:** plant




The class `IloMSInventoryMaxCostFunction` is used to evaluate the cost of violating the maximal inventory over time.
An instance of `IloMSInventoryMaxCostFunction` is used to represent a piecewise or stepwise linear cost function to evaluate the cost of maintaining inventory over time. This feature is only used by the planning engine.




**See Also:** IloMSRecipe, IloMSMaterial

| Method Summary | |
|---|---|
| public IloNum | getFixedCost(IloInt level) const |
| public IloNum | getInventoryMax(IloInt level) const |
| public IloInt | getNumberOfLevels() |
| public IloNum | getVariableCost(IloInt level) const |
| public IloBool | isGeneratedFromDaysOfSupply() const |
| public void | setFixedCost(IloInt level, IloNum value) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInventoryMax(IloInt level, IloNum value) |

| | |
|---|---|
| public void | setVariableCost(IloInt level, IloNum value) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloNum` **`getFixedCost`**`(IloInt level) const`

This method returns the fixed cost incurred to enter this level of inventory; that is, the fixed cost incurred when the maximal inventory of the previous level is exceeded.

`public IloNum` **`getInventoryMax`**`(IloInt level) const`

This method returns the amount of inventory up to which this level applies.

`public IloInt` **`getNumberOfLevels`**`()`

This modifier returns the number of levels associated with the invoking function.

`public IloNum` **`getVariableCost`**`(IloInt level) const`

This method returns the slope of the piecewise linear cost function for this level of inventory. That is, it returns the variable cost for inventory included in this level (from the maximal inventory of the previous level to the maximal inventory of this level).

`public IloBool` **`isGeneratedFromDaysOfSupply`**`() const`

This method returns true if this inventory function has been generated from the days of supply values. Days of supply are transformed by PPO to a set of cost functions.

`public void` **`setFixedCost`**`(IloInt level, IloNum value)`

This method sets the fixed cost value incurred to enter this level of inventory; that is, the fixed cost when the maximal inventory of the previous level is exceeded.

`public void` **`setIdentifier`**`(IloMSIdentifier identifier)`

This modifier associates an `identifier` with the invoking function.

`public void` **`setInventoryMax`**`(IloInt level, IloNum value)`

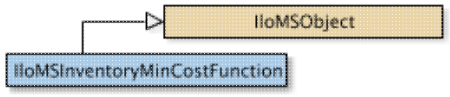
This method sets the amount of inventory value up to which this level applies.

```
public void setVariableCost(IloInt level, IloNum value)
```

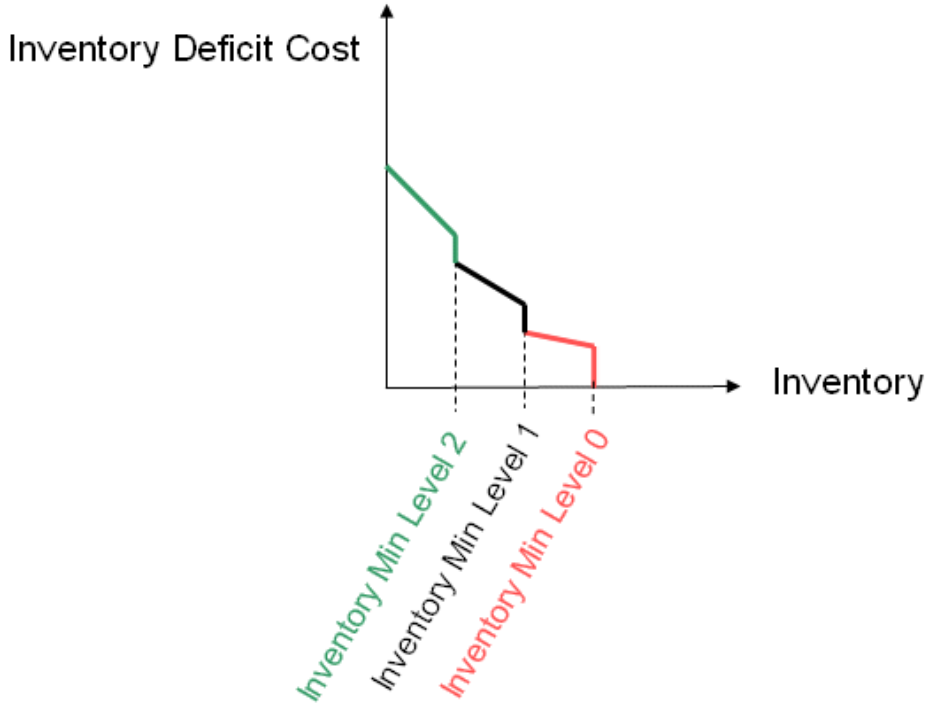
This method sets the slope of the piecewise linear cost function for this level of inventory. That is, it sets the variable cost for inventory included in this level (from the maximal inventory of the previous level to the maximal inventory of this level).

# Class IloMSInventoryMinCostFunction

**Definition file:** ilplant/costfunctions.h
**Library:** plant




The class `IloMSInventoryMinCostFunction` is used to evaluate the cost of violating the minimal inventory (creating an inventory deficit) over time.
An instance of `IloMSInventoryMinCostFunction` is used to represent a piecewise or stepwise linear cost function for penalizing inventory deficits. This feature is only used by the planning engine.




**See Also:** IloMSRecipe, IloMSMaterial

| Method Summary | |
|---|---|
| public IloNum | getFixedCost(IloInt level) const |
| public IloNum | getInventoryMin(IloInt level) const |
| public IloInt | getNumberOfLevels() |
| public IloNum | getVariableCost(IloInt level) const |
| public IloBool | isGeneratedFromDaysOfSupply() const |
| public void | setFixedCost(IloInt level, IloNum value) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInventoryMin(IloInt level, IloNum value) |
| public void | setVariableCost(IloInt level, IloNum value) |

| Inherited Methods from `IloMSObject` |
| --- |
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloNum` **`getFixedCost`**`(IloInt level) const`

This method returns the fixed cost incurred if the inventory falls below the value of minimal inventory of the given `level`.

`public IloNum` **`getInventoryMin`**`(IloInt level) const`

This method returns the minimal inventory for this level.

`public IloInt` **`getNumberOfLevels`**`()`

This modifier returns the number of levels associated with the invoking function.

`public IloNum` **`getVariableCost`**`(IloInt level) const`

This method returns the absolute value of the negative slope of the piecewise linear cost function for an inventory deficit occurring with this level; that is, the variable cost from below the minimal inventory for this level down to the minimal inventory of the next level (0 if undefined).

`public IloBool` **`isGeneratedFromDaysOfSupply`**`() const`

This method returns true if this inventory function has been generated from the days of supply values. Days of supply are transformed by PPO to a set of cost functions.

`public void` **`setFixedCost`**`(IloInt level, IloNum value)`

This method sets the fixed cost incurred if the inventory falls below the minimal inventory for the given level.

`public void` **`setIdentifier`**`(IloMSIdentifier identifier)`

This modifier associates an `identifier` with the invoking function.

`public void` **`setInventoryMin`**`(IloInt level, IloNum value)`

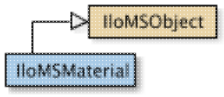
This method sets the minimal amount of inventory for this level.

```
public void setVariableCost(IloInt level, IloNum value)
```

This method sets the absolute value of the negative slope of the piecewise linear cost function for a violation of the minimal inventory of this level; that is, from the minimal inventory of this level down to the minimal inventory of the next level (0 if undefined).

# Class IloMSMaterial

**Definition file:** ilplant/material.h
**Library:** plant




The `IloMSMaterial` class represents Stock-Keeping Units; it is used to represent finished products, raw materials, or intermediates.
A material may be consumed or produced by the activity prototypes of recipes.

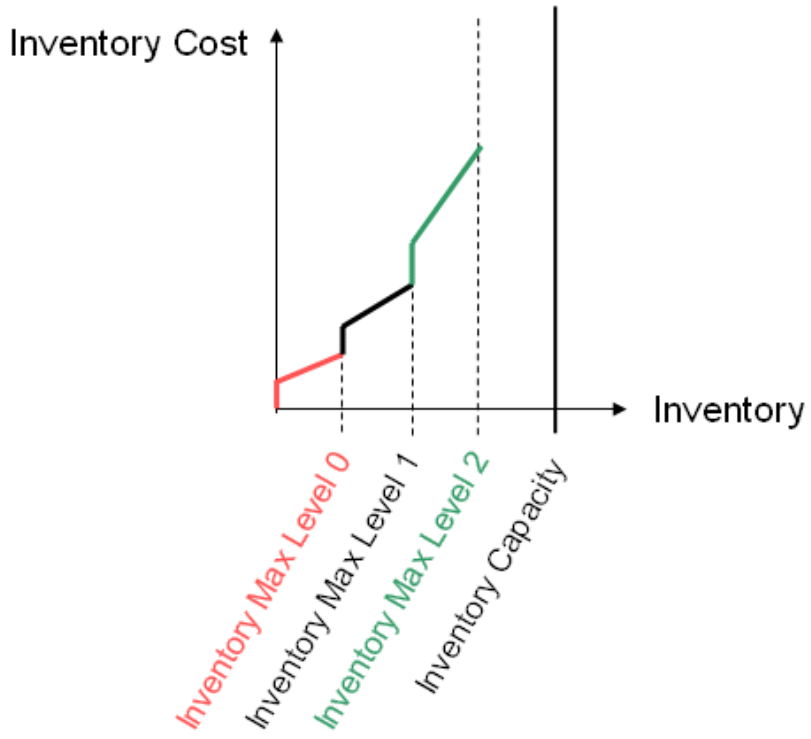
A material may have unique characteristics of size, style, and inventory. For example, consider two models of white shirts that differ in size or style but that are located in the same warehouse; each model of shirt would be a different instance of `IloMSMaterial`. Similarly, the same model of shirt located in two different inventories would also be represented by two different `IloMSMaterial` instances. In this latter case, the separate `IloMSMaterial` instances can be associated with an `IloMSMaterialFamily` to group the results from the different warehouses in reports and graphs.

**Units of Measure**

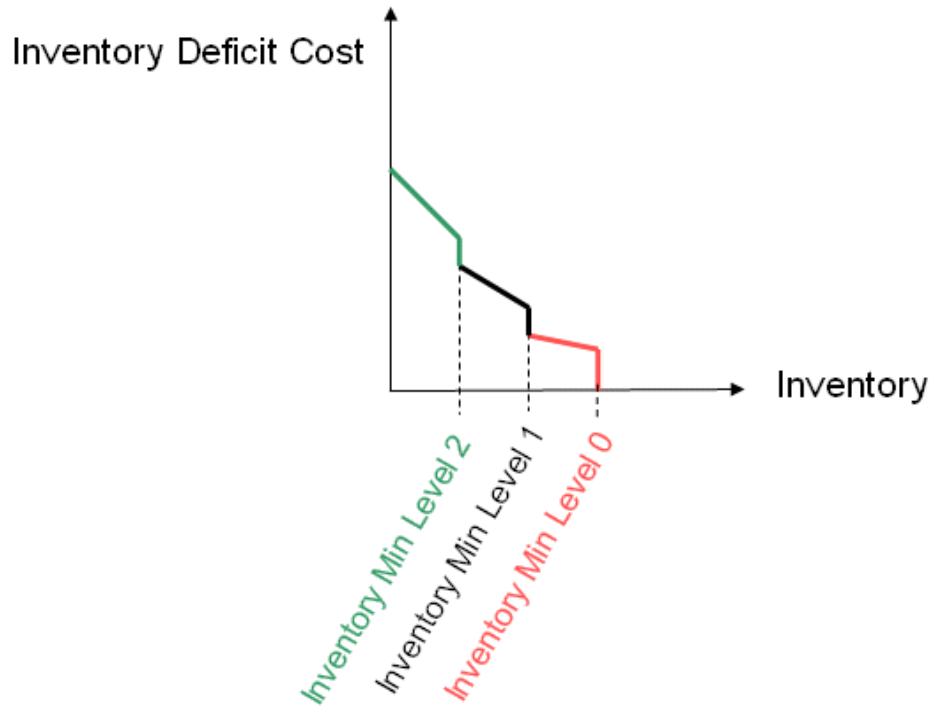
You can define various units of measurement for a material. The **primary** unit is used for modeling and computation to measure the material quantity from recipe production and/or consumption, for determination of storage quantities, and so forth. You can define **secondary** alternative units; each secondary unit has a conversion factor that relates it to the one primary unit. Then you can select one of these secondary units as the **display** unit for display purposes in the Graphical User Interface.

**Material Inventory Costs**

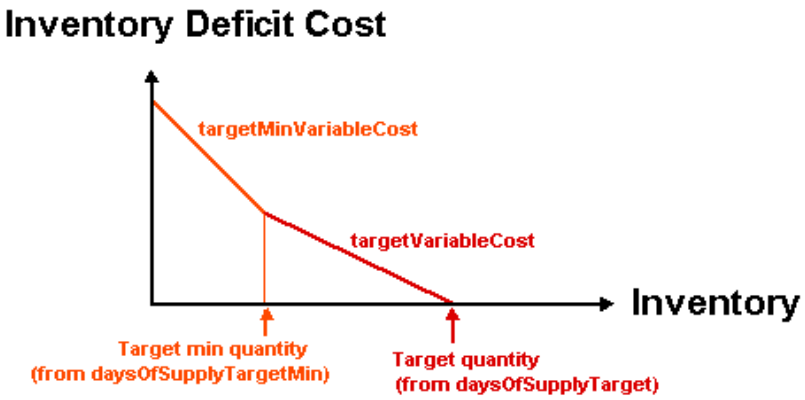
The inventory cost is defined using piecewise or stepwise functions as follows:



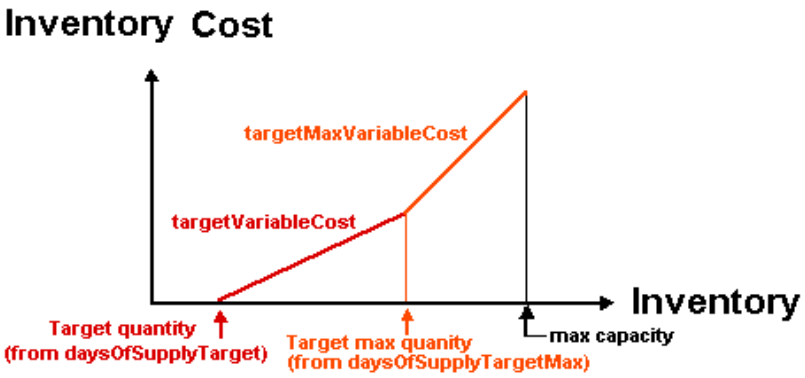
Also for the inventory deficit cost:




You can use the notion of "target minimum days of supply" to define the inventory min cost function. This creates a cost function like the following:




You can use the notion of "target maximum days of supply" to define the inventory cost function. This creates a cost function like the following:




**See Also:** IloMSRecipe, IloMSUnit, IloMSMaterialFamily

| Method Summary | |
|---|---|
| public void | addSecondaryUnit(IloMSUnit secondaryUnit, IloNum numerator, IloNum denominator) |
| public IloNum | convert(IloNum quantity, IloMSUnit fromUnit, IloMSUnit toUnit) const |
| public IloMSMaterial | deepCopy(const char * name, IloMSIdentifier identifier) const |
| public IloInt | getAverageLeadTime() const |
| public IloNum | getComputedInitialQuantity() const |
| public IloNum | getConsumedFixedQuantity(IloInt index) const |
| public IloNum | getConsumedVariableQuantity(IloInt index) const |
| public IloMSAbstractActivity | getConsumingActivityPrototype(IloInt index) const |
| public IloMSRecipe | getConsumingRecipe(IloInt index) const |
| public IloNum | getDaysOfSupplyTarget() const |
| public IloNum | getDaysOfSupplyTargetMax() const |
| public IloNum | getDaysOfSupplyTargetMin() const |
| public IloNum | getDemandVariability() const |
| public IloMSUnit | getDisplayUnit() const |
| public IloNum | getFixedCostOfInventoryMax(IloInt time, IloInt levelNumber) const |
| public IloNum | getFixedCostOfInventoryMin(IloInt time, IloInt levelNumber) const |
| public IloNum | getInventoryCapacity() const |
| public IloNum | getInventoryCostFunctionTimeStep() const |
| public IloNum | getInventoryMax(IloInt time, IloInt levelNumber) const |
| public IloMSInventoryMaxCostFunction | getInventoryMaxCostFunction(IloInt i) const |
| public IloMSInventoryMaxCostFunction | |

| | |
|---|---|
| | getInventoryMaxCostFunctionAtTime(IloInt time) const |
| public IloInt | getInventoryMaxCostFunctionValidityEnd(IloInt i) const |
| public IloInt | getInventoryMaxCostFunctionValidityStart(IloInt i) const |
| public IloNum | getInventoryMin(IloInt time, IloInt levelNumber) const |
| public IloMSInventoryMinCostFunction | getInventoryMinCostFunction(IloInt i) const |
| public IloMSInventoryMinCostFunction | getInventoryMinCostFunctionAtTime(IloInt time) const |
| public IloInt | getInventoryMinCostFunctionValidityEnd(IloInt i) const |
| public IloInt | getInventoryMinCostFunctionValidityStart(IloInt i) const |
| public IloInt | getLeadTimeStdDeviation() const |
| public ILOMSDEPRECATED IloMSMaterialFamily | getMaterialFamily() const |
| public IloInt | getMaturity() const |
| public IloInt | getNumberOfConsumingActivityPrototypes() const |
| public IloInt | getNumberOfConsumingRecipes() const |
| public IloInt | getNumberOfInventoryMaxCostFunctions() const |
| public IloInt | getNumberOfInventoryMinCostFunctions() const |
| public IloInt | getNumberOfLevelsOfInventoryMax(IloInt time) const |
| public IloInt | getNumberOfLevelsOfInventoryMin(IloInt time) const |
| public IloInt | getNumberOfProcurements() const |
| public IloInt | getNumberOfProducingActivityPrototypes() const |
| public IloInt | getNumberOfProducingRecipes() const |
| public IloInt | getNumberOfSecondaryUnits() const |
| public IloInt | getNumberOfStorageUnits() const |
| public IloMSPeggingStrategy | getPeggingStrategy() const |
| public IloMSUnit | getPrimaryUnit() const |
| public IloMSProcurement | getProcurement(IloInt index) const |
| public IloNum | getProducedFixedQuantity(IloInt index) const |
| public IloNum | getProducedVariableQuantity(IloInt index) const |
| public IloMSAbstractActivity | getProducingActivityPrototype(IloInt index) const |
| public IloMSRecipe | getProducingRecipe(IloInt index) const |
| public IloMSUnit | getSecondaryUnit(IloInt index) const |

| | |
|---|---|
| public IloNum | getSecondaryUnitDenominator(IloInt index) const |
| public IloNum | getSecondaryUnitNumerator(IloInt index) const |
| public IloMSServiceLevelType | getServiceLevelType() const |
| public IloInt | getShelfLife() const |
| public IloMSStorageUnit | getStorageUnit(IloInt index) const |
| public IloNum | getTargetMaxVariableCost() const |
| public IloNum | getTargetMinVariableCost() const |
| public IloNum | getTargetServiceLevel() const |
| public IloNum | getTargetVariableCost() const |
| public IloNum | getVariableCostOfInventoryMax(IloInt time, IloInt levelNumber) const |
| public IloNum | getVariableCostOfInventoryMin(IloInt time, IloInt levelNumber) const |
| public IloNum | getWeight() const |
| public IloBool | hasDaysOfSupplyTargetMax() const |
| public IloBool | hasDaysOfSupplyTargetMin() const |
| public IloBool | hasDisplayUnit() const |
| public IloBool | hasInventoryCapacity() const |
| public IloBool | hasInventoryMaxCostFunctionAtTime(IloInt time) const |
| public IloBool | hasInventoryMinCostFunctionAtTime(IloInt time) const |
| public IloInt | hasMaturity() const |
| public IloBool | hasPrimaryUnit() const |
| public IloBool | hasShelfLife() const |
| public void | newMaterialQuality(IloMSQuality quality, IloNum levelMin, IloNum levelMax) |
| public void | removeInventoryMaxCostFunction(IloInt i) |
| public void | removeInventoryMaxCostFunction(IloMSInventoryMaxCostF function, IloInt start, IloInt end) |
| public void | removeInventoryMinCostFunction(IloInt i) |
| public void | removeInventoryMinCostFunction(IloMSInventoryMinCostF function, IloInt start, IloInt end) |
| public void | setAverageLeadTime(IloInt avgLeadTime) |
| public void | setDaysOfSupplyTarget(IloNum days) |
| public void | setDaysOfSupplyTargetMax(IloNum days) |
| public void | setDaysOfSupplyTargetMin(IloNum days) |
| public void | setDemandVariability(IloNum demandVariability) |
| public void | setDisplayUnit(IloMSUnit unit) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInitialQuantity(IloNum quantity) |

| | |
|---|---|
| public void | setInventoryCapacity(IloNum value) |
| public void | setInventoryCostFunctionTimeStep(IloInt timeStep) |
| public void | setInventoryMaxCostFunction(IloMSInventoryMaxCostFunc function, IloInt validityStart, IloInt validityEnd) |
| public void | setInventoryMinCostFunction(IloMSInventoryMinCostFunc function, IloInt validityStart, IloInt validityEnd) |
| public void | setLeadTimeStdDeviation(IloInt leadTimeStdDeviation) |
| public ILOMSDEPRECATED void | setMaterialFamily(IloMSMaterialFamily materialFamily) |
| public void | setMaturity(IloInt maturity) |
| public void | setPeggingStrategy(IloMSPeggingStrategy strategy) |
| public void | setPrimaryUnit(IloMSUnit unit) |
| public void | setServiceLevelType(IloMSServiceLevelType slType) |
| public void | setShelfLife(IloInt shelfLife) |
| public void | setTargetMaxVariableCost(IloNum cost) |
| public void | setTargetMinVariableCost(IloNum cost) |
| public void | setTargetServiceLevel(IloNum serviceLevel) |
| public void | setTargetVariableCost(IloNum cost) |
| public void | setWeight(IloNum weight) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public void **addSecondaryUnit**(IloMSUnit secondaryUnit, IloNum numerator, IloNum denominator)

This method adds a secondary unit of measure in which the quantities of this material can be expressed. Define a conversion factor expressed by a `numerator` and `denominator` to convert from the primary unit to the `secondaryUnit`.

Example: Let the Piece be the primary unit of a material. Imagine a Piece is composed of three Cups, and six Pieces make a Box. Then the conversion factor from primary unit (Piece) to Cup is defined by the ratio 3/1 and the conversion factor from primary unit (Piece) to Box is defined by the ratio 1/6.

public IloNum **convert**(IloNum quantity, IloMSUnit fromUnit, IloMSUnit toUnit) const

This method converts the `quantity` of this material expressed in `fromUnit` to the `toUnit`.

```
public IloMSMaterial deepCopy(const char * name, IloMSIdentifier identifier) const
```

This method clones the invoking material, associating the specified name and identifier to the copy. It also copies the producing recipes. This method adds the clone to the families of the original and duplicates the conversions for the unit of measures.

```
public IloInt getAverageLeadTime() const
```

This method returns the average lead time for the material. Lead time is defined as total time to produce material. This value is used in the service-level-based computation of the safety stock. A value of zero will disable the safety stock calculation.

```
public IloNum getComputedInitialQuantity() const
```

This method returns the quantity of the invoking material that is initially available.

```
public IloNum getConsumedFixedQuantity(IloInt index) const
```

This method returns the fixed quantity of this material consumed by a given activity prototype. An exception is thrown if the given `index` is out of bounds.

```
public IloNum getConsumedVariableQuantity(IloInt index) const
```

This method returns the variable quantity of this material consumed by a given activity prototype. An exception is thrown if the given `index` is out of bounds.

```
public IloMSAbstractActivity getConsumingActivityPrototype(IloInt index) const
```

This method returns a specific activity prototype consumer of this material. An exception is thrown if the given `index` is out of bounds.

```
public IloMSRecipe getConsumingRecipe(IloInt index) const
```

This method returns a specific consumer recipe of this material. An exception is thrown if the given `index` is out of bounds.

```
public IloNum getDaysOfSupplyTarget() const
```

This method returns the targeted ideal days of supply.

```
public IloNum getDaysOfSupplyTargetMax() const
```

This method returns the targeted maximal days of supply.

```
public IloNum getDaysOfSupplyTargetMin() const
```

This method returns the targeted minimal days of supply.

```
public IloNum getDemandVariability() const
```

This method returns the demand variability ratio. This ratio (between 0 and 1) is used by the planning module to compute a safety stock value from the target service level. A value of zero means that demands are certain.

```
public IloMSUnit getDisplayUnit() const
```

This method returns the display unit of material measurement. The material quantity displays in this unit in the Plant PowerOps Application Interface.

```
public IloNum getFixedCostOfInventoryMax(IloInt time, IloInt levelNumber) const
```

This method returns the fixed cost defined at this level of the inventory max cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloNum getFixedCostOfInventoryMin(IloInt time, IloInt levelNumber) const
```

This method returns the fixed cost defined at level `levelNumber` of the inventory min cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloNum getInventoryCapacity() const
```

This method returns the maximal theoretical inventory for the invoking material.

```
public IloNum getInventoryCostFunctionTimeStep() const
```

This method returns the time step used to compute inventory costs.

```
public IloNum getInventoryMax(IloInt time, IloInt levelNumber) const
```

This method returns the inventory max defined at this level of the inventory max cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloMSInventoryMaxCostFunction getInventoryMaxCostFunction(IloInt i) const
```

This method returns the inventory max cost functions with the given index of the invoking material. One can use several inventory levels and associate increasing costs with these levels. This feature is only used by the planning engine.

```
public IloMSInventoryMaxCostFunction getInventoryMaxCostFunctionAtTime(IloInt time)
const
```

This method returns the inventory max cost functions valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloInt getInventoryMaxCostFunctionValidityEnd(IloInt i) const
```

This method returns the end time of the validity interval for the inventory max cost functions with the given index in the invoking material. This feature is only used by the planning engine.

```
public IloInt getInventoryMaxCostFunctionValidityStart(IloInt i) const
```

This method returns the start time of the validity interval for the inventory max cost functions with the given index in the invoking material. This feature is only used by the planning engine.

```
public IloNum getInventoryMin(IloInt time, IloInt levelNumber) const
```

This method returns the inventory min defined at this level of the inventory min cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloMSInventoryMinCostFunction getInventoryMinCostFunction(IloInt i) const
```

This method returns the inventory min cost functions with the given index of the invoking material. One can use several inventory levels and associate increasing costs with those levels. This feature is only used by the planning engine.

```
public IloMSInventoryMinCostFunction getInventoryMinCostFunctionAtTime(IloInt time)
const
```

This method returns the inventory min cost functions valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloInt getInventoryMinCostFunctionValidityEnd(IloInt i) const
```

This method returns the end time of the validity interval for the inventory min cost functions with the given index in the invoking material. This feature is only used by the planning engine.

```
public IloInt getInventoryMinCostFunctionValidityStart(IloInt i) const
```

This method returns the start time of the validity interval for the inventory min cost functions with the given index in the invoking material. This feature is only used by the planning engine.

```
public IloInt getLeadTimeStdDeviation() const
```

This method returns the lead time standard deviation. The lead time is assumed to follow a Gaussian normal law of average `getAverageLeadTime()` and standard deviation as returned by `getLeadTimeStdDeviation()`. This value is used in the computation of service-level safety stocks. A value of zero (the default) means that the lead time is certain.

```
public ILOMSDEPRECATED IloMSMaterialFamily getMaterialFamily() const
```

This deprecated method retrieves the primary family of this material.

```
public IloInt getMaturity() const
```

This method returns the maturation time; that is, the number of time units after production before the material is mature enough to be consumed.

```
public IloInt getNumberOfConsumingActivityPrototypes() const
```

This method returns the number of activity prototypes in all recipes that are consumers of the material.

```
public IloInt getNumberOfConsumingRecipes() const
```

This method returns the number of recipes that are consumers of the material.

```
public IloInt getNumberOfInventoryMaxCostFunctions() const
```

This method returns the number of inventory max cost functions on the resource bucket.

```
public IloInt getNumberOfInventoryMinCostFunctions() const
```

This method returns the number of inventory min cost functions on the resource bucket.

```
public IloInt getNumberOfLevelsOfInventoryMax(IloInt time) const
```

This method returns the number of levels of the inventory max cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloInt getNumberOfLevelsOfInventoryMin(IloInt time) const
```

This method returns the number of levels of the inventory min cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloInt getNumberOfProcurements() const
```

This method returns the number of procurements of the material that exist in the model.

```
public IloInt getNumberOfProducingActivityPrototypes() const
```

This method returns the number of activity prototypes in all recipes that are producers of the material.

```
public IloInt getNumberOfProducingRecipes() const
```

This method returns the number of recipes that are producers of the material.

```
public IloInt getNumberOfSecondaryUnits() const
```

This method returns the number of secondary units in which this material can be expressed.

```
public IloInt getNumberOfStorageUnits() const
```

This method returns the number of storage units in which this material is storable.

```
public IloMSPeggingStrategy getPeggingStrategy() const
```

This method returns the type of pegging strategy to use for material flow arcs.

**See Also:** IloMSPeggingStrategy

```
public IloMSUnit getPrimaryUnit() const
```

This method returns the primary unit used for measuring material quantity. The classes `IloMSMaterialProduction` and `IloMSStorageUnit` use this same unit of measure.

```
public IloMSProcurement getProcurement(IloInt index) const
```

This method returns the procurement of the material with the corresponding `index`.

```
public IloNum getProducedFixedQuantity(IloInt index) const
```

This method returns the fixed quantity of this material produced by a given activity prototype. An exception is thrown if the given `index` is out of bounds.

```
public IloNum getProducedVariableQuantity(IloInt index) const
```

This method returns the variable quantity of this material produced by a given activity prototype. An exception is thrown if the given `index` is out of bounds.

```
public IloMSAbstractActivity getProducingActivityPrototype(IloInt index) const
```

This method returns a specific activity prototype producer of this material. An exception is thrown if the given `index` is out of bounds.

```
public IloMSRecipe getProducingRecipe(IloInt index) const
```

This method returns a specific producer recipe. An exception is thrown if the given `index` is out of bounds.

```
public IloMSUnit getSecondaryUnit(IloInt index) const
```

This method returns the secondary unit with the corresponding index in which this material can be expressed.

```
public IloNum getSecondaryUnitDenominator(IloInt index) const
```

This method returns the denominator of the converting factor between the secondary unit with corresponding index and the primary unit of this material.

```
public IloNum getSecondaryUnitNumerator(IloInt index) const
```

This method returns, for this material, the numerator of the converting factor between the secondary unit (with the corresponding index) and the primary unit.

```
public IloMSServiceLevelType getServiceLevelType() const
```

This method returns the service level type associated with the material.

```
public IloInt getShelfLife() const
```

This method returns the shelf life; that is, the number of time units before the material expires.

```
public IloMSStorageUnit getStorageUnit(IloInt index) const
```

This method returns the storage unit with the corresponding index in which this material can be stored.

```
public IloNum getTargetMaxVariableCost() const
```

This method returns the variable cost incurred per day and per unit of material when inventory is above the maximal target inventory.

```
public IloNum getTargetMinVariableCost() const
```

This method returns the variable cost incurred per day and per unit of material when inventory is below the minimal target inventory.

```
public IloNum getTargetServiceLevel() const
```

This method returns the target service level. This value expresses a minimum value of service level. The precise meaning of the service level depends on the *ServiceLevelType* enumerated value.

**See Also:** IloMSServiceLevelType

```
public IloNum getTargetVariableCost() const
```

This method returns the variable cost incurred per day and per unit of material when inventory is above the minimal target inventory and below maximal target inventory (that is, within the coverage corridor).

```
public IloNum getVariableCostOfInventoryMax(IloInt time, IloInt levelNumber) const
```

This method returns the variable cost defined at this level of the inventory max cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloNum getVariableCostOfInventoryMin(IloInt time, IloInt levelNumber) const
```

This method returns the variable cost defined at this level of the inventory min cost function valid at the given `time` for the invoking material. This feature is only used by the planning engine.

```
public IloNum getWeight() const
```

This method returns the length of one unit of the invoking material.

This method returns the width of one unit of the invoking material.

This method returns the depth of one unit of the invoking material.

This method returns the volume occupied by one unit of the invoking material.

This method returns the weight of one unit of the invoking material.

```
public IloBool hasDaysOfSupplyTargetMax() const
```

This method returns true if a target of maximal days of supply is defined.

```
public IloBool hasDaysOfSupplyTargetMin() const
```

This method returns true if a target of minimal days of supply is defined.

```
public IloBool hasDisplayUnit() const
```

This method returns true if a display unit has been defined on this material.

```
public IloBool hasInventoryCapacity() const
```

This method returns true if the maximal theoretical inventory has been set to a value other than infinity (which is the default inventory capacity).

```
public IloBool hasInventoryMaxCostFunctionAtTime(IloInt time) const
```

This method returns true if an inventory max cost function is valid at the given `time` for the invoking material. It returns false otherwise. This feature is only used by the planning engine.

```
public IloBool hasInventoryMinCostFunctionAtTime(IloInt time) const
```

This method returns true if an inventory min cost function is valid at the given `time` for the invoking material. It returns false otherwise. This feature is only used by the planning engine.

```
public IloInt hasMaturity() const
```

This method returns true if a maturation time is defined for this material.

```
public IloBool hasPrimaryUnit() const
```

This method returns true if a primary unit has been defined on this material.

```
public IloBool hasShelfLife() const
```

This method returns true if a shelf life is defined for this material.

```
public void newMaterialQuality(IloMSQuality quality, IloNum levelMin, IloNum
levelMax)
```

This method adds a `quality` to the invoking material with lower and upper bounds to enforce.

```
public void removeInventoryMaxCostFunction(IloInt i)
```

This method removes the inventory max cost function of the given index for the invoking material. This feature is only used by the planning engine.

```
public void removeInventoryMaxCostFunction(IloMSInventoryMaxCostFunction function,
IloInt start, IloInt end)
```

This method removes the inventory max cost `function` for the invoking material previously defined between `start` and `end`. This feature is only used by the planning engine.

```
public void removeInventoryMinCostFunction(IloInt i)
```

This method removes the inventory min cost function of the given index `i` from the invoking material. This feature is only used by the planning engine.

```
public void removeInventoryMinCostFunction(IloMSInventoryMinCostFunction function,
IloInt start, IloInt end)
```

This method removes the inventory min cost `function` from the invoking material previously defined between `start` and `end`. This feature is only used by the planning engine.

```
public void setAverageLeadTime(IloInt avgLeadTime)
```

This method sets the average lead time of the material. This value is used in service-level computations of safety stocks. The default value is 0.

```
public void setDaysOfSupplyTarget(IloNum days)
```

This method sets the targeted ideal days of supply.

```
public void setDaysOfSupplyTargetMax(IloNum days)
```

This method sets the targeted maximal days of supply.

```
public void setDaysOfSupplyTargetMin(IloNum days)
```

This method sets the targeted minimal days of supply.

```
public void setDemandVariability(IloNum demandVariability)
```

This method sets the demand variability of the material. In service-level computation of safety stocks, each demand of the material is assumed to follow a Gaussian probability law, with the mean value being the demand quantity itself. The standard deviation is computed by multiplying the demand quantity by the material's demand variability ratio.

This value is a ratio between 0 and 1 (strict). A value of zero means the demand has no uncertainty at all, while a value close to 1 means the demand quantity is very poorly known. The default value is 0.

```
public void setDisplayUnit(IloMSUnit unit)
```

This method sets the display unit of material measurement. The material quantity displays in this unit in the Plant PowerOps Application Interface.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking material. An exception is thrown if the given `identifier` is already used.

```
public void setInitialQuantity(IloNum quantity)
```

This method sets the quantity of the invoking material that is initially available independently of any specified material flow arc.

```
public void setInventoryCapacity(IloNum value)
```

This method sets the maximal theoretical inventory for the invoking material as hard constraint.

This is useful in fixed location systems where a SKU is assigned a permanent location and no other items are stored there. In a floating location system where goods are stored wherever there is appropriate space for them, the `setQuantityMax` method of the `IloMSStorageUnit` class must be used instead to limit the warehouse capacity.

```
public void setInventoryCostFunctionTimeStep(IloInt timeStep)
```

This method sets the time step used to compute inventory costs.

```
public void setInventoryMaxCostFunction(IloMSInventoryMaxCostFunction function,
IloInt validityStart, IloInt validityEnd)
```

This method sets the inventory max cost `function` for the invoking material between `validityStart` and `validityEnd`. This feature is only used by the planning engine.

```
public void setInventoryMinCostFunction(IloMSInventoryMinCostFunction function,
IloInt validityStart, IloInt validityEnd)
```

This method sets the inventory min cost `function` for the invoking material between `validityStart` and `validityEnd`. This feature is only used by the planning engine.

```
public void setLeadTimeStdDeviation(IloInt leadTimeStdDeviation)
```

This method sets the lead time standard deviation of the material. For service-level safety stock computations, the lead time is assumed to follow a Gaussian (normal) law with mean value AVERAGE_LEAD_TIME, and standard deviation LEAD_TIME_STD_DEVIATION. The default value is 0, meaning the lead time has no uncertainty.

```
public ILOMSDEPRECATED void setMaterialFamily(IloMSMaterialFamily materialFamily)
```

This deprecated method assigns a primary `family` to this material.

```
public void setMaturity(IloInt maturity)
```

This method sets the maturation time. Maturation time is the minimal number of time units that must elapse after material production completes before that material can be consumed. For example, if the production of a material ends at `t-1`, then at `t + maturity` the material is consumable; at `t + maturity - 1` the material is not yet consumable.

When defining maturity or shelf life, initial quantities must be expressed as procurements received in the past with a clear production date.

```
public void setPeggingStrategy(IloMSPeggingStrategy strategy)
```

This method sets the type of pegging strategy to use for material flow arcs.

**See Also:** IloMSPeggingStrategy

```
public void setPrimaryUnit(IloMSUnit unit)
```

This method is used in the case of a hollow material. It sets the maximal length of a material that would fit in the hollow.

This method is used in the case of a hollow material. It sets the maximal width of a material that would fit in the hollow.

This method is used in the case of a hollow material. It sets the maximal depth of a material that would fit in the hollow.

This method is used in the case of a hollow material. It sets the maximal volume of a material that would fit in the hollow.

This method is used in the case of a hollow material. It sets the maximal weight of a material that would fit in the hollow.

This method sets the primary unit used for measuring material quantity. The classes `IloMSMaterialProduction` and `IloMSStorageUnit` use this same unit of measure.

```
public void setServiceLevelType(IloMSServiceLevelType slType)
```

This method sets the service level type of the material. The default value is `IloMSServiceLevelDisabled`, for which no service level stock computations are performed. You can base the type on the probability of a stock-out event (*alpha* type) or on the demand quantity that is met by stock-on-hand (*beta* type).

The actual level (a numerical value) for this service type is set by the method `IloMSMaterial::setTargetServiceLevel`.

**See Also:** IloMSServiceLevelType

```
public void setShelfLife(IloInt shelfLife)
```

This method is used in the case of a hollow material. It returns the maximal length of a material that would fit in the hollow space.

This method is used in the case of a hollow material. It returns the maximal width of a material that would fit in the hollow space.

This method is used in the case of a hollow material. It returns the maximal depth of a material that would fit in the hollow space.

This method is used in the case of a hollow material. It returns the maximal volume of a material that would fit in the hollow space.

This method is used in the case of a hollow material. It returns the maximal weight of a material that would fit in the hollow space.

This method sets the shelf life; that is, the number of time units before the material expires and is no longer consumable. For example, if the production of a material ends at `t-1`, then at `t - 1 + shelf life` the material is still consumable; at `t + shelf life` the material is no longer consumable.

When defining maturity or shelf life, initial quantities must be expressed as procurements received in the past with a clear production date.

```
public void setTargetMaxVariableCost(IloNum cost)
```

This method sets the variable cost incurred per day and per unit of material when inventory is above the maximal target inventory.

```
public void setTargetMinVariableCost(IloNum cost)
```

This method sets the variable cost incurred per day and per unit of material when inventory is below the minimal target inventory.

```
public void setTargetServiceLevel(IloNum serviceLevel)
```

This method sets the target service level of the material. This value is interpreted as a probability, the semantics of which depends upon the service level type set by `IloMSMaterial::setServiceLevelType`. The service level must be between 0.5 and 1 (strict). The default value is 0.95.

As an example, the default service level of 0.95 means that in combination with an *alpha* service level type, we want to limit the probability of a stock-out event to 5%. In combination with a *beta* or *fill rate* service level type, we want to meet 95% of all demand *quantity*.

**See Also:** IloMSServiceLevelType

```
public void setTargetVariableCost(IloNum cost)
```

This method sets the variable cost incurred per day and per unit of material when inventory is above the minimal target inventory and below maximal target inventory (that is, within the coverage corridor).

```
public void setWeight(IloNum weight)
```

This method sets the length of one unit of the invoking material.

This method sets the width of one unit of the invoking material.

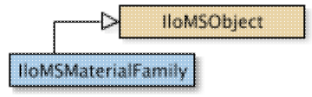
This method sets the depth of one unit of the invoking material.

This method sets the volume occupied by one unit of the invoking material.

This method sets the weight of one unit of the invoking material.

# Class IloMSMaterialFamily

**Definition file:** ilplant/materialfamily.h
**Library:** plant




The `IloMSMaterialFamily` class is used to represent material families. A material may be a member of several families. Families are grouped by their type in the GUI for aggregation purpose.

**See Also:** IloMSMaterial

| Method Summary | |
|---|---|
| public void | add(IloMSMaterial material) |
| public void | display(ostream & stream) const |
| public IloMSMaterialFamilyCardinalityConstraint | getCardinalityConstraint(IloInt index) const |
| public IloMSMaterial | getMaterial(IloInt index) const |
| public IloInt | getNumberOfCardinalityConstraints() const |
| public IloInt | getNumberOfMaterials() const |
| public IloMSIdentifier | getType() const |
| public IloBool | isMember(IloMSMaterial material) const |
| public void | remove(IloMSMaterial material) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setType(IloMSIdentifier type) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`add`**`(IloMSMaterial material)`

This method adds the material to the invoking family.

`public void` **`display`**`(ostream & stream) const`

This method displays the material family in the stream passed as argument.

`public IloMSMaterialFamilyCardinalityConstraint` **`getCardinalityConstraint`**`(IloInt`

```
index) const
```

This method returns the n*th* cardinality constraint attached to the material family.

**See Also:** IloMSMaterialFamilyCardinalityConstraint

```
public IloMSMaterial getMaterial(IloInt index) const
```

This method returns the material with the `index` in the invoking family.

```
public IloInt getNumberOfCardinalityConstraints() const
```

This method returns the number of cardinality constraints attached to the material family.

**See Also:** IloMSMaterialFamilyCardinalityConstraint

```
public IloInt getNumberOfMaterials() const
```

This method returns the number of materials in this family.

```
public IloMSIdentifier getType() const
```

This method retrieves the type of the family.

```
public IloBool isMember(IloMSMaterial material) const
```

This method return true if the material is a member of the invoking family.

```
public void remove(IloMSMaterial material)
```

This method removes the material from the invoking family.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking material family. An exception is thrown if the given `identifier` is already used.

```
public void setType(IloMSIdentifier type)
```

This method registers the invoking family in the collection of families of same type.

# Class IloMSMaterialFamilyCardinalityConstraint

**Definition file:** ilplant/materialfamilycardct.h
**Library:** plant


The `IloMSMaterialFamilyCardinalityConstraint` class is used to represent cardinality constraints on the number of products that can be produced in a given time interval.

| Method Summary | |
|---|---|
| `public IloMSBucketSequence` | `getBucketSequence() const` |
| `public IloInt` | `getCardinalityMax() const` |
| `public IloMSMaterialFamily` | `getMaterialFamily() const` |
| `public void` | `setBucketSequence(IloMSBucketSequence sequence)` |
| `public void` | `setCardinalityMax(IloInt cardMax)` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSBucketSequence` **`getBucketSequence`**`() const`

This method returns the bucket sequence of the invoking cardinality constraint.

`public IloInt` **`getCardinalityMax`**`() const`

This method returns the maximum number of materials that can be produced in each bucket of the constraint's bucket sequence.

**See Also:** IloMSBucketSequence

`public IloMSMaterialFamily` **`getMaterialFamily`**`() const`

This method returns the material family of the invoking constraint.

`public void` **`setBucketSequence`**`(IloMSBucketSequence sequence)`

This method sets the bucket sequence used by the invoking cardinality constraint to count the number of produced materials.

**See Also:** IloMSBucketSequence

```
public void setCardinalityMax(IloInt cardMax)
```

This method sets the maximum number of materials that can be produced in each bucket of the constraint's bucket sequence.

**See Also:** IloMSBucketSequence

# Class IloMSMaterialProduction

**Definition file:** ilplant/materialproduction.h
**Library:** plant


The class `IloMSMaterialProduction` is used to represent all production and consumption of a material.
A positive material production quantity represents production of material; a negative quantity represents material consumption.

| Method Summary | |
|---|---|
| public IloMSAbstractActivity | getActivity() const |
| public IloNum | getComputedQuantity() const |
| public IloNum | getComputedQuantityOnPrototype(IloNum batchSize) const |
| public IloNum | getFixedQuantity() const |
| public IloMSMaterial | getMaterial() const |
| public IloInt | getMaxNumberOfPeggingArcs() const |
| public IloMSMode | getMode() const |
| public IloNum | getRatioMax() const |
| public IloNum | getRatioMin() const |
| public IloMSRecipe | getRecipe() const |
| public IloMSAbstractActivity | getStorageActivity() const |
| public IloMSStorageUnit | getStorageUnit() const |
| public IloInt | getTimeOffset() const |
| public IloNum | getVariableQuantity() const |
| public IloBool | hasActivity() const |
| public IloBool | hasMode() const |
| public IloBool | hasRecipe() const |
| public IloBool | hasStorageActivity() const |
| public IloBool | hasStorageUnit() const |
| public IloBool | isConsuming() const |
| public IloBool | isContinuous() const |
| public IloBool | isProducing() const |
| public void | setContinuous(IloBool continuous) |
| public void | setFixedQuantity(IloNum fixedQuantity) |
| public void | setMaterial(IloMSMaterial material) |
| public void | setMaxNumberOfPeggingArcs(IloInt maxNumber) |
| public void | setRatioMax(IloNum rMax) |
| public void | setRatioMin(IloNum rMin) |
| public void | setStorageActivity(IloMSAbstractActivity storageActivity) |

| | |
|---|---|
| public void | setStorageUnit(IloMSStorageUnit storageUnit) |
| public void | setTimeOffset(IloInt timeOffset) |
| public void | setVariableQuantity(IloNum quantity) |

| Inherited Methods from IloMSObject |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public IloMSAbstractActivity **getActivity**() const

This method returns the associated activity.

public IloNum **getComputedQuantity**() const

This method returns the sum of the fixed and variable quantities produced (or consumed if quantity is negative) for the invoking material production object. It throws an exception if called on a material production prototype.

public IloNum **getComputedQuantityOnPrototype**(IloNum batchSize) const

This method returns the quantity (fixed + variable) generated by an activity that would belong to a production order with the batch size specified as parameter. It throws an exception if called on a generated material production.

public IloNum **getFixedQuantity**() const

This method returns the produced (consumed if the value is negative) fixed quantity of the invoking material production object.

public IloMSMaterial **getMaterial**() const

This method returns the produced or consumed material of the invoking material production object.

public IloInt **getMaxNumberOfPeggingArcs**() const

This method returns the maximum number of incoming (consumption) or outgoing (production) pegging arcs.

public IloMSMode **getMode**() const

This method returns the associated mode.

```
public IloNum getRatioMax() const
```

This method returns the ratio maximum in the recipe for the invoking material production or consumption.

```
public IloNum getRatioMin() const
```

This method returns the ratio minimum in the recipe for the invoking material production or consumption.

Note that if the ratio minimum is different than the ratio maximum in a recipe, then the recipe is considered to be a flexible recipe template for planning. All positive quantities on material productions must then have the same absolute value; likewise, all negative quantities on material productions must have the same absolute value.

```
public IloMSRecipe getRecipe() const
```

This method returns the associated recipe.

```
public IloMSAbstractActivity getStorageActivity() const
```

This method returns the associated storage activity.

```
public IloMSStorageUnit getStorageUnit() const
```

This method returns the associated storage unit.

```
public IloInt getTimeOffset() const
```

This method returns the associated time offset.

```
public IloNum getVariableQuantity() const
```

This method returns the produced (consumed if the value is negative) variable quantity of the invoking material production object.

```
public IloBool hasActivity() const
```

This method returns `true` if the invoking material production has an associated activity.

```
public IloBool hasMode() const
```

This method returns `true` if the invoking material production has an associated mode.

```
public IloBool hasRecipe() const
```

This method returns `true` if the invoking material production has an associated recipe.

```
public IloBool hasStorageActivity() const
```

This method returns `true` if the invoking material production has an associated storage activity.

```
public IloBool hasStorageUnit() const
```

This method returns `true` if the invoking material production has an associated storage unit.

```
public IloBool isConsuming() const
```

This method returns true if the invoking material production object has a negative production quantity.

```
public IloBool isContinuous() const
```

This method returns `true`, if the production/consumption is continuous and `false` if it is discrete.

```
public IloBool isProducing() const
```

This method returns true if the invoking material production object has a positive production quantity.

```
public void setContinuous(IloBool continuous)
```

This method sets the production/consumption to be continuous or discrete.

```
public void setFixedQuantity(IloNum fixedQuantity)
```

This method sets the fixed produced (consumed if `quantity` is negative) fixed quantity of the invoking material production object.

```
public void setMaterial(IloMSMaterial material)
```

This method changes the produced or consumed material of the invoking material production object.

```
public void setMaxNumberOfPeggingArcs(IloInt maxNumber)
```

This method sets the maximum number of incoming (consumption) or outgoing (production) pegging arcs.

```
public void setRatioMax(IloNum rMax)
```

This method sets the ratio maximum in the recipe for the invoking material production or consumption.

```
public void setRatioMin(IloNum rMin)
```

This method sets the ratio minimum in the recipe for the invoking material production or consumption.

Note that if the ratio minimum is different than the ratio maximum in a recipe, then the recipe is considered to be a flexible recipe template for planning. All positive quantities on material productions must then have the same absolute value; likewise, all negative quantities on material productions must have the same absolute value.

```
public void setStorageActivity(IloMSAbstractActivity storageActivity)
```

This method sets the associated storage activity.

```
public void setStorageUnit(IloMSStorageUnit storageUnit)
```

This method sets the associated storage unit.

```
public void setTimeOffset(IloInt timeOffset)
```

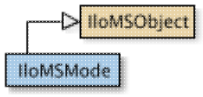
This method sets the associated time offset.

```
public void setVariableQuantity(IloNum quantity)
```

This method sets the produced (consumed if `quantity` is negative) variable quantity of the invoking material production object.

# Class IloMSMode

**Definition file:** ilplant/mode.h
**Library:** plant




The `IloMSMode` class is used to represent the different ways to perform an activity.
A mode is a way of performing an activity. For instance, an activity can have a longer or a shorter processing time depending on the resource on which it is scheduled. The cost of performing an activity on a resource or on another resource may also be different. Some activities can be interrupted by resource breaks, and some cannot, and some activities have limits on the length of breaks allowed.

When using recipes and production orders to express the scheduling problem, it is possible to describe the processing time in two parts: a fixed one independent of the batch size (production order size), and a variable one depending on the batch size. The effective processing time of generated activities is computed as follows: *effective_processing_time* = *prototype_processing_time* + *prototype_variable_processing_time***batch_size*. Note that *prototype_processing_time* may still vary between *prototype_processing_time_min* and *prototype_processing_time_max*.

All the methods of the `IloMSMode` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSAbstractActivity, IloMSCalendar, IloMSRecipe, IloMSProductionOrder

| Method Summary | |
|---|---|
| public IloBool | contains(IloMSResource res) |
| public IloMSAbstractActivity | getActivity() const |
| public IloNum | getBatchSizeMax() const |
| public IloNum | getBatchSizeMin() const |
| public IloInt | getBreakDurationMax() const |
| public IloMSCalendar | getCalendar() const |
| public IloInt | getEndMax() const |
| public IloInt | getEndMin() const |
| public IloNum | getFixedCost() const |
| public IloInt | getFixedProcessingTimeMax() const |
| public IloInt | getFixedProcessingTimeMin() const |
| public IloMSIdentifier | getLineId() const |
| public IloMSMaterialProduction | getMaterialProduction(IloInt index) const |
| public IloInt | getMaxEndDurationInBreak() const |
| public IloInt | getNumber() const |
| public IloInt | getNumberOfMaterialProductions() const |
| public IloInt | getNumberOfResourceConstraints() const |
| public IloInt | getNumberOfSecondaryResourceConstraints() const |
| public IloInt | getPrimaryRequiredCapacity() const |
| public IloMSResource | getPrimaryResource() const |
| public IloMSResourceConstraint | getResourceConstraint(IloInt index) const |

| | |
|---|---|
| public IloInt | getSecondaryRequiredCapacity(IloInt index) const |
| public IloMSResource | getSecondaryResource(IloInt index) const |
| public IloMSResourceConstraint | getSecondaryResourceConstraint(IloInt index) const |
| public IloInt | getStartMax() const |
| public IloInt | getStartMin() const |
| public IloNum | getUnperformedCost() const |
| public IloNum | getUnperformedSetupCost() const |
| public IloInt | getUnperformedSetupTime() const |
| public IloNum | getVariableCost() const |
| public IloNum | getVariableProcessingTime() const |
| public IloBool | hasCalendar() const |
| public IloBool | hasPrimaryResource() const |
| public IloBool | isShiftBreakable() const |
| public IloMSResourceConstraint | newPrimaryResourceConstraint(IloMSResource resource, IloInt capacity) |
| public IloMSResourceConstraint | newSecondaryResourceConstraint(IloMSResource resource, IloInt capacity) |
| public void | setBatchSizeMax(IloNum batchSizeMax) |
| public void | setBatchSizeMin(IloNum batchSizeMin) |
| public void | setBreakDurationMax(IloInt value) |
| public void | setCalendar(IloMSCalendar calendar) |
| public void | setEndMax(IloInt emax) |
| public void | setEndMin(IloInt emin) |
| public void | setFixedCost(IloNum cost) |
| public void | setFixedProcessingTime(IloInt ptime) |
| public void | setFixedProcessingTimeMax(IloInt pmax) |
| public void | setFixedProcessingTimeMin(IloInt pmin) |
| public void | setLineId(IloMSIdentifier id) |
| public void | setMaxEndDurationInBreak(IloInt value) |
| public void | setShiftBreakable(IloBool value) |
| public void | setStartMax(IloInt smax) |
| public void | setStartMin(IloInt smin) |
| public void | setUnperformedCost(IloNum unperformedCost) |
| public void | setUnperformedSetupCost(IloNum setupCost) |
| public void | setUnperformedSetupTime(IloInt setupTime) |
| public void | setVariableCost(IloNum cost) |
| public void | setVariableProcessingTime(IloNum variableProcessingTime) |

| **Inherited Methods from `IloMSObject`** |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloBool contains(IloMSResource res)
```

This predicate is true if this mode contains `res` as a primary or secondary resource.

```
public IloMSAbstractActivity getActivity() const
```

This method returns the activity performed by the invoking mode.

```
public IloNum getBatchSizeMax() const
```

This method returns the additional constraint on maximum batch size for any order using this mode. It restricts the recipe batch size max. A typical application is when using a single multi-mode recipe, to constrain the allowable batch size for orders flowing through a particular path.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

```
public IloNum getBatchSizeMin() const
```

This method returns the additional constraint on minimum batch size for any order using this mode. It restricts the recipe batch size min. A typical application is when using a single multi-mode recipe, to constrain the allowable batch size for orders flowing through a particular path.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

```
public IloInt getBreakDurationMax() const
```

This method returns the maximal break duration over which the activity in the invoking mode cannot be interrupted by a break.

```
public IloMSCalendar getCalendar() const
```

This method returns the calendar associated with the invoking mode. An exception is thrown if the mode has no calendar.

```
public IloInt getEndMax() const
```

This method returns the latest possible end time of the activity in the invoking mode.

```
public IloInt getEndMin() const
```

This method returns the earliest possible end time of the activity in the invoking mode.

```
public IloNum getFixedCost() const
```

This method returns the fixed cost of the invoking mode for an activity prototype. The fixed cost is the part of the cost which is independent of the batch size.

```
public IloInt getFixedProcessingTimeMax() const
```

This method returns the upper bound of the part of the processing time which is independent of the batch size for the invoking mode.

```
public IloInt getFixedProcessingTimeMin() const
```

This method returns the lower bound of the part of the processing time which is independent of the batch size for the invoking mode.

```
public IloMSIdentifier getLineId() const
```

This method returns the line identifier of the invoking mode (by default, the line identifier of the primary resource of the mode, or -1 if the mode has no specific line identifier and no primary resource).

```
public IloMSMaterialProduction getMaterialProduction(IloInt index) const
```

This method returns the material production of the invoking mode with the given `index`.

```
public IloInt getMaxEndDurationInBreak() const
```

This method returns the maximal duration over which the activity in the invoking mode can end in a break.

```
public IloInt getNumber() const
```

This method returns the index of the mode of the activity to which it belongs.

```
public IloInt getNumberOfMaterialProductions() const
```

This method returns the number of material productions associated with the invoking mode.

```
public IloInt getNumberOfResourceConstraints() const
```

This method returns the number of resource constraints associated with the invoking mode (primary + secondary).

```
public IloInt getNumberOfSecondaryResourceConstraints() const
```

This method returns the number of secondary resource constraints associated with the invoking mode.

```
public IloInt getPrimaryRequiredCapacity() const
```

This method returns the required capacity of the primary resource. An exception is thrown if the mode has no primary resource constraint.

```
public IloMSResource getPrimaryResource() const
```

This method returns the primary resource required by the invoking mode.

```
public IloMSResourceConstraint getResourceConstraint(IloInt index) const
```

This method returns the required resource constraint of the invoking mode with the given `index`.

```
public IloInt getSecondaryRequiredCapacity(IloInt index) const
```

This method returns the capacity required on the secondary resource of the invoking mode with the given `index`.

```
public IloMSResource getSecondaryResource(IloInt index) const
```

This method returns the required secondary resource of the invoking mode with the given `index`.

```
public IloMSResourceConstraint getSecondaryResourceConstraint(IloInt index) const
```

This method returns the required secondary resource of the invoking mode with the given `index`.

```
public IloInt getStartMax() const
```

This method returns the latest possible start time of the activity in the invoking mode.

```
public IloInt getStartMin() const
```

This method returns the earliest possible start time of the activity in the invoking mode.

```
public IloNum getUnperformedCost() const
```

This method returns the cost of not performing the activity in the invoking mode.

```
public IloNum getUnperformedSetupCost() const
```

This method returns the setup cost of the activity if the invoking mode has been chosen and the activity is in unperformed status.

```
public IloInt getUnperformedSetupTime() const
```

This method returns the setup time of the activity if the invoking mode has been chosen and the activity is in unperformed status.

```
public IloNum getVariableCost() const
```

This method returns the variable cost of the invoking mode for an activity prototype. The variable cost is the part of the cost which is proportional to the batch size.

```
public IloNum getVariableProcessingTime() const
```

This method returns the variable processing time of an activity prototype. It is used for batch size dependent processing time.

The effective processing time of generated activities is computed as follows: *effective_processing_time* = *prototype_processing_time* + *prototype_variable_processing_time* batch_size*, where *batch_size* is the size of the production order implementing a recipe.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

```
public IloBool hasCalendar() const
```

This method returns true if the invoking mode has a calendar and false otherwise.

```
public IloBool hasPrimaryResource() const
```

This predicate returns true if this mode has a primary resource constraint.

```
public IloBool isShiftBreakable() const
```

This method returns the shift breakable flag for the activity in the invoking mode. An activity that is "shift-breakable" can overlap a shift change. An activity that is not shift-breakable must be completely executed within a shift.

```
public IloMSResourceConstraint newPrimaryResourceConstraint(IloMSResource resource,
IloInt capacity)
```

This method creates a primary resource constraint on the invoking mode if none already exists and the specified resource is not already referred to by another resource constraint.

```
public IloMSResourceConstraint newSecondaryResourceConstraint(IloMSResource
resource, IloInt capacity)
```

This method creates a secondary resource constraint on the invoking mode if the specified resource is not already referred to by another resource constraint.

```
public void setBatchSizeMax(IloNum batchSizeMax)
```

This method enforces that the batch size of any order using this mode is lower than the specified value. It restricts the recipe batch size max. A typical application is when using a single multi-mode recipe, to constrain the allowable batch size for orders flowing through a particular path.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

```
public void setBatchSizeMin(IloNum batchSizeMin)
```

This method enforces that the batch size of any order using this mode is greater than the specified value. It restricts the recipe batch size min. A typical application is when using a single multi-mode recipe, to constrain the allowable batch size for orders flowing through a particular path.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

```
public void setBreakDurationMax(IloInt value)
```

This method sets to the given `value` the maximal break duration over which the activity in the invoking mode cannot be interrupted by a break.

```
public void setCalendar(IloMSCalendar calendar)
```

This method sets the calendar of the invoking mode. If a calendar is not specified, then the mode uses the calendar of its primary resource. Note that breaks and work periods are taken into account on calendars assigned to modes, but resource capacities are not. Calendars on resources do account for capacities. An exception is thrown if a calendar has already been declared for the mode.

```
public void setEndMax(IloInt emax)
```

This method sets the latest possible end time (`emax`) of the activity in the invoking mode.

```
public void setEndMin(IloInt emin)
```

This method sets the earliest possible end time (`emin`) of the activity in the invoking mode.

```
public void setFixedCost(IloNum cost)
```

This method sets the fixed cost of the invoking mode for an activity prototype. The parameter `cost` is the part of the mode cost which is independent of the batch size. An exception is thrown if the given `cost` is negative.

```
public void setFixedProcessingTime(IloInt ptime)
```

This method sets the part of the processing time which is independent of the batch size for the invoking mode.

```
public void setFixedProcessingTimeMax(IloInt pmax)
```

This method sets the upper bound of the part of the processing time which is independent of the batch size for the invoking mode.

```
public void setFixedProcessingTimeMin(IloInt pmin)
```

This method sets the lower bound of the part of the processing time which is independent of the batch size for the invoking mode.

```
public void setLineId(IloMSIdentifier id)
```

This method sets the line identifier of the invoking mode to the given value `id`. An exception is thrown if the given `id` is negative.

```
public void setMaxEndDurationInBreak(IloInt value)
```

This method sets to the given `value` the maximal duration over which the activity in the invoking mode can end in a break.

```
public void setShiftBreakable(IloBool value)
```

This method sets the shift breakable flag for the activity in the invoking mode. An activity that is "shift-breakable" can overlap a shift change. An activity that is not shift-breakable must be completely executed within a shift.

```
public void setStartMax(IloInt smax)
```

This method sets the latest possible start time (`smax`) of the activity in the invoking mode.

```
public void setStartMin(IloInt smin)
```

This method sets the earliest possible start time (`smin`) of the activity in the invoking mode.

```
public void setUnperformedCost(IloNum unperformedCost)
```

This method sets the cost of not performing the activity in the invoking mode. If the activity is a prototype, the effective unperformed cost of generated activities is computed as follows: *effective_unperformed_cost* = *prototype_unperformed_cost***batch_size*, where *batch_size* is the size of the production order implementing a recipe. An exception is thrown if the given `unperformedCost` is negative.

```
public void setUnperformedSetupCost(IloNum setupCost)
```

This method sets to the given value the setup cost of the activity if it is not performed in the invoking mode. The parameter `setupCost` is the setup cost to add to the global setup cost if the invoking mode has been chosen

and the activity is in unperformed status. An exception is thrown if the given `setupCost` is strictly negative.

```
public void setUnperformedSetupTime(IloInt setupTime)
```

This method sets to the given value the setup time of the activity if it is not performed in the invoking mode. The parameter `setupTime` is the setup time to add to the global setup time if the invoking mode has been chosen and the activity is in unperformed status. An exception is thrown if the given `setupTime` is strictly negative.

```
public void setVariableCost(IloNum cost)
```

This method sets the cost of the invoking mode for an activity prototype. The parameter `cost` is the part of the mode cost which is proportional to the batch size. An exception is thrown if the given `cost` is negative.

```
public void setVariableProcessingTime(IloNum variableProcessingTime)
```

This method sets the variable processing time of an activity prototype. It is used for batch size dependent processing time.

The effective processing time of generated activities is computed as follows: *effective_processing_time* = *prototype_processing_time* + *prototype_variable_processing_time* batch_size*, where *batch_size* is the size of the production order implementing a recipe.

**See Also:** IloMSRecipe, IloMSProductionOrder, IloMSActivity

# Class IloMSModel

**Definition file:** ilplant/model.h
**Library:** plant


The `IloMSModel` class gathers the objects that define the manufacturing problem to be solved.
The manufacturing problem may consist of a planning problem and a scheduling problem. The objects that define the planning problem include recipes, materials and demands.

The objects that define the scheduling problem include activities, resources, modes for executing the activities, associated resource constraints, precedence constraints, calendars, and so forth.

The bridge between planning and scheduling is realized via the notions of production orders and material flow arcs and by the description of recipes using activity prototypes.

The `IloMSModel` object is also used to set the weights associated with different optimization criteria and to interact with the optimizer.

All the methods of the `IloMSModel` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSActivity, IloMSResource, IloMSRecipe, IloMSProductionOrder, IloMSMaterial, IloMSAbstractMaterialFlowArc, IloMSDemand

| Constructor Summary | |
|---|---|
| public | IloMSModel() |

| Method Summary | |
|---|---|
| public void | close() |
| public void | commitCurrentBatchingSolution() |
| public IloInt | convertDateToTime(const IloMSDate & date) const |
| public IloMSDate | convertTimeToDate(IloInt time) const |
| public IloMSModel | copy() |
| public void | copyBatchingFrom(IloMSModel origin) |
| public void | copyPlanningSolutionFrom(IloMSModel origin) |
| public void | copySchedulingSolutionFrom(IloMSModel origin) |
| public void | end() |
| public void | generateActivities() |
| public IloMSActivityChain | getActivityChainPrototype(IloInt index) const |
| public IloMSActivityChain | getActivityChainPrototypeByIdentifier(IloMSIdent identifier) const |
| public IloMSAbstractActivity | getActivityPrototypeByIdentifier(IloMSIdentifier identifier) const |

| | |
|---|---|
| public IloInt | getBatchingHorizon() const |
| public IloNum | getBatchingWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getBatchingWeight(IloMSIdentifier criterionId) const |
| public IloMSBucket | getBucket(IloInt index) const |
| public IloMSBucket | getBucketByIdentifier(IloMSIdentifier identifier) const |
| public IloMSBucketSequence | getBucketSequence(IloInt index) const |
| public IloMSCalendar | getCalendar(IloInt index) const |
| public IloMSCalendar | getCalendarByIdentifier(IloMSIdentifier identifier) const |
| public IloMSChecker | getChecker() |
| public IloMSBatchingSolution | getCurrentBatchingSolution() const |
| public IloMSOptimizationProfile | getCurrentOptimizationProfile() const |
| public IloMSPlanningSolution | getCurrentPlanningSolution() const |
| public IloMSSchedulingSolution | getCurrentSchedulingSolution() const |
| public IloMSDate | getDateOrigin() const |
| public static IloInt | getDefaultTimeCheckingTolerance() |
| public IloMSDemand | getDemand(IloInt index) const |
| public IloMSDemand | getDemandByIdentifier(IloMSIdentifier identifier) const |
| public IloMSBucketSequence | getDisplayedBucketSequence() const |
| public IloNum | getElapsedTime() const |
| public IloInt | getEndMax() const |
| public IloNum | getEpsilon() const |
| public IloInt | getIndex(IloMSResource res) const |
| public IloInt | getIndex(IloMSActivity act) const |
| public IloInt | getIntDateOrigin() const |
| public IloMSInventoryMaxCostFunction | getInventoryMaxCostFunction(IloInt index) const |
| public IloMSInventoryMaxCostFunction | getInventoryMaxCostFunctionByIdentifier(IloMSIde identifier) const |
| public IloMSInventoryMinCostFunction | getInventoryMinCostFunction(IloInt index) const |
| public IloMSInventoryMinCostFunction | getInventoryMinCostFunctionByIdentifier(IloMSIde identifier) const |
| public IloInt | getMakespanOrigin() const |
| public IloMSMaterial | getMaterial(IloInt index) const |
| public IloMSMaterial | getMaterialByIdentifier(IloMSIdentifier identifier) const |
| public IloMSMaterialFamily | getMaterialFamily(IloMSIdentifier materialFamilyType, IloInt index) const |
| public IloMSMaterialFamily | getMaterialFamily(IloInt index) const |
| public IloMSMaterialFamily | getMaterialFamilyByIdentifier(IloMSIdentifier |

| | |
|---|---|
| | identifier) const |
| public IloMSIdentifier | getMaterialFamilyType(IloInt index) const |
| public IloMSAbstractMaterialFlowArc | getMaterialFlowArc(IloInt index) const |
| public IloInt | getMemoryUsage() const |
| public IloInt | getNumberOfActivityChainPrototypes() const |
| public IloInt | getNumberOfBuckets() const |
| public IloInt | getNumberOfBucketSequences() const |
| public IloInt | getNumberOfCalendars() const |
| public IloInt | getNumberOfDemands() const |
| public IloInt | getNumberOfInventoryMaxCostFunctions() const |
| public IloInt | getNumberOfInventoryMinCostFunctions() const |
| public IloInt | getNumberOfMaterialFamilies(IloMSIdentifier materialFamilyType) const |
| public IloInt | getNumberOfMaterialFamilies() const |
| public IloInt | getNumberOfMaterialFamilyTypes() const |
| public IloInt | getNumberOfMaterialFlowArcs() const |
| public IloInt | getNumberOfMaterials() const |
| public IloInt | getNumberOfOptimizationCriteria() const |
| public IloInt | getNumberOfOptimizationProfiles() const |
| public IloInt | getNumberOfProcurements() const |
| public IloInt | getNumberOfProductionOrders() const |
| public IloInt | getNumberOfQualities() const |
| public IloInt | getNumberOfRecipeFamilies(IloMSIdentifier recipeFamilyType) const |
| public IloInt | getNumberOfRecipeFamilies() const |
| public IloInt | getNumberOfRecipeFamilyTypes() const |
| public IloInt | getNumberOfRecipes() const |
| public IloInt | getNumberOfResourceCapacityCostFunctions() const |
| public IloInt | getNumberOfResourceFamilies(IloMSIdentifier type) const |
| public IloInt | getNumberOfResourceFamilies() const |
| public IloInt | getNumberOfResourceFamilyTypes() const |
| public IloInt | getNumberOfResources() const |
| public IloInt | getNumberOfScopes() const |
| public IloInt | getNumberOfSetupMatrices() const |
| public IloInt | getNumberOfStandardKPIs() const |
| public IloInt | getNumberOfUnits() const |
| public IloMSOptimizationCriterion | getOptimizationCriterion(int index) const |

| | |
|---|---|
| public IloMSOptimizationCriterion | getOptimizationCriterionByIdentifier(IloMSIdenti identifier) const |
| public IloMSOptimizationProfile | getOptimizationProfile(IloInt index) const |
| public IloMSOptimizationProfile | getOptimizationProfileByIdentifier(IloMSIdentifi identifier) const |
| public IloMSBucketSequence | getOptimizedBucketSequence() |
| public IloInt | getPlanningHorizon() const |
| public IloInt | getPlanningOrigin() const |
| public IloNum | getPlanningWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getPlanningWeight(IloMSIdentifier criterionId) const |
| public IloMSProcurement | getProcurement(IloInt index) const |
| public IloMSProductionOrder | getProductionOrder(IloInt index) const |
| public IloMSProductionOrder | getProductionOrderByIdentifier(IloMSIdentifier identifier) const |
| public IloInt | getProductivityGrain() const |
| public IloMSQuality | getQuality(IloInt index) const |
| public IloMSQuality | getQualityByIdentifier(IloMSIdentifier identifier) const |
| public IloMSRecipe | getRecipe(IloInt index) const |
| public IloMSRecipe | getRecipeByIdentifier(IloMSIdentifier identifier) const |
| public IloMSRecipeFamily | getRecipeFamily(IloMSIdentifier recipeFamilyType, IloInt index) const |
| public IloMSRecipeFamily | getRecipeFamily(IloInt index) const |
| public IloMSRecipeFamily | getRecipeFamilyByIdentifier(IloMSIdentifier identifier) const |
| public IloMSIdentifier | getRecipeFamilyType(IloInt index) const |
| public IloMSRepairAlgorithmI * | getRepairAlgorithm() |
| public IloMSReplicateAlgorithmI * | getReplicateAlgorithm() |
| public IloMSResource | getResource(IloInt index) const |
| public IloMSResource | getResourceByIdentifier(IloMSIdentifier identifier) const |
| public IloMSResourceFamily | getResourceFamily(IloInt index) const |
| public IloMSResourceFamily | getResourceFamily(IloMSIdentifier resourceFamilyType, IloInt index) const |
| public IloMSResourceFamily | getResourceFamilyByIdentifier(IloMSIdentifier identifier) const |
| public IloMSIdentifier | getResourceFamilyType(IloInt index) const |
| public IloInt | getSchedulingHorizon() const |
| public IloNum | getSchedulingWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getSchedulingWeight(IloMSIdentifier criterionId) const |

| | |
|---|---|
| public IloMSScope | getScope(IloInt index) const |
| public IloMSScope | getScopeByIdentifier(IloMSIdentifier identifier) const |
| public IloMSSetupMatrix | getSetupMatrix(IloInt index) const |
| public IloMSSetupMatrix | getSetupMatrixByIdentifier(IloMSIdentifier identifier) const |
| public IloInt | getSlackOnPlannedEndTime() const |
| public IloInt | getSlackOnPlannedStartTime() const |
| public IloMSStandardKPI | getStandardKPI(IloInt index) const |
| public IloInt | getStartMin() const |
| public IloNum | getTime() const |
| public IloInt | getTimeCheckingTolerance() const |
| public IloInt | getTimeUnit() const |
| public const char * | getTimeZone() const |
| public IloMSUnit | getUnit(IloInt index) const |
| public IloMSUnit | getUnitByIdentifier(IloMSIdentifier identifier) const |
| public const char * | getVersion() const |
| public IloNum | getWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getWeight(IloMSIdentifier criterionId) const |
| public IloBool | hasPlanningData() const |
| public IloBool | hasSchedulingData() const |
| public IloBool | isClosed() |
| public IloBool | isTimeZoneDefined() const |
| public IloMSActivityChain | newActivityChainPrototype(IloMSRecipe recipe) |
| public IloMSActivityCompatibilityConstraint | newActivityCompatibilityConstraint(IloMSAbstract act1, IloMSAbstractActivity act2, IloMSActivityCompatibilityType type) |
| public IloMSActivity | newActivityPrototype(IloMSRecipe recipe) |
| public IloMSBatchingSolution | newBatchingSolution() |
| public IloMSBucket | newBucket(IloInt startTime, IloInt endTime, IloMSBucketSequence bucketSequence) |
| public IloMSBucket | newBucket(IloInt startTime, IloInt endTime) |
| public IloMSBucketSequence | newBucketSequence() |
| public IloMSBucketTemplate | newBucketTemplate(IloMSBucketTemplateSequence bucketTemplateSequence, IloInt bucketRank, IloMSBucketType type, IloMSBucketPeriodUnit unit, IloInt numberOfPeriods) |
| public IloMSBucketTemplateSequence | newBucketTemplateSequence(IloMSBucketSequence bucketSequence) |
| public IloMSCalendar | newCalendar() |

| | |
|---|---|
| public IloMSCalendarInterval | newCalendarInterval(IloMSCalendar calendar, IloInt startTime, IloInt endTime) |
| public IloMSDefaultCheckForStop | newDefaultCheckForStop() |
| public IloMSDemand | newDemand(IloMSMaterial material, IloNum quantity) |
| public IloMSDueDate | newDueDate(IloMSAbstractActivity activity, IloInt time) |
| public IloMSDueDate | newDueDate(IloMSDemand demand, IloInt time) |
| public IloMSInventoryMaxCostFunction | newInventoryMaxCostFunction() |
| public IloMSInventoryMinCostFunction | newInventoryMinCostFunction() |
| public IloMSMaterial | newMaterial() |
| public IloMSMaterialFamily | newMaterialFamily() |
| public IloMSMaterialFamilyCardinalityConstraint | newMaterialFamilyCardinalityConstraint(IloMSMate family, IloMSBucketSequence sequence, IloInt cardMax) |
| public IloMSMaterialProduction | newMaterialProduction(IloMSMode mode, IloMSMaterial material, IloNum variableQuantity, IloNum fixedQuantity) |
| public IloMSMaterialProduction | newMaterialProduction(IloMSAbstractActivity absAct, IloMSMaterial material, IloNum variableQuantity, IloNum fixedQuantity) |
| public IloMSMaterialProduction | newMaterialProduction(IloMSMode mode, IloMSMaterial material, IloNum quantity) |
| public IloMSMaterialProduction | newMaterialProduction(IloMSAbstractActivity absAct, IloMSMaterial material, IloNum quantity) |
| public IloMSMode | newMode(IloMSAbstractActivity activity) |
| public static IloMSModel | newModel() |
| public IloMSOptimizationProfile | newOptimizationProfile(IloMSOptimizationProfile profile) |
| public IloMSOptimizationProfile | newOptimizationProfile() |
| public IloMSPlanningSolution | newPlanningSolution() |
| public IloMSPrecedence | newPrecedence(IloMSAbstractActivity pred, IloMSAbstractActivity succ, IloMSPrecedenceType type, IloInt delayMin, IloInt delayMax) |
| public IloMSProcurement | newProcurement(IloMSMaterial material, IloNum quantity) |
| public IloMSProcurementToDemandArc | newProcurementToDemandArc(IloMSMaterial material, IloNum quantity, IloMSProcurement pred, IloMSDemand succ) |
| public IloMSProcurementToProdArc | newProcurementToProdArc(IloMSMaterial material, IloNum quantity, IloMSProcurement pred, IloMSProductionOrder succ) |
| public IloMSProcurementToStorageArc | newProcurementToStorageArc(IloMSMaterial material, IloNum quantity, IloMSProcurement pred) |

| | |
|---|---|
| public IloMSProdToDemandArc | newProdToDemandArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder pred, IloMSDemand succ) |
| public IloMSProdToStorageArc | newProdToStorageArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder pred) |
| public IloMSQuality | newQuality() |
| public IloMSRecipe | newRecipe() |
| public IloMSRecipeFamily | newRecipeFamily() |
| public IloMSRecipeFamilyFilter | newRecipeFamilyFilter(IloMSScope scope) |
| public IloMSResource | newResource(IloInt capacity) |
| public IloMSResourceCapacityCostFunction | newResourceCapacityCostFunction() |
| public IloMSResourceConstraint | newResourceConstraint(IloMSMode mode, IloMSResource resource, IloInt capacity, IloBool isPrimary) |
| public IloMSResourceFamily | newResourceFamily() |
| public IloMSScope | newScope() |
| public IloMSSetupMatrix | newSetupMatrix() |
| public IloMSActivity | newStorageActivityPrototype(IloMSActivity producingActivity, IloMSMaterial producedMaterial) |
| public IloMSStorageToDemandArc | newStorageToDemandArc(IloMSMaterial material, IloNum quantity, IloMSDemand succ) |
| public IloMSStorageToProdArc | newStorageToProdArc(IloMSMaterial material, IloNum quantity, IloMSProductionOrder succ) |
| public IloMSStorageUnit | newStorageUnit() |
| public IloMSUnit | newUnit() |
| public IloMSIdentifier | newUUID() |
| public IloMSRecipe | newWasteRecipe(IloMSMaterial material, IloNum wasteCost) |
| public void | putSetting(const char * valueName, const char * value) |
| public void | readSettings(const char * filename) |
| public void | registerLicense(char * plantLicense, int plantSignature) |
| public void | removeAllProperties() |
| public void | removeAutomaticRecipes() |
| public void | removeNotFirmedData() |
| public void | removeProperty(const char * name) |
| public IloBool | repeg() |
| public void | reserveCurrentBatchingSolution() |
| public void | rollbackCurrentBatchingSolution() |
| public void | setBatchingHorizon(IloInt horizon) |
| public void | |

| | |
|---|---|
| | setBatchingWeight(IloMSOptimizationCriterion criterion, IloNum value) |
| public void | setBatchingWeight(IloMSIdentifier criterionId, IloNum value) |
| public void | setCheckForStop(IloMSCheckForStop hook) |
| public void | setCurrentBatchingSolution(IloMSBatchingSolution solution) |
| public void | setCurrentMaterialFamilyType(IloMSIdentifier familyType) |
| public void | setCurrentOptimizationProfile(IloMSOptimizationP profile) |
| public void | setCurrentPlanningSolution(IloMSPlanningSolution solution) |
| public void | setCurrentRecipeFamilyType(IloMSIdentifier familyType) |
| public void | setCurrentResourceFamilyType(IloMSIdentifier familyType) |
| public void | setDateOrigin(IloMSDate date) |
| public void | setDisplayedBucketSequence(IloMSBucketSequence sequence) |
| public void | setEndMax(IloInt latestEndTime) |
| public void | setIntDateOrigin(IloInt inSecondsSince_01_01_2001) |
| public void | setIntProperty(const char * name, IloInt value) |
| public void | setMakespanOrigin(IloInt makespanOrigin) |
| public void | setNumProperty(const char * name, IloNum value) |
| public void | setObject(IloAny object) |
| public void | setOptimizedBucketSequence(IloMSBucketSequence sequence) |
| public void | setPlanningHorizon(IloInt horizon) |
| public void | setPlanningWeight(IloMSOptimizationCriterion criterion, IloNum value) |
| public void | setPlanningWeight(IloMSIdentifier criterionId, IloNum value) |
| public void | setProductivityGrain(IloInt grain) |
| public void | setSchedulingHorizon(IloInt horizon) |
| public void | setSchedulingWeight(IloMSOptimizationCriterion criterion, IloNum value) |
| public void | setSchedulingWeight(IloMSIdentifier criterionId, IloNum value) |
| public void | setSlackOnPlannedEndTime(IloInt value) |
| public void | setSlackOnPlannedStartTime(IloInt value) |
| public void | setStartMin(IloInt earliestStartTime) |
| public void | setStringProperty(const char * name, const char * value) |

| | |
|---|---|
| public void | setTimeCheckingTolerance(IloInt toleranceInTimeUnits) |
| public void | setTimeUnit(IloInt inSeconds) |
| public void | setTimeZone(const char * identifier) |
| public void | setTraceFile(const char * filename=0, ios_base::openmode mode=ios_base::out) |
| public void | setTraceLevel(IloInt level) |
| public void | setWeight(IloMSOptimizationCriterion criterion, IloNum value) |
| public void | setWeight(IloMSIdentifier criterionId, IloNum value) |
| public IloBool | solve() const |
| public void | write(const char * filename, IloBool writeSettings) |
| public void | writeSettings(const char * filename) |
| public void | writeSolution(const char * filename, IloMSSchedulingSolution solution) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Constructors

public **IloMSModel**()

This constructor creates a new `IloMSModel` object.

## Methods

public void **close**()

This method is used to close the model.

After the creation of all objects that are to be contained in the model (including the precedence and mode compatibility constraints, activity chains and due dates), the model can be closed. If the model is not closed then there is no access to any implicitly created setup activities, and `IloMSBatchingSolution::newSchedulingSolution` will raise an exception. If you call IloMSModel::solve without closing the model, it will be closed automatically.

public void **commitCurrentBatchingSolution**()

internal

This method deletes the earlier batching solution saved when calling the `reserveCurrentBatchingSolution()` method. A typical use of this method is to save a successful solution in a custom batching algorithm.

```
public IloInt convertDateToTime(const IloMSDate & date) const
```

This method converts a `date` since the date origin of the problem to a number of time units.

```
public IloMSDate convertTimeToDate(IloInt time) const
```

This method converts the number of `time` units since the date origin of the problem into a date.

```
public IloMSModel copy()
```

This method returns a copy of the invoking model.

```
public void copyBatchingFrom(IloMSModel origin)
```

This methods copies the production orders and arcs existing on the current batching solution of the model passed as parameter (`origin`) to the current batching solution of the invoking model.

Note that the method `generateActivities` and `copySchedulingSolutioFrom` must be invoked to get a complete solution.

```
public void copyPlanningSolutionFrom(IloMSModel origin)
```

This methods copies the planned productions and planned deliveries from the current planning solution of the model passed as parameter (`origin`) to the current planning solution of the invoking model.

Previous planned productions and planned deliveries are erased.

```
public void copySchedulingSolutionFrom(IloMSModel origin)
```

This methods copies the scheduling information from the current scheduling solution of the model passed as parameter (`origin`) to the current scheduling solution of the invoking model.

This is performed for all activities in the target model that have a counterpart in the original model.

```
public void end()
```

This method cleans up the environment owned by the invoking `IloMSModel` object.

```
public void generateActivities()
```

This method generates for all production orders the real activities from the prototype activities contained in the recipes, and associated constraints. Note that the `close` and `solve` methods automatically call `generateActivities`.

```
public IloMSActivityChain getActivityChainPrototype(IloInt index) const
```

This method returns the activity chain prototype with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSActivityChain getActivityChainPrototypeByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the activity chain with the given `identifier`. An exception is thrown if there is no activity chain with the given `identifier`.

```
public IloMSAbstractActivity getActivityPrototypeByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the activity prototype with the given `identifier`. An exception is thrown if there is no activity prototype with the given `identifier`.

```
public IloInt getBatchingHorizon() const
```

This deprecated method returns the end of horizon to consider when batching. Use the method `getSchedulingHorizon`.

```
public IloNum getBatchingWeight(IloMSOptimizationCriterion criterion) const
```

This method returns the weight of the given criterion.

```
public IloNum getBatchingWeight(IloMSIdentifier criterionId) const
```

This method returns the weight of the given criterion.

```
public IloMSBucket getBucket(IloInt index) const
```

This method returns the time bucket with the given `index`. The returned time bucket is one owned by the bucket sequence specified as the optimized bucket sequence. An exception is thrown if the given `index` is out of bounds.

```
public IloMSBucket getBucketByIdentifier(IloMSIdentifier identifier) const
```

This method returns the time bucket with the given `identifier`. An exception is thrown if there is no time bucket with the given `identifier`.

```
public IloMSBucketSequence getBucketSequence(IloInt index) const
```

This method returns the bucket sequence with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSCalendar getCalendar(IloInt index) const
```

This method returns the calendar with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSCalendar getCalendarByIdentifier(IloMSIdentifier identifier) const
```

This method returns the calendar with the given `identifier`. An exception is thrown if there is no calendar with the given `identifier`.

```
public IloMSChecker getChecker()
```

This method returns the checker of the invoking model.

```
public IloMSBatchingSolution getCurrentBatchingSolution() const
```

This method returns the currently active batching solution.

```
public IloMSOptimizationProfile getCurrentOptimizationProfile() const
```

This function returns the current optimization profile containing all overridden parameters for engines.

```
public IloMSPlanningSolution getCurrentPlanningSolution() const
```

This method returns the current planning solution.

```
public IloMSSchedulingSolution getCurrentSchedulingSolution() const
```

This method returns the currently active scheduling solution. Note that after a successful solve, the previous scheduling solution is destroyed and thus any prior reference is invalid.

```
public IloMSDate getDateOrigin() const
```

This method returns the absolute date origin. The given date is understood to be in UTC.

```
public static IloInt getDefaultTimeCheckingTolerance()
```

This method returns the default tolerance used for filtering nonfatal error messages with regard to time and capacity violations in the checkers.

```
public IloMSDemand getDemand(IloInt index) const
```

This method returns the demand with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSDemand getDemandByIdentifier(IloMSIdentifier identifier) const
```

This method returns the demand with the given `identifier`. An exception is thrown if there is no demand with the given `identifier`.

```
public IloMSBucketSequence getDisplayedBucketSequence() const
```

This function returns the bucket sequence used by the GUI.

```
public IloNum getElapsedTime() const
```

This method returns the time, in seconds, elapsed since the invocation of the method `solve()`.

```
public IloInt getEndMax() const
```

This method returns the common latest end time associated with all the activities of the model.

```
public IloNum getEpsilon() const
```

This method returns the default tolerance used for floating point arithmetic.

```
public IloInt getIndex(IloMSResource res) const
```

This method returns the index of the resource `res` in the model. Many Plant PowerOps objects can be accessed via an index ranging from `0` to the number of objects in the model minus one.

```
public IloInt getIndex(IloMSActivity act) const
```

This method returns the index of the activity `act` in the model. Many Plant PowerOps objects can be accessed via an index ranging from `0` to the number of objects in the model minus one. The index is available for all actual activities of the model, not for the activity prototypes of the recipe. To find out the index of an activity prototype in a recipe use the `getPrototypeIndex(act)` method on the `IloMSRecipe`.

```
public IloInt getIntDateOrigin() const
```

This method returns the date origin, expressed in seconds since January 1st, 2001 UTC.

```
public IloMSInventoryMaxCostFunction getInventoryMaxCostFunction(IloInt index)
const
```

This method returns the maximal inventory cost function with the given index. An exception is thrown if the given `index` is out of bounds.

```
public IloMSInventoryMaxCostFunction
getInventoryMaxCostFunctionByIdentifier(IloMSIdentifier identifier) const
```

This method returns the maximal inventory cost function with the given `identifier`. An exception is thrown if there is no maximal inventory cost function with the given `identifier`.

```
public IloMSInventoryMinCostFunction getInventoryMinCostFunction(IloInt index)
const
```

This method returns the minimal inventory cost function with the given index. An exception is thrown if the given `index` is out of bounds.

```
public IloMSInventoryMinCostFunction
getInventoryMinCostFunctionByIdentifier(IloMSIdentifier identifier) const
```

This method returns the minimal inventory cost function with the given `identifier`. An exception is thrown if there is no minimal inventory cost function with the given `identifier`.

```
public IloInt getMakespanOrigin() const
```

This method returns the makespan origin, that is, the time that serves as a basis to calculate the makespan. The makespan is the difference between the end time of the last activity and the returned origin.

```
public IloMSMaterial getMaterial(IloInt index) const
```

This method returns the material with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSMaterial getMaterialByIdentifier(IloMSIdentifier identifier) const
```

This method returns the material with the given `identifier`. An exception is thrown if there is no material with the given `identifier`.

```
public IloMSMaterialFamily getMaterialFamily(IloMSIdentifier materialFamilyType,
IloInt index) const
```

This function returns the material family with the `index` of the corresponding material family type.

```
public IloMSMaterialFamily getMaterialFamily(IloInt index) const
```

This method returns the material family with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSMaterialFamily getMaterialFamilyByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the material family with the given `identifier`. An exception is thrown if there is no material family with the given `identifier`.

```
public IloMSIdentifier getMaterialFamilyType(IloInt index) const
```

This function returns the material family type identifier with the corresponding `index`.

```
public IloMSAbstractMaterialFlowArc getMaterialFlowArc(IloInt index) const
```

This method returns the material flow arc with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloInt getMemoryUsage() const
```

This method returns the memory usage of the model.

```
public IloInt getNumberOfActivityChainPrototypes() const
```

This method returns the number of activity chains in the model.

```
public IloInt getNumberOfBuckets() const
```

This function returns the number of existing time buckets used by the optimizer.

```
public IloInt getNumberOfBucketSequences() const
```

This function returns the number bucket sequence; at least one should be defined

```
public IloInt getNumberOfCalendars() const
```

This method returns the number of calendars in the model.

```
public IloInt getNumberOfDemands() const
```

This function returns the number of existing demands.

```
public IloInt getNumberOfInventoryMaxCostFunctions() const
```

This method returns the number of inventory max cost functions in the model.

```
public IloInt getNumberOfInventoryMinCostFunctions() const
```

This method returns the number of inventory min cost functions in the model.

```
public IloInt getNumberOfMaterialFamilies(IloMSIdentifier materialFamilyType) const
```

This function returns the number of material families of the corresponding material family type.

```
public IloInt getNumberOfMaterialFamilies() const
```

This function returns the number of existing material families.

```
public IloInt getNumberOfMaterialFamilyTypes() const
```

This function returns the total number of material family types.

```
public IloInt getNumberOfMaterialFlowArcs() const
```

This function returns the number of existing material flow arcs.

```
public IloInt getNumberOfMaterials() const
```

This function returns the number of existing materials.

```
public IloInt getNumberOfOptimizationCriteria() const
```

This method returns the number of optimization criteria.

```
public IloInt getNumberOfOptimizationProfiles() const
```

This function returns the number of existing optimization profiles.

```
public IloInt getNumberOfProcurements() const
```

This function returns the number of existing procurements.

```
public IloInt getNumberOfProductionOrders() const
```

This function returns the number of existing production orders.

```
public IloInt getNumberOfQualities() const
```

This function returns the number of existing material qualities.

```
public IloInt getNumberOfRecipeFamilies(IloMSIdentifier recipeFamilyType) const
```

This function returns the number of recipe families of the corresponding recipe family type.

```
public IloInt getNumberOfRecipeFamilies() const
```

This function returns the number of existing recipe families.

```
public IloInt getNumberOfRecipeFamilyTypes() const
```

This function returns the total number of recipe family types.

```
public IloInt getNumberOfRecipes() const
```

This function returns the number of existing recipes.

```
public IloInt getNumberOfResourceCapacityCostFunctions() const
```

This method returns the number of resource capacity cost functions in the model.

```
public IloInt getNumberOfResourceFamilies(IloMSIdentifier type) const
```

This function returns the number of existing resource families of the corresponding type.

```
public IloInt getNumberOfResourceFamilies() const
```

This function returns the number of existing resource families.

```
public IloInt getNumberOfResourceFamilyTypes() const
```

This function returns the total number of resource family types.

```
public IloInt getNumberOfResources() const
```

This method returns the number of resources in the model.

```
public IloInt getNumberOfScopes() const
```

This method returns the number of scopes.

```
public IloInt getNumberOfSetupMatrices() const
```

This method returns the number of setup matrices in the model.

```
public IloInt getNumberOfStandardKPIs() const
```

This method returns the number of standard Key Performance Indicators defined in PPO.

```
public IloInt getNumberOfUnits() const
```

This function returns the number of existing units of measure.

```
public IloMSOptimizationCriterion getOptimizationCriterion(int index) const
```

This method returns the optimization criterion at `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSOptimizationCriterion
getOptimizationCriterionByIdentifier(IloMSIdentifier identifier) const
```

This method returns the optimization criterion with the given `identifier`. An exception is thrown if there is no optimization criterion with the given `identifier`.

```
public IloMSOptimizationProfile getOptimizationProfile(IloInt index) const
```

This method returns the optimization profile with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSOptimizationProfile getOptimizationProfileByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the optimization profile with the given `identifier`. An exception is thrown if there is no optimization profile with the given `identifier`.

```
public IloMSBucketSequence getOptimizedBucketSequence()
```

This function returns the bucket sequence used by the optimizer.

The default bucket sequence is `null`.

```
public IloInt getPlanningHorizon() const
```

This method returns the end of horizon to consider when planning. The planning engine will take decisions over the optimized bucket sequence until the latest bucket that ends exactly at or before this limit. See `setPlanningHorizon` for more information.

```
public IloInt getPlanningOrigin() const
```

This method returns the origin time of the inventory. This typically equals the start time of the first time bucket. All procurements received before this date are considered to be in the initial stock.

```
public IloNum getPlanningWeight(IloMSOptimizationCriterion criterion) const
```

This method returns the weight of the given criterion.

```
public IloNum getPlanningWeight(IloMSIdentifier criterionId) const
```

This method returns the weight of the given criterion.

```
public IloMSProcurement getProcurement(IloInt index) const
```

This method returns the procurement with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSProductionOrder getProductionOrder(IloInt index) const
```

This method returns the production order with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSProductionOrder getProductionOrderByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the production order with the given `identifier`. An exception is thrown if there is no production order with the given `identifier`.

```
public IloInt getProductivityGrain() const
```

This method returns the precision of the productivity computation.

```
public IloMSQuality getQuality(IloInt index) const
```

This method returns the quality with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSQuality getQualityByIdentifier(IloMSIdentifier identifier) const
```

This method returns the quality with the given `identifier`. An exception is thrown if there is no unit with the given `identifier`.

```
public IloMSRecipe getRecipe(IloInt index) const
```

This method returns the recipe with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSRecipe getRecipeByIdentifier(IloMSIdentifier identifier) const
```

This method returns the recipe with the given `identifier`. An exception is thrown if there is no recipe with the given `identifier`.

```
public IloMSRecipeFamily getRecipeFamily(IloMSIdentifier recipeFamilyType, IloInt
index) const
```

This function returns the recipe family with the `index` of the corresponding recipe family type.

```
public IloMSRecipeFamily getRecipeFamily(IloInt index) const
```

This method returns the recipe family with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSRecipeFamily getRecipeFamilyByIdentifier(IloMSIdentifier identifier)
const
```

This method returns the recipe family with the given `identifier`. An exception is thrown if there is no recipe family with the given `identifier`.

```
public IloMSIdentifier getRecipeFamilyType(IloInt index) const
```

This function returns the recipe family type identifier with the corresponding `index`.

```
public IloMSRepairAlgorithmI * getRepairAlgorithm()
```

This method returns the repair algorithm of the invoking model.

```
public IloMSReplicateAlgorithmI * getReplicateAlgorithm()
```

This method returns the replicate algorithm of the invoking model.

```
public IloMSResource getResource(IloInt index) const
```

This method returns the resource with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSResource getResourceByIdentifier(IloMSIdentifier identifier) const
```

This method returns the resource with the given `identifier`. An exception is thrown if there is no resource with the given `identifier`.

```
public IloMSResourceFamily getResourceFamily(IloInt index) const
```

This function returns the resource family with the `index`.

```
public IloMSResourceFamily getResourceFamily(IloMSIdentifier resourceFamilyType,
IloInt index) const
```

This function returns the resource family with the `index` of the corresponding resource family type.

```
public IloMSResourceFamily getResourceFamilyByIdentifier(IloMSIdentifier
identifier) const
```

This method returns the resource family with the given `identifier`. An exception is thrown if there is no resource family with the given `identifier`.

```
public IloMSIdentifier getResourceFamilyType(IloInt index) const
```

This function returns the resource family type identifier with the corresponding `index`.

```
public IloInt getSchedulingHorizon() const
```

This method returns the end time of the scheduling horizon of the current optimization profile. The scheduling horizon affects the batching engine, scheduling engine, and GUI behavior. See the method `setSchedulingHorizon` for more information.

```
public IloNum getSchedulingWeight(IloMSOptimizationCriterion criterion) const
```

This method returns the weight of the given criterion.

```
public IloNum getSchedulingWeight(IloMSIdentifier criterionId) const
```

This method returns the weight of the given criterion.

```
public IloMSScope getScope(IloInt index) const
```

This method returns the scope with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSScope getScopeByIdentifier(IloMSIdentifier identifier) const
```

This method returns the scope with the given `identifier`. An exception is thrown if there is no scope with the given `identifier`.

```
public IloMSSetupMatrix getSetupMatrix(IloInt index) const
```

This method returns the setup matrix with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSSetupMatrix getSetupMatrixByIdentifier(IloMSIdentifier identifier)
const
```

This method returns the setup matrix with the given `identifier`. An exception is thrown if there is no setup matrix with the given `identifier`.

```
public IloInt getSlackOnPlannedEndTime() const
```

This method returns the slack applied by the lot streaming engine to the end time of production orders generated from the planning solution.

```
public IloInt getSlackOnPlannedStartTime() const
```

This method returns the slack applied by the lot streaming engine to the start time of production orders generated from the planning solution.

```
public IloMSStandardKPI getStandardKPI(IloInt index) const
```

This method returns the standard Key Performance Indicator with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloInt getStartMin() const
```

This method returns the common earliest start time associated with all the activities of the model.

```
public IloNum getTime() const
```

This method returns the time, in seconds, since the **creation** of the invoking `IloMSModel` object.

```
public IloInt getTimeCheckingTolerance() const
```

This method returns the tolerance used for filtering nonfatal error messages with regard to time and capacity violations in the checkers.

```
public IloInt getTimeUnit() const
```

This method returns the time unit, expressed in seconds.

```
public const char * getTimeZone() const
```

This method returns the identifier of the time zone.

```
public IloMSUnit getUnit(IloInt index) const
```

This method returns the unit of measure with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSUnit getUnitByIdentifier(IloMSIdentifier identifier) const
```

This method returns the unit of measure with the given `identifier`. An exception is thrown if there is no unit with the given `identifier`.

```
public const char * getVersion() const
```

This method returns the version number as the string "Major.Minor.Technical Tag", with the copyright and build information of the PPO version you are using.

```
public IloNum getWeight(IloMSOptimizationCriterion criterion) const
```

This method returns the weight of the given criterion.

```
public IloNum getWeight(IloMSIdentifier criterionId) const
```

This method returns the weight of the given criterion.

```
public IloBool hasPlanningData() const
```

This predicate is true if the model currently contains planning data, typically time buckets.

```
public IloBool hasSchedulingData() const
```

This predicate is true if the model currently contains scheduling data, typically, production orders or activities.

```
public IloBool isClosed()
```

This method returns true if the model has been closed, and false otherwise. See IloMSModel::close.

```
public IloBool isTimeZoneDefined() const
```

This predicate is true if the model has a defined time zone different from the default one.

```
public IloMSActivityChain newActivityChainPrototype(IloMSRecipe recipe)
```

This method creates and returns a new activity chain prototype to be used for constraining several activity prototypes. For related information see the PPO_ACTIVITY_CHAIN table in the *Data Schema*.

```
public IloMSActivityCompatibilityConstraint
newActivityCompatibilityConstraint(IloMSAbstractActivity act1,
IloMSAbstractActivity act2, IloMSActivityCompatibilityType type)
```

This method creates and returns a new activity compatibility constraint of the given `type`.

The possible types are `IloMSSameLineId` which constrains the two modes (of the two given activities) to have the same line identifier; `IloMSSamePrimaryResource` which constrains the two modes to have the same primary resource; `IloMSSamePrimaryResourceAndCapacity` constrains the two modes to have the same primary resource and capacity requirement; `IloMSSamePerformedStatus` means that the two activities have to be in the same performed status (either both performed or both unperformed), and `IloMSDifferentPerformedStatus` means the opposite.

The other types are: `IloMSPerformedImpliesPerformed`, `IloMSUnperformedImpliesUnperformed`, `IloMSPerformedImpliesUnperformed`, `IloMSUnperformedImpliesPerformed` which state that the status of `act1` logically implies the status of `act2`.

**See Also:** IloMSActivityCompatibilityType

```
public IloMSActivity newActivityPrototype(IloMSRecipe recipe)
```

This function creates and returns a new activity prototype for the `recipe`. For related information see the PPO_ACTIVITY_PROTO table in the *Data Schema*.

```
public IloMSBatchingSolution newBatchingSolution()
```

This method returns a new instance of a batching solution.

```
public IloMSBucket newBucket(IloInt startTime, IloInt endTime, IloMSBucketSequence
bucketSequence)
```

This method creates and returns a new time bucket beginning at `startTime`, ending at `endTime` and for the sequence `bucketSequence`. The sequence `bucketSequence` passed as parameter cannot be null. For related information see the PPO_BUCKET table in the *Data Schema*.

```
public IloMSBucket newBucket(IloInt startTime, IloInt endTime)
```

This method creates and returns a new time bucket in the optimized bucket sequence beginning at `startTime` and ending at `endTime`. For related information see the PPO_BUCKET table in the *Data Schema*.

```
public IloMSBucketSequence newBucketSequence()
```

This method creates and returns a new bucket sequence. For related information see the PPO_BUCKET_SEQUENCE table in the *Data Schema*.

```
public IloMSBucketTemplate newBucketTemplate(IloMSBucketTemplateSequence
bucketTemplateSequence, IloInt bucketRank, IloMSBucketType type,
IloMSBucketPeriodUnit unit, IloInt numberOfPeriods)
```

This method creates and returns a new bucket template. For related information see the PPO_BUCKET_TEMPLATE table in the *Data Schema*.

```
public IloMSBucketTemplateSequence newBucketTemplateSequence(IloMSBucketSequence
bucketSequence)
```

This method creates and returns a new bucket template sequence. For related information see the PPO_BUCKET_SEQUENCE table in the *Data Schema*.

```
public IloMSCalendar newCalendar()
```

This method creates and returns a new calendar. For related information see the PPO_CALENDAR table in the *Data Schema*.

```
public IloMSCalendarInterval newCalendarInterval(IloMSCalendar calendar, IloInt
startTime, IloInt endTime)
```

This method creates and returns a new calendar interval on the calendar `calendar`. For related information see the PPO_CALENDAR_INTERVAL table in the *Data Schema*.

```
public IloMSDefaultCheckForStop newDefaultCheckForStop()
```

This method creates and returns a new default check for the stop object.

```
public IloMSDemand newDemand(IloMSMaterial material, IloNum quantity)
```

This function creates and returns a new demand for the specified `quantity` of `material`. For related information see the PPO_DEMAND table in the *Data Schema*.

```
public IloMSDueDate newDueDate(IloMSAbstractActivity activity, IloInt time)
```

This method creates and returns a new due date for the given `activity`. For related information see the PPO_DUE_DATE table in the *Data Schema*.

```
public IloMSDueDate newDueDate(IloMSDemand demand, IloInt time)
```

This method creates and returns a new due date for the given `demand`. For related information see the PPO_DUE_DATE table in the *Data Schema*.

```
public IloMSInventoryMaxCostFunction newInventoryMaxCostFunction()
```

This method defines a maximal inventory cost function. The class `IloMSInventoryMaxCostFunction` is used to evaluate the cost of violating the maximal inventory over time. For related information see the PPO_INVENTORY_MAX_COST_FCT table in the *Data Schema*.

**See Also:** IloMSInventoryMaxCostFunction

```
public IloMSInventoryMinCostFunction newInventoryMinCostFunction()
```

This method defines a minimal inventory cost function. The class `IloMSInventoryMinCostFunction` is used to evaluate the cost of violating the safety stock over time. For related information see the PPO_INVENTORY_MIN_COST_FCT table in the *Data Schema*.

**See Also:** IloMSInventoryMinCostFunction

```
public IloMSMaterial newMaterial()
```

This function creates and returns a new material. For related information see the PPO_MATERIAL table in the *Data Schema*.

```
public IloMSMaterialFamily newMaterialFamily()
```

This function creates and returns a new material family. For related information see the PPO_MATERIAL_FAMILY table in the *Data Schema*.

```
public IloMSMaterialFamilyCardinalityConstraint
newMaterialFamilyCardinalityConstraint(IloMSMaterialFamily family,
IloMSBucketSequence sequence, IloInt cardMax)
```

This function creates a new material family cardinality constraint on `family`. This constraint is active in planning, and expresses the fact that at most `cardMax` materials of the family can be produced in each bucket of `sequence`. Note that while `sequence` may differ from the sequence used to create planning buckets, the two must have the same milestones. For example, if the planning engine uses daily buckets, the cardinality sequence may use two-day buckets, but not half-day buckets.

```
public IloMSMaterialProduction newMaterialProduction(IloMSMode mode, IloMSMaterial
material, IloNum variableQuantity, IloNum fixedQuantity)
```

This method defines a production (or consumption if `quantity` is negative) of the `material` by the `mode`. All other modes of the associated activity will not be affected when calling this method. If you want to define a common behavior for all modes of an activity, it is recommended to pass the activity instead of the mode. For related information see the PPO_MATERIAL_PRODUCTION_PROTO table in the *Data Schema*.

```
public IloMSMaterialProduction newMaterialProduction(IloMSAbstractActivity absAct,
IloMSMaterial material, IloNum variableQuantity, IloNum fixedQuantity)
```

This method defines a production or consumption (if `quantity` is negative) of the `material` by the `activity`. All modes of this activity will produce/consume the `material` in the same way. If `absAct` is an activity prototype, the material production will have a reference to the corresponding recipe. For related information see the PPO_MATERIAL_PRODUCTION_PROTO table in the *Data Schema*.

```
public IloMSMaterialProduction newMaterialProduction(IloMSMode mode, IloMSMaterial
material, IloNum quantity)
```

This method defines a production (or consumption if `quantity` is negative) of the `material` by the `mode`. All other modes of the associated activity will not be affected when calling this method. If you want to define a common behavior for all modes of an activity, it is recommended to pass the activity instead of the mode. For related information see the PPO_MATERIAL_PRODUCTION_PROTO table in the *Data Schema*.

```
public IloMSMaterialProduction newMaterialProduction(IloMSAbstractActivity absAct,
IloMSMaterial material, IloNum quantity)
```

This method defines a production or consumption (if `quantity` is negative) of the `material` by the `activity`. All modes of this activity will produce/consume the `material` in the same way. If `absAct` is an activity prototype, the material production will have a reference to the corresponding recipe. For related information see the PPO_MATERIAL_PRODUCTION_PROTO table in the *Data Schema*.

```
public IloMSMode newMode(IloMSAbstractActivity activity)
```

This method creates and returns a new mode for the given activity. For related information see the PPO_MODE_PROTO table in the *Data Schema*.

```
public static IloMSModel newModel()
```

This method creates and returns a new `IloMSModel` object owning its own memory allocation environment. For related information see the PPO_MODEL table in the *Data Schema*.

```
public IloMSOptimizationProfile newOptimizationProfile(IloMSOptimizationProfile
profile)
```

This function creates and returns a new optimization profile, which is a copy of the given profile For related information see the PPO_OPTIMIZATION_PROFILE table in the *Data Schema*.

```
public IloMSOptimizationProfile newOptimizationProfile()
```

This function creates and returns a new optimization profile. For related information see the PPO_OPTIMIZATION_PROFILE table in the *Data Schema*.

```
public IloMSPlanningSolution newPlanningSolution()
```

This method returns a new instance of a planning solution.

```
public IloMSPrecedence newPrecedence(IloMSAbstractActivity pred,
IloMSAbstractActivity succ, IloMSPrecedenceType type, IloInt delayMin, IloInt
delayMax)
```

This method creates and returns a new precedence constraint between the predecessor activity `pred` and the successor activity `succ`.

The `type` of the precedence constraint can be `IloMSPrecedenceType::StartToStart` for start to start of the activities, `IloMSPrecedenceType::StartToEnd` for start to end, `IloMSPrecedenceType::EndToStart` for end to start, or `IloMSPrecedenceType::EndToEnd` for end to end.

The parameters `delayMin` and `delayMax` are used to set the minimal and maximal delays for the constraint.

For a precedence constraint of type `IloMSPrecedenceType::StartToStart`, `delayMin` is the minimal delay between the start times of the two activities. This minimal delay must be positive or null. The special value -1 indicates that no minimal delay constraint applies. The parameter `delayMax` is the maximal delay between the start times of the two activities. This maximal delay must be positive or null. The special value -1 indicates that no maximal delay constraint applies.

For a precedence constraint of type start to end `IloMSPrecedenceType::StartToEnd`, `delayMin` is the minimal delay between the start time of `pred` and the end time of `succ`. This minimal delay must be positive or null. The special value -1 indicates that no minimal delay constraint applies. The parameter `delayMax` is the maximal delay between the start time of `pred` and the end time of `succ`. This maximal delay must be positive or null. The special value -1 indicates that no maximal delay constraint applies.

For a precedence constraint of type end to start `IloMSPrecedenceType::EndToStart`, `delayMin` is the minimal delay between the end time of `pred` and the start time of `succ`. This minimal delay must be positive or null. The special value -1 indicates that no minimal delay constraint applies. The parameter `delayMax` is the maximal delay between the end time of `pred` and the start time of `succ`. This maximal delay must be positive or null. The special value -1 indicates that no maximal delay constraint applies.

For a precedence constraint of type end to end `IloMSPrecedenceType::EndToEnd`, the parameter `delayMin` is the minimal delay between the end times of the two activities. This minimal delay must be positive or null. The special value -1 indicates that no minimal delay constraint applies. The parameter `delayMax` is the maximal delay between the end times of the two activities. This maximal delay must be positive or null. The special value -1 indicates that no maximal delay constraint applies.

For any type of constraint, an exception is thrown if the given `delayMin` or the given `delayMax` is strictly smaller than -1.

For related information see the PPO_PROD_PROD_PRECED table in the *Data Schema*.

**See Also:** IloMSPrecedenceType

```
public IloMSProcurement newProcurement(IloMSMaterial material, IloNum quantity)
```

This method returns the new procurement with the specified `quantity` for the `material`.

```
public IloMSProcurementToDemandArc newProcurementToDemandArc(IloMSMaterial
material, IloNum quantity, IloMSProcurement pred, IloMSDemand succ)
```

This method creates and returns a new material flow arc between the procurement `pred` and the demand `succ`.

```
public IloMSProcurementToProdArc newProcurementToProdArc(IloMSMaterial material,
```

```
IloNum quantity, IloMSProcurement pred, IloMSProductionOrder succ)
```

This method creates and returns a new material flow arc between the procurement `pred` and the production order `succ`. This introduces an implicit temporal constraint between the procurement receipt time and a consumer activity of `succ` consuming the `material`.

```
public IloMSProcurementToStorageArc newProcurementToStorageArc(IloMSMaterial
material, IloNum quantity, IloMSProcurement pred)
```

This deprecated method creates and returns a new material flow arc between the procurement `pred` and stock.

```
public IloMSProdToDemandArc newProdToDemandArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder pred, IloMSDemand succ)
```

This method creates and returns a new material flow arc between the predecessor production order `pred` and the successor demand `succ`.

The `quantity` is in this version indicative and not taken into account in the optimization.

For related information see the PPO_PROD_TO_DEMAND_ARC table in the *Data Schema*.

```
public IloMSProdToStorageArc newProdToStorageArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder pred)
```

This deprecated method creates and returns a new material flow arc between the predecessor production order `pred` and stock.

```
public IloMSQuality newQuality()
```

This method returns the new material quality to be used for blending problems.

```
public IloMSRecipe newRecipe()
```

This function creates and returns a new recipe. For related information see the PPO_RECIPE table in the *Data Schema*.

```
public IloMSRecipeFamily newRecipeFamily()
```

This function creates and returns a new recipe family. For related information see the PPO_RECIPE_FAMILY table in the *Data Schema*.

```
public IloMSRecipeFamilyFilter newRecipeFamilyFilter(IloMSScope scope)
```

This function creates and returns a new recipe family filter on a scope. For related information see the PPO_RECIPE_FAMILY_FILTER table in the *Data Schema*.

```
public IloMSResource newResource(IloInt capacity)
```

This method creates and returns a new resource with the given `capacity`. An exception is thrown if the given `capacity` is negative. For related information see the PPO_RESOURCE table in the *Data Schema*.

```
public IloMSResourceCapacityCostFunction newResourceCapacityCostFunction()
```

This method defines a resource capacity cost function. The class `IloMSResourceCapacityCostFunction` is used to evaluate the cost of using a resource over time. For related information see the PPO_RESOURCE_CAPACITY_COST_FCT table in the *Data Schema*.

**See Also:** IloMSResourceCapacityCostFunction

```
public IloMSResourceConstraint newResourceConstraint(IloMSMode mode, IloMSResource
resource, IloInt capacity, IloBool isPrimary)
```

This method creates a new resource constraint for the given `mode`. The parameter `mode` is the mode of the activity for which the resource is required. The parameter `resource` is the required resource, and `capacity` is the required capacity. The parameter `isPrimary` is a Boolean indicating if the given `resource` is the primary resource for the given `mode`.

An exception is thrown if the given `capacity` is negative, if the given `mode` already has a resource constraint for the given `resource`, or if `isPrimary` is true and the given `mode` already has a primary resource constraint.

```
public IloMSResourceFamily newResourceFamily()
```

This method returns the new resource family to be used for aggregating resources in the GUI.

```
public IloMSScope newScope()
```

This function creates and returns a new scope. For related information see the PPO_SCOPE table in the *Data Schema*.

```
public IloMSSetupMatrix newSetupMatrix()
```

This method creates and returns a new setup matrix. For related information see the PPO_SETUP_MATRIX table in the *Data Schema*.

```
public IloMSActivity newStorageActivityPrototype(IloMSActivity producingActivity,
IloMSMaterial producedMaterial)
```

This function creates and returns a new storage activity prototype associated with the corresponding `activity` and producing this `material`.

The `material` specified must be produced by the producing activity in a storage unit associated with the resource referred to as the primary resource by its modes. (See `IloMSStorageUnit.setResource` and `IloMSMaterialProduction.setStorageUnit`). Note that the consuming recipes must also consume from this storage unit.

When a storage activity is declared, the instance of `IloMSMaterialProduction` representing the consumption must also specify a maximum number of incoming pegging arcs equal to one; see `IloMSMaterialProduction.setMaxNumberOfPeggingArcs`.

For related information see the PPO_ACTIVITY_PROTO table in the *Data Schema*.

```
public IloMSStorageToDemandArc newStorageToDemandArc(IloMSMaterial material, IloNum
quantity, IloMSDemand succ)
```

This deprecated method creates and returns a new material flow arc between the storage and the demand `succ`.

```
public IloMSStorageToProdArc newStorageToProdArc(IloMSMaterial material, IloNum
quantity, IloMSProductionOrder succ)
```

This deprecated method creates and returns a new material flow arc between stock and the production order `succ`.

```
public IloMSStorageUnit newStorageUnit()
```

This method creates and returns a new storage unit. For related information see the PPO_STORAGE_UNIT table in the *Data Schema*.

```
public IloMSUnit newUnit()
```

This function creates and returns a new unit of measure. For related information see the PPO_UNIT table in the *Data Schema*.

```
public IloMSIdentifier newUUID()
```

This method creates and returns a Universally Unique Identifier (UUID).

The intent of UUIDs is to enable distributed systems to uniquely identify information without significant central coordination. Thus, anyone can create a UUID and use it to identify something with reasonable confidence that the identifier will never be unintentionally used by anyone for anything else. Information labeled with UUIDs can therefore be used for identifying PPO objects in a relational database.

```
public IloMSRecipe newWasteRecipe(IloMSMaterial material, IloNum wasteCost)
```

This helper function creates and returns a new recipe for discarding material. In the material rebalancing engine, the waste criterion is used to penalize the excess of semi-finished materials to be thrown away due to a mismatch between semi-finished and finished products. In the planning engine, the waste criterion is used to penalize the amount of material to throw away because the shelf life expired. The planning engine enforces that not a single expired item remains in stock. In order to achieve that, it is mandatory to create waste recipes for any material with limited shelf life (a violation in the checker signals that as a warning if waste recipe are forgotten). The total waste cost is defined for the planning as the sum of processing cost of all "planned productions" of waste recipes.

The recipe created has an activity prototype of fixed processing time 1 with a variable processing cost equal to the `wasteCost`. If the material is storable in multiple storage units, this function creates a mode of the activity for consuming from each possible storage unit. For related information see the PPO_RECIPE and PPO_MODE

tables in the *Data Schema*.

```
public void putSetting(const char * valueName, const char * value)
```

This method enables you to set a global parameter. To set parameters dependent upon the optimization profile, use `IloMSOptimizationProfile::putSetting` instead.

The key `slackOnPlannedStartTime` expects an integer value (default is zero if inventory corridors are defined; otherwise `IloMSIntPlusInfinity`). It defines the number of time units of anticipation allowed to the scheduling engine with respect to the start of the time bucket chosen by the planning engine.

The key `slackOnPlannedEndTime` expects an integer value (default is `IloMSIntPlusInfinity`). It defines the number of time units of delay allowed to the scheduling engine with respect to the end of the time bucket chosen by the planning engine.

The key `modeOfPlanningOnly` expects a boolean value (default is false). If the value is true, then the scheduling engine is forced to respect the decisions taken by the planning engine with respect to chosen resources. Note that if super resources have been defined, then the scheduling engine choices are restricted to the subresources of the super resource chosen by the planning. This is a way to limit a possible combinatorial explosion of modes in scheduling.

The key `bShortName` expects a boolean value. If the value is false (default) then the activity names (as labeled on the **Gantt Diagram** for example) are obtained by concatenating the production order name to the activity prototype name. If the value is true, then the activity labels are copied from the activity prototype name with special control characters available that start with the caret ^ symbol. See `IloMSActivity` for more information on the pattern syntax.

The key `MapFile` expects a string value. This parameter allows a background map to be displayed in the **Distribution Planning** view for multi-location problems. By default, the directory data and images are scanned.

The key `MapGraphicFactor` expects a floating point value. This parameter is used to tune the representation of nodes and arcs in the **Distribution Planning** view.

The key `PUSH_BALANCE_MATERIAL_INVENTORIES` expects a boolean value (default is true). If the value is false, material inventories won't be equilibrated (other considerations aside).

```
public void readSettings(const char * filename)
```

This method reads the settings from a file in CSV format. The parameter `filename` is the name of the input file.

```
public void registerLicense(char * plantLicense, int plantSignature)
```

This method registers the runtime license `IloMSModel`.

```
public void removeAllProperties()
```

This method removes all properties.

```
public void removeAutomaticRecipes()
```

This method deletes all recipes with recipe type 'Automatic' and the associated production orders and pegging arcs (even firm ones). Corresponding planned productions are also removed from the planning solution.

Automatic recipes are typically created automatically by PPO when the settings `bAutomaticGenerationInflowRecipes` and `bAutomaticGenerationWasteRecipes` are set to true.

```
public void removeNotFirmedData()
```

This method deletes all production orders and pegging arcs on the current batching solution that do not have a firm quantity.

```
public void removeProperty(const char * name)
```

This method removes the property with the given `name`.

```
public IloBool repeg()
```

This method runs the repegging algorithm on the current model.

```
public void reserveCurrentBatchingSolution()
```

internal

This method retains the current batching solution for possible rollback and creates a copy of the current batching solution for use within a custom batching algorithm.

```
public void rollbackCurrentBatchingSolution()
```

internal

This method deletes the current batching solution and reinstalls an earlier version saved when calling the `reserveCurrentBatchingSolution()` method. A typical use of this method is when a failure occurs in a customized batching algorithm.

```
public void setBatchingHorizon(IloInt horizon)
```

This deprecated method sets the end of horizon to consider when batching. Use `setSchedulingHorizon`.

```
public void setBatchingWeight(IloMSOptimizationCriterion criterion, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setBatchingWeight(IloMSIdentifier criterionId, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setCheckForStop(IloMSCheckForStop hook)
```

This method creates a hook to stop an optimization engine.

```
public void setCurrentBatchingSolution(IloMSBatchingSolution solution)
```

This method sets the specified instance of a batching solution as the current batching solution.

```
public void setCurrentMaterialFamilyType(IloMSIdentifier familyType)
```

This method selects the current material family type for aggregation purpose in the GUI.

```
public void setCurrentOptimizationProfile(IloMSOptimizationProfile profile)
```

This function sets the current optimization profile containing all overridden parameters for engines.

```
public void setCurrentPlanningSolution(IloMSPlanningSolution solution)
```

Set the specified instance of a planning solution as the current planning solution.

```
public void setCurrentRecipeFamilyType(IloMSIdentifier familyType)
```

This method selects the current recipe family type for aggregation purpose in the GUI.

```
public void setCurrentResourceFamilyType(IloMSIdentifier familyType)
```

This method selects the current resource family type for aggregation purpose in the GUI.

```
public void setDateOrigin(IloMSDate date)
```

This method sets the absolute date origin. All relative times (given as integer) are interpreted with respect to this date origin. The given date is understood to be in UTC.

```
public void setDisplayedBucketSequence(IloMSBucketSequence sequence)
```

This function changes the bucket sequence used by the GUI.

It has to be set at least once (except when no bucket is defined).

```
public void setEndMax(IloInt latestEndTime)
```

This method sets the latest possible end time for all the activities of the model.

```
public void setIntDateOrigin(IloInt inSecondsSince_01_01_2001)
```

This method defines the date, expressed in seconds since January 1st 2001 UTC, that corresponds to the time 0 (zero) used in the model. For example, `setIntDateOrigin(31 * 24 * 3600)` means that time 0 corresponds to February 1st, 2001.

```
public void setIntProperty(const char * name, IloInt value)
```

This method sets the value of an integer property named `name`.

```
public void setMakespanOrigin(IloInt makespanOrigin)
```

This method sets the makespan origin, that is, the time that serves as a basis to calculate the makespan. The makespan is the difference between the end time of the last activity and the given `makespanOrigin`.

```
public void setNumProperty(const char * name, IloNum value)
```

This method sets the value of a numeric property named `name`.

```
public void setObject(IloAny object)
```

This method provides a name for the invoking `IloMSModel` object.

This modifier associates a pointer with an external object to the invoking object.

```
public void setOptimizedBucketSequence(IloMSBucketSequence sequence)
```

This function changes the bucket sequence used by the optimizer.

It has to be set at least once (except when no bucket is defined).

```
public void setPlanningHorizon(IloInt horizon)
```

This method sets the end of horizon to consider when planning. The planning engine will take decisions over the optimized bucket sequence until the latest bucket that ends exactly at or before this limit.

Let E be the end of the latest bucket that ends exactly at or before the planning horizon. The planning engine generates only planned deliveries and corresponding planned productions for demands whose earliest possible delivery can start before E (DELIVERY_START_MIN < E). Demands that are to be delivered after E (DELIVERY_START_MIN ≥ E) are taken into account in the inventory corridor by forcing the planning engine to create enough planned productions to stay between the minimum and the maximum number of days of supply at the end of the horizon.

```
public void setPlanningWeight(IloMSOptimizationCriterion criterion, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setPlanningWeight(IloMSIdentifier criterionId, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setProductivityGrain(IloInt grain)
```

This method is used to set the precision of the productivity computation. The default value is 100.

```
public void setSchedulingHorizon(IloInt horizon)
```

This method sets the end time of the scheduling horizon of the current optimization profile, and thereby affects the batching engine, scheduling engine, and GUI behavior.

The batching engine does not consider planned productions or planned deliveries that start in time buckets equal to or after the scheduling horizon. Therefore, such planning information is not converted into production orders and material flow arcs, and no corresponding activities are created. In the scheduling solution, the nondelivery cost for a demand that has its delivery end max greater than this horizon is computed based on the pegged quantity, not on the quantity of the demand itself.

Integrated planning and scheduling problems (where both planning and scheduling are required) have a further consideration regarding solution display in the GUI. If the scheduling horizon is less than the planning horizon, then for time periods before the scheduling horizon the solution data is based on the scheduling solution. For time periods after the scheduling horizon, the planning solution data (planned productions and planned deliveries) are displayed in the appropriate views of the GUI.

```
public void setSchedulingWeight(IloMSOptimizationCriterion criterion, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setSchedulingWeight(IloMSIdentifier criterionId, IloNum value)
```

This method sets the weight of the given criterion to the given value.

```
public void setSlackOnPlannedEndTime(IloInt value)
```

This method sets the slack applied by the scheduling engine to the end time of production orders generated from the planning solution.

```
public void setSlackOnPlannedStartTime(IloInt value)
```

This method sets the slack applied by the scheduling engine to the start time of production orders generated from the planning solution.

```
public void setStartMin(IloInt earliestStartTime)
```

This method sets the earliest possible start time for all the activities of the model.

```
public void setStringProperty(const char * name, const char * value)
```

This method sets the value of a string property named `name`.

```
public void setTimeCheckingTolerance(IloInt toleranceInTimeUnits)
```

This method sets the tolerance used for filtering nonfatal error messages with regard to time and capacity violations in the checkers.

```
public void setTimeUnit(IloInt inSeconds)
```

This method defines the time unit, expressed in seconds, used throughout the model. For example, `setTimeUnit(60)` means that an activity of duration 1 takes one minute in reality (`setTimeUnit(60)` = one minute, `setTimeUnit(3600)` = one hour, etc.). This parameter has no impact on the optimization.

```
public void setTimeZone(const char * identifier)
```

This method sets the identifier of the time zone.

A list of possible time zone settings is available in the *Date and time display* section of the reference material.

```
public void setTraceFile(const char * filename=0, ios_base::openmode
mode=ios_base::out)
```

This method writes information about the optimization run to an output device. If the passed filename is 0 (zero) then messages are written to the standard output. The detail of information reported depends on the trace level (see `IloMSModel::setTraceLevel`). The parameter `filename` is the name of the output trace file.

```
public void setTraceLevel(IloInt level)
```

This method serves to activate or deactivate the tracing of the optimization run. The possible values for the parameter `level` are 0 (zero) or 1. If `level` is 0 then trace is deactivated; if it is 1, then trace is activated.

```
public void setWeight(IloMSOptimizationCriterion criterion, IloNum value)
```

This method sets the weight of the given criterion to the given `value`.

```
public void setWeight(IloMSIdentifier criterionId, IloNum value)
```

This method sets the weight of the given criterion to the given `value`. The possible criteria are: TotalCleanupCost, TotalEarlinessCost, TotalInventoryCost (excess), TotalInventoryDeficitCost, TotalNonDeliveryCost, TotalRevenue, TotalProcessingCost (mode costs), TotalSetupCost, TotalSetupTime, TotalTardinessCost, TotalUnperformedCost

Other values are possible but their use with the scheduling engine is discouraged (TotalIdleCost, TotalResourceCost, Makespan, MaxEarlinessCost, MaxTardinessCost).

```
public IloBool solve() const
```

This method solves the problem using the appropriate engines. It may use the planning engine, the lot sizing engine and/or the scheduling engine, depending on the requirements of the current optimization profile. It returns true if a solution is found; false otherwise. Note that in case of success the previous solutions are no longer valid. A new planning/batching/scheduling solution instance may have replaced the previous current planning/batching/scheduling solution. For this reason, do not use a reference to a solution obtained before solving.

```
public void write(const char * filename, IloBool writeSettings)
```

This method writes the problem instance to a CSV file. The parameter `filename` is the name of the output file. The parameter `writeSettings` is a Boolean specifying if the parameter settings shall be saved in the file as well.

```
public void writeSettings(const char * filename)
```

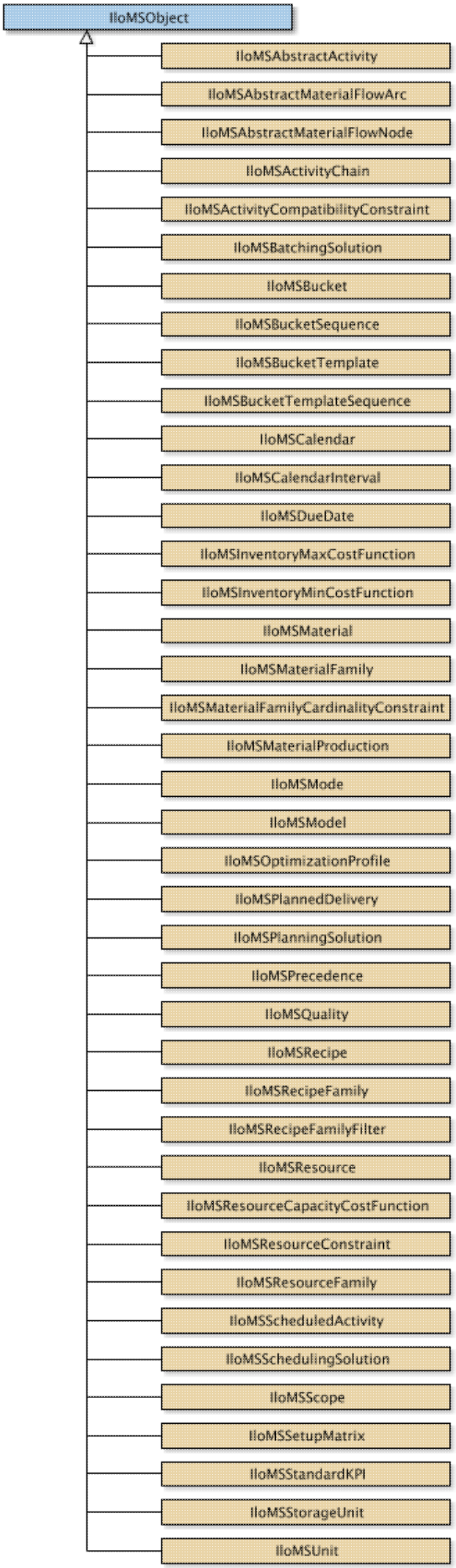
This method writes the settings to a file in CSV format. The parameter `filename` is the name of the output file.

```
public void writeSolution(const char * filename, IloMSSchedulingSolution solution)
```

This method writes the given solution to a file in CSV format. The parameter `filename` is the name of the output file. The parameter `solution` is the solution to be written.

# Class IloMSObject

**Definition file:** ilplant/object.h
**Library:** plant


IloMSObject
IloMSAbstractActivity
IloMSAbstractMaterialFlowArc
IloMSAbstractMaterialFlowNode
IloMSActivityChain
IloMSActivityCompatibilityConstraint
IloMSBatchingSolution
IloMSBucket
IloMSBucketSequence
IloMSBucketTemplate
IloMSBucketTemplateSequence
IloMSCalendar
IloMSCalendarInterval
IloMSDueDate
IloMSInventoryMaxCostFunction
IloMSInventoryMinCostFunction
IloMSMaterial
IloMSMaterialFamily
IloMSMaterialFamilyCardinalityConstraint
IloMSMaterialProduction
IloMSMode
IloMSModel
IloMSOptimizationProfile
IloMSPlannedDelivery
IloMSPlanningSolution
IloMSPrecedence
IloMSQuality
IloMSRecipe
IloMSRecipeFamily
IloMSRecipeFamilyFilter
IloMSResource
IloMSResourceCapacityCostFunction
IloMSResourceConstraint
IloMSResourceFamily
IloMSScheduledActivity
IloMSSchedulingSolution
IloMSScope
IloMSSetupMatrix
IloMSStandardKPI
IloMSStorageUnit
IloMSUnit

The class `IloMSObject` is used to represent all types of Plant PowerOps objects.
This mother class contains all the general services (`get/setName`, `get/setObject`, and named properties) common to all classes used to model the problem and describe the solution. The named properties mechanism is a simple way to add fields to existing concepts without subclassing.

| Method Summary | |
|---|---|
| public void | display(ostream & out) const |
| public IloMSIdentifier | getIdentifier() const |
| public IloInt | getIntProperty(const char * name) const |
| public IloMSModel | getModel() const |
| public const char * | getName() const |
| public IloNum | getNumProperty(const char * name) const |
| public IloAny | getObject() const |
| public const char * | getStringProperty(const char * name) const |
| public IloBool | hasIdentifier() const |
| public IloBool | isPropertyDefined(const char * name) const |
| public void | removeAllProperties() |
| public void | removeProperty(const char * name) |
| public void | setIntProperty(const char * name, IloInt) |
| public void | setName(const char * name) |
| public void | setNumProperty(const char * name, IloNum) |
| public void | setObject(IloAny object) |
| public void | setStringProperty(const char * name, const char *) |
| public const char * | toString() const |

## Methods

`public void` **`display`**`(ostream & out) const`

This method displays the object on the output stream `out`.

`public IloMSIdentifier` **`getIdentifier`**`() const`

This accessor returns the identifier of the invoking object. An exception is thrown if the invoking object has no identifier.

`public IloInt` **`getIntProperty`**`(const char * name) const`

This method returns the value of the integer property named `name`.

`public IloMSModel` **`getModel`**`() const`

This method returns the instance of `IloMSModel` on which the invoking object has been instantiated.

```
public const char * getName() const
```

This accessor returns the name of the invoking object.

```
public IloNum getNumProperty(const char * name) const
```

This method returns the value of the numeric property named `name`.

```
public IloAny getObject() const
```

This accessor returns a pointer to the external object associated with the invoking object.

```
public const char * getStringProperty(const char * name) const
```

This method returns the value of the string property named `name`.

```
public IloBool hasIdentifier() const
```

This predicate returns true if the invoking object has an identifier.

```
public IloBool isPropertyDefined(const char * name) const
```

This method allows testing for the existence of a named property.

```
public void removeAllProperties()
```

This method removes all properties.

```
public void removeProperty(const char * name)
```

This method removes the property with the given `name`.

```
public void setIntProperty(const char * name, IloInt)
```

This method sets the value of an integer property named `name`.

```
public void setName(const char * name)
```

This modifier provides a name for the invoking object.

```
public void setNumProperty(const char * name, IloNum)
```

This method sets the value of a numeric property named `name`.

```
public void setObject(IloAny object)
```

This modifier associates a pointer with an external object to the invoking object.

```
public void setStringProperty(const char * name, const char *)
```

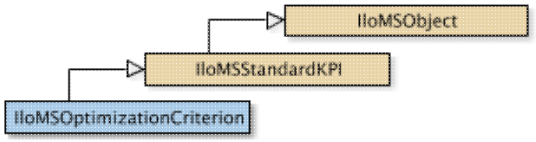
This method sets the value of a string property named `name`.

```
public const char * toString() const
```

This method returns a character string containing a display message.

# Class IloMSOptimizationCriterion

**Definition file:** ilplant/kpi.h
**Library:** plant




The `IloMSOptimizationCriterion` class is used to represent optimization criteria.
An optimization criterion is a specific type of Key Performance Indicator that can be optimized by the PPO optimization engines.

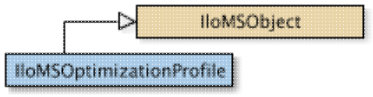
**See Also:** IloMSStandardKPI

| Inherited Methods from `IloMSStandardKPI` |
|---|
| `getValue, getValue, getValue, isEditable, isVisible, setVisible, toMinimize` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

# Class IloMSOptimizationProfile

**Definition file:** ilplant/optimizationprofile.h
**Library:** plant




The `IloMSOptimizationProfile` class is used to represent optimization profiles.
Optimization profiles are containers of parameters to tune the optimization engines.

| Method Summary | |
|---|---|
| public IloBool | areBatchIntegralityConstraintsEnabledForPlanning() |
| public IloBool | areBatchSizeConstraintsEnabledForPlanning() |
| public IloBool | areIntegralityAfterBatchingHorizonConstraintsEnabledForPlanning() |
| public IloBool | areReservoirCapacityConstraintsEnabledForScheduling() |
| public IloBool | areResourceCapacityConstraintsEnabledForPlanning() |
| public IloBool | areSetupConstraintsEnabledForPlanning() |
| public IloBool | areShelfLifeAndMaturityConstraintsEnabledForPlanning() |
| public IloBool | areStockConstraintsEnabledForPlanning() |
| public void | display(ostream & stream) const |
| public IloMSBatchingAlgorithm | getBatchingAlgorithm() const |
| public IloInt | getBatchingHorizon() const |
| public IloNum | getBatchingRelativeGap() const |
| public IloNum | getBatchingTimeLimit() const |
| public IloNum | getBatchingWeight(IloMSOptimizationCriterion criterion) const |
| public IloMSPlanningAlgorithm | getPlanningAlgorithm() const |
| public IloInt | getPlanningHorizon() const |
| public IloNum | getPlanningRelativeGap() const |
| public IloNum | getPlanningTimeLimit() const |
| public IloNum | getPlanningWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getRebalancingTimeLimit() const |
| public IloInt | getSchedulingHorizon() const |
| public IloNum | getSchedulingTimeLimit() const |
| public IloNum | getSchedulingWeight(IloMSOptimizationCriterion criterion) const |
| public IloNum | getWeight(IloMSOptimizationCriterion criterion) const |
| public IloBool | isBatchingRequired() const |
| public IloBool | isPlanningRequired() const |
| public IloBool | isRebalancingRequired() const |
| public IloBool | isRepeggingRequired() const |
| public IloBool | isSchedulingOrPostSchedulingRequired() const |
| public IloBool | isSchedulingRequired() const |
| public void | putSetting(const char * key, const char * value) |

| | |
|---|---|
| public void | setBatchingAlgorithm(IloMSBatchingAlgorithm algo) |
| public void | setBatchingHorizon(IloInt horizon) |
| public void | setBatchingRelativeGap(IloNum gap) |
| public void | setBatchingRequired(IloBool batchingRequired) |
| public void | setBatchingTimeLimit(IloNum timeLimit) |
| public void | setBatchingWeight(IloMSOptimizationCriterion criterion, IloNum weight) |
| public void | setBatchIntegralityConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setBatchSizeConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setIntegralityAfterBatchingHorizonConstraintsEnabledForPlanning(Il enabled) |
| public void | setPlanningAlgorithm(IloMSPlanningAlgorithm algo) |
| public void | setPlanningHorizon(IloInt horizon) |
| public void | setPlanningRelativeGap(IloNum gap) |
| public void | setPlanningRequired(IloBool planningRequired) |
| public void | setPlanningTimeLimit(IloNum timeLimit) |
| public void | setPlanningWeight(IloMSOptimizationCriterion criterion, IloNum weight) |
| public void | setRebalancingRequired(IloBool rebalancingRequired) |
| public void | setRebalancingTimeLimit(IloNum timeLimit) |
| public void | setRepeggingRequired(IloBool repeggingRequired) |
| public void | setReservoirCapacityConstraintsEnabledForScheduling(IloBool enabled) |
| public void | setResourceCapacityConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setSchedulingHorizon(IloInt horizon) |
| public void | setSchedulingRequired(IloBool schedulingRequired) |
| public void | setSchedulingTimeLimit(IloNum timeLimit) |
| public void | setSchedulingWeight(IloMSOptimizationCriterion criterion, IloNum weight) |
| public void | setScope(IloMSScope scope) |
| public void | setSetupConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setShelfLifeAndMaturityConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setStockConstraintsEnabledForPlanning(IloBool enabled) |
| public void | setWeight(IloMSOptimizationCriterion criterion, IloNum weight) |

| **Inherited Methods from `IloMSObject`** |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloBool` **`areBatchIntegralityConstraintsEnabledForPlanning`**`()`

This method returns true if the constraint on recipes requiring integer batch sizes is enforced for the planning module in this optimization profile.

`public IloBool` **`areBatchSizeConstraintsEnabledForPlanning`**`()`

This method returns true if the constraints on minimal and maximal batch sizes are enforced for the planning module in this optimization profile.

`public IloBool` **`areIntegralityAfterBatchingHorizonConstraintsEnabledForPlanning`**`()`

This accessor returns the relative gap used to stop shipping optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

This modifier sets the relative gap used to stop shipping optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

This method returns false if a relaxation window has been defined between the batching and the planning horizon. If false is returned, all binary and integer variables are considered as floating point variables.

`public IloBool` **`areReservoirCapacityConstraintsEnabledForScheduling`**`()`

This method returns true if the constraints on the maximum capacity of storage units are enforced by the scheduling engine in this optimization profile. Otherwise it returns false. Enforcing the maximum capacity constraint on storage units consumes processing time and memory, and may lead to infeasibilities.

`public IloBool` **`areResourceCapacityConstraintsEnabledForPlanning`**`()`

This method returns true if the constraint on resource finite capacity is enforced for the planning module in this optimization profile.

`public IloBool` **`areSetupConstraintsEnabledForPlanning`**`()`

This method returns true if the constraint on setup times is enforced for the planning module in this optimization profile.

`public IloBool` **`areShelfLifeAndMaturityConstraintsEnabledForPlanning`**`()`

This method returns true if the constraints on maturity and shelf life of materials are enforced for the planning module in this optimization profile. If true, the planning module will prohibit demand delivery or material consumption by production orders if the material is not mature or already expired.

`public IloBool` **`areStockConstraintsEnabledForPlanning`**`()`

This method returns true if the constraints on maximum inventory are enforced for the planning module in this optimization profile.

```
public void display(ostream & stream) const
```

This method displays the optimization profile in the stream passed as argument.

```
public IloMSBatchingAlgorithm getBatchingAlgorithm() const
```

This method returns the batching algorithm that the solve procedure uses if batching is required.

```
public IloInt getBatchingHorizon() const
```

This deprecated method returns the time of the batching horizon end. Use the method `getSchedulingHorizon`.

```
public IloNum getBatchingRelativeGap() const
```

This accessor returns the relative gap used to stop batching optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

```
public IloNum getBatchingTimeLimit() const
```

This accessor returns the time limit specific to the batching engine. Only the optimized batching engine takes it into account.

```
public IloNum getBatchingWeight(IloMSOptimizationCriterion criterion) const
```

This accessor returns the weight for the given `criterion` in the invoking optimization profile.

```
public IloMSPlanningAlgorithm getPlanningAlgorithm() const
```

This method returns the planning algorithm that the planning engine will use.

```
public IloInt getPlanningHorizon() const
```

This accessor returns the time of the planning horizon end. The planning engine will take decisions over the optimized bucket sequence until the latest bucket that ends exactly at or before this limit. See `setPlanningHorizon` for more information.

```
public IloNum getPlanningRelativeGap() const
```

This accessor returns the relative gap used to stop planning optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

```
public IloNum getPlanningTimeLimit() const
```

This accessor returns the time limit specific to the planning engine.

```
public IloNum getPlanningWeight(IloMSOptimizationCriterion criterion) const
```

This accessor returns the weight for the given `criterion` in the invoking optimization profile.

```
public IloNum getRebalancingTimeLimit() const
```

This accessor returns the time limit specific to the material rebalancing engine.

```
public IloInt getSchedulingHorizon() const
```

This method returns the end time of the scheduling horizon. The scheduling horizon affects the batching engine, scheduling engine, and GUI behavior. See the method `setSchedulingHorizon` for more information.

```
public IloNum getSchedulingTimeLimit() const
```

This accessor returns the time limit specific to the scheduling engine.

```
public IloNum getSchedulingWeight(IloMSOptimizationCriterion criterion) const
```

This accessor returns the weight for the given `criterion` in the invoking optimization profile.

```
public IloNum getWeight(IloMSOptimizationCriterion criterion) const
```

This accessor returns the weight for the given `criterion` in the invoking optimization profile.

```
public IloBool isBatchingRequired() const
```

This method returns true if there is a need for a batching and pegging pass with the batching engine.

```
public IloBool isPlanningRequired() const
```

This method returns true if there is a need for a planning pass with the planning engine.

```
public IloBool isRebalancingRequired() const
```

This method returns true if there is a need for a material rebalancing pass.

```
public IloBool isRepeggingRequired() const
```

This method returns true if there is a need for a final demand repegging pass.

```
public IloBool isSchedulingOrPostSchedulingRequired() const
```

This method returns true if there is a need for a scheduling or post-processing pass based on existing scheduling solution.

```
public IloBool isSchedulingRequired() const
```

This method returns true if there is a need for a scheduling pass with the scheduling engine.

```
public void putSetting(const char * key, const char * value)
```

This modifier associates a `value` with the `key` in the internal settings of the invoking optimization profile.

Note that the value must be interpretable as string or integer, floating point, or Boolean within the semantics of the corresponding setting. See `IloMSModel::putSetting` for more information on the documented settings.

**See Also:** IloMSActivity, IloMSModel

```
public void setBatchingAlgorithm(IloMSBatchingAlgorithm algo)
```

This method sets the batching algorithm that the solve procedure uses if batching is required.

```
public void setBatchingHorizon(IloInt horizon)
```

This deprecated method sets the time of the batching horizon end. Use `setSchedulingHorizon`.

```
public void setBatchingRelativeGap(IloNum gap)
```

This modifier sets the relative gap used to stop batching optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

```
public void setBatchingRequired(IloBool batchingRequired)
```

This method is used to express the need for a batching and pegging pass with the batching engine.

```
public void setBatchingTimeLimit(IloNum timeLimit)
```

This modifier sets the time limit expressed in seconds specific to the batching engine. Only the optimized batching engine takes it into account.

```
public void setBatchingWeight(IloMSOptimizationCriterion criterion, IloNum weight)
```

This modifier associates a `weight` for the `criterion` in the invoking optimization profile.

```
public void setBatchIntegralityConstraintsEnabledForPlanning(IloBool enabled)
```

When this method is called with false as parameter, the planning module relaxes the constraints enforcing integer batch sizes.

```
public void setBatchSizeConstraintsEnabledForPlanning(IloBool enabled)
```

When this method is called with false as parameter, the planning module relaxes the minimal and maximal batch size constraints.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking optimization profile. An exception is thrown if the given `identifier` is already used.

```
public void setIntegralityAfterBatchingHorizonConstraintsEnabledForPlanning(IloBool
enabled)
```

When this method is called with false as parameter, a relaxation window is defined between the scheduling and the planning horizon. The batching engine uses the horizon of the scheduling horizon.

```
public void setPlanningAlgorithm(IloMSPlanningAlgorithm algo)
```

This method sets the planning algorithm that the planning engine uses.

```
public void setPlanningHorizon(IloInt horizon)
```

This modifier sets the time of the planning horizon end. The planning engine will take decisions over the optimized bucket sequence until the latest bucket that ends exactly at or before this limit.

Let E be the end of the latest bucket that ends exactly at or before the planning horizon. The planning engine generates only planned deliveries and corresponding planned productions for demands whose earliest possible delivery can start before E (DELIVERY_START_MIN < E). Demands that are to be delivered after E (DELIVERY_START_MIN ≥ E) are taken into account in the inventory corridor by forcing the planning engine to create enough planned productions to stay between the minimum and the maximum number of days of supply at the end of the horizon.

.

```
public void setPlanningRelativeGap(IloNum gap)
```

This modifier sets the relative gap used to stop planning optimization. For example, if the relative gap is 0.01, optimization will stop as soon as it is proven that the cost of the current solution is not more than 1% above the optimum.

```
public void setPlanningRequired(IloBool planningRequired)
```

This method is used to express the need for a planning pass with the planning engine.

```
public void setPlanningTimeLimit(IloNum timeLimit)
```

This modifier sets the time limit expressed in seconds specific to the planning engine.

```
public void setPlanningWeight(IloMSOptimizationCriterion criterion, IloNum weight)
```

This modifier associates a `weight` for the `criterion` in the invoking optimization profile.

```
public void setRebalancingRequired(IloBool rebalancingRequired)
```

This method is used to express the need for a material rebalancing pass with the material rebalancing engine.

```
public void setRebalancingTimeLimit(IloNum timeLimit)
```

This modifier sets the time limit expressed in seconds specific to the material rebalancing engine.

```
public void setRepeggingRequired(IloBool repeggingRequired)
```

This method is used to express the need for a final demand repegging pass.

```
public void setReservoirCapacityConstraintsEnabledForScheduling(IloBool enabled)
```

When this method is called with true as parameter, the engine module enforces the maximum capacity constraints of storage units. When it is called with false as parameter, the scheduling module relaxes the maximum capacity constraints of storage units. Enforcing the maximum capacity constraint on storage units consumes processing time and memory, and may lead to infeasibilities.

```
public void setResourceCapacityConstraintsEnabledForPlanning(IloBool enabled)
```

When this method is called with false as parameter, the planning module considers all resources as infinite capacity resources.

```
public void setSchedulingHorizon(IloInt horizon)
```

This method sets the end time of the scheduling horizon, and thereby affects the batching engine, scheduling engine, and GUI behavior.

The batching engine does not consider planned productions or planned deliveries that start in time buckets equal to or after the scheduling horizon. Therefore, such planning information is not converted into production orders and material flow arcs, and no corresponding activities are created. In the scheduling solution, the nondelivery cost for a demand that has its delivery end max greater than this horizon is computed based on the pegged quantity, not on the quantity of the demand itself.

Integrated planning and scheduling problems (where both planning and scheduling are required) have a further consideration regarding solution display in the GUI. If the scheduling horizon is less than the planning horizon, then for time periods before the scheduling horizon the solution data is based on the scheduling solution. For time periods after the scheduling horizon, the planning solution data (planned productions and planned deliveries) are displayed in the appropriate views of the GUI.

```
public void setSchedulingRequired(IloBool schedulingRequired)
```

This method is used to express the need for a scheduling pass with the scheduling engine.

```
public void setSchedulingTimeLimit(IloNum timeLimit)
```

This modifier sets the time limit expressed in seconds specific to the scheduling engine.

```
public void setSchedulingWeight(IloMSOptimizationCriterion criterion, IloNum
weight)
```

This modifier associates a `weight` for the `criterion` in the invoking optimization profile.

```
public void setScope(IloMSScope scope)
```

This modifier associates a `scope` to the invoking optimization profile.

```
public void setSetupConstraintsEnabledForPlanning(IloBool enabled)
```

When this method is called with false as parameter the planning module relaxes setup time constraints.

```
public void setShelfLifeAndMaturityConstraintsEnabledForPlanning(IloBool enabled)
```

When this method is called with false as parameter, the planning module does not take into account shelf life and maturity constraints. In such a case material consumption and demand delivery by production orders is valid even if the material is not yet mature or is already expired.

```
public void setStockConstraintsEnabledForPlanning(IloBool enabled)
```

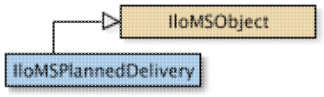
When this method is called with false as parameter, the planning module relaxes the maximum inventory constraints on all materials.

```
public void setWeight(IloMSOptimizationCriterion criterion, IloNum weight)
```

This modifier associates a `weight` for the `criterion` in the invoking optimization profile.

# Class IloMSPlannedDelivery

**Definition file:** ilplant/planneddelivery.h
**Library:** plant




The class `IloMSPlannedDelivery` is used in a planning solution to represent the amount of material that exits from inventory in a period of time to satisfy a specific demand. This amount is in addition to any already-fixed pegging arcs to the demand.
Satisfying a demand is to be understood as shipping a quantity of material in a certain time window determined by the planning engine [planned time min;planned time max]. A demand may be satisfied in several shipments occurring at different moments; therefore several instances of `IloMSPlannedDelivery` may coexist for the same demand in the same planning solution with different time windows. These time windows correspond to time buckets.

Instances of this class are owned by a planning solution.

**See Also:** IloMSPlanningSolution, IloMSPlanningEngine, IloMSDemand

| Method Summary | |
|---|---|
| public void | display(ostream & stream) const |
| public IloMSDemand | getDemand() const |
| public IloNum | getFirmQuantityMax() const |
| public IloNum | getFirmQuantityMin() const |
| public IloInt | getPlannedTimeMax() const |
| public IloInt | getPlannedTimeMin() const |
| public IloNum | getQuantity() const |
| public IloBool | isFirm() const |
| public void | setDemand(IloMSDemand dem) |
| public void | setFirm(IloBool firm) |
| public void | setFirmQuantityMax(IloNum firmQuantityMax) |
| public void | setFirmQuantityMin(IloNum firmQuantityMin) |
| public void | setPlannedTimeMax(IloInt endMax) |
| public void | setPlannedTimeMin(IloInt startMin) |
| public void | setQuantity(IloNum qty) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public void **display**(ostream & stream) const

This method displays the planned delivery in the stream passed as argument.

```
public IloMSDemand getDemand() const
```

This method returns the demand that the invoking object satisfies partly or completely.

```
public IloNum getFirmQuantityMax() const
```

This method returns the maximum quantity to be shipped in the specified time bucket for the corresponding demand.

```
public IloNum getFirmQuantityMin() const
```

This method returns the minimum quantity to be shipped in the specified time bucket for the corresponding demand.

```
public IloInt getPlannedTimeMax() const
```

This method returns the planned time before which this demand delivery can occur. When determined by the planning engine, this is typically the end time of a bucket.

```
public IloInt getPlannedTimeMin() const
```

This method returns the minimal planned time at which an amount of material is planned to be shipped for satisfying a demand. When determined by the planning engine, this is typically the start time of a bucket.

```
public IloNum getQuantity() const
```

This method returns the quantity to ship for a demand in a time window. This time window is typically a time bucket determined by the planning engine.

```
public IloBool isFirm() const
```

This method returns the status of the invoking object (firm quantity min is strictly positive).

```
public void setDemand(IloMSDemand dem)
```

This method sets the demand that the invoking object satisfies partly or completely.

```
public void setFirm(IloBool firm)
```

This method sets the firm quantity min and max to the current quantity.

A firm planned delivery forces the planning engine to respect the corresponding shipped quantity for the corresponding demand in the corresponding bucket.

```
public void setFirmQuantityMax(IloNum firmQuantityMax)
```

This method sets the maximum quantity to be shipped in the specified time bucket for the corresponding demand.

```
public void setFirmQuantityMin(IloNum firmQuantityMin)
```

This method sets the minimum quantity to be shipped in the specified time bucket for the corresponding demand.

```
public void setPlannedTimeMax(IloInt endMax)
```

This method sets the planned time before which the demand delivery must occur. The demand delivery has to occur before `endMax`.

```
public void setPlannedTimeMin(IloInt startMin)
```

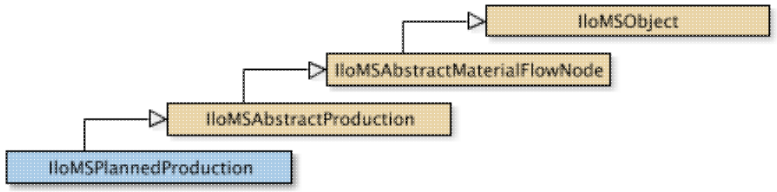
This method sets the minimal planned time at which the delivery can occur. The demand delivery has to occur at `startMin` or later.

```
public void setQuantity(IloNum qty)
```

This method sets the quantity `qty` to ship in a time window for the specific demand to which it is attached.

# Class IloMSPlannedProduction

**Definition file:** ilplant/plannedprod.h
**Library:** plant




The class `IloMSPlannedProduction` is used to represent the quantity of a recipe that is planned to be executed in a period of time.
Instances of the class `IloMSPlannedProduction` are owned by a planning solution.

Planned productions can be seen as future production orders planned to be executed within a time window. The batching engine typically takes this information as input data, and splits the quantity of work planned into several production orders in order that the batch size constraints are respected, and the cardinality constraints are enforced on the material flow.

Several planned productions may coexist for the same time bucket in the planning solution. Each one typically uses a different set of resources; this is why an instance of `IloMSPlannedProduction` refers to the mode prototypes of the original recipes chosen by the planning engine.

**See Also:** IloMSPlanningSolution, IloMSPlanningEngine, IloMSBatchingEngine, IloMSAbstractProduction, IloMSRecipe, IloMSMode

| Method Summary | |
|---|---|
| public void | addMode(IloMSMode modePrototype) |
| public void | display(ostream & stream) const |
| public IloNum | getFirmBatchSizeMax() const |
| public IloNum | getFirmBatchSizeMin() const |
| public IloMSMode | getMode(IloInt activityPrototypeIndex) const |
| public IloInt | getNumberOfModes() const |
| public IloInt | getPlannedNumberOfBatches() const |
| public IloInt | getPlannedTimeMax() const |
| public IloInt | getPlannedTimeMin() const |
| public ILOMSDEPRECATED IloNum | getQuantity() const |
| public void | setFirmBatchSizeMax(IloNum ub) |
| public void | setFirmBatchSizeMin(IloNum lb) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setPlannedNumberOfBatches(IloInt number) |
| public void | setPlannedTimeMax(IloInt value) |
| public void | setPlannedTimeMin(IloInt value) |
| public ILOMSDEPRECATED void | setQuantity(IloNum qty) |
| public void | setRecipe(IloMSRecipe recipe) |

| Inherited Methods from `IloMSAbstractProduction` |
|---|

| addPlannedMode, getBatchSize, getNumberOfPlannedModes, getPlannedMode, getPlannedTimeMax, getPlannedTimeMin, getRecipe, isFirm, setBatchSize, setFirm, setPlannedTimeMax, setPlannedTimeMin |
|---|

| Inherited Methods from IloMSAbstractMaterialFlowNode |
|---|
| getCategory, getMaterialFlowNodeType, isFirm, setCategory, setFirm, toDemand, toProductionOrder |

| Inherited Methods from IloMSObject |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public void addMode(IloMSMode modePrototype)
```

This method adds a reference to the mode prototype in the original recipe. This mode prototype corresponds to a decision of the planning engine with respect to which resource was used among the alternatives defined in the original recipe.

```
public void display(ostream & stream) const
```

This method displays the planned production in the `stream` passed as argument.

```
public IloNum getFirmBatchSizeMax() const
```

This method retrieves the batch size upper bound used by the planning engine for execution of the recipe in the corresponding time bucket.

```
public IloNum getFirmBatchSizeMin() const
```

This method retrieves the batch size lower bound used by the planning engine for execution of the recipe in the corresponding time bucket.

```
public IloMSMode getMode(IloInt activityPrototypeIndex) const
```

This method returns the mode of#-th activity prototype. This mode prototype corresponds to a decision of the planning engine with respect to which resource is to be used among the alternatives defined in the original recipe.

```
public IloInt getNumberOfModes() const
```

This method returns the number of mode prototypes corresponding to the decisions of the planning engine, with respect to which resource was used among the alternatives defined in the original recipe.

```
public IloInt getPlannedNumberOfBatches() const
```

This method returns the planned number of batches found by the planning engine in the associated time bucket.

```
public IloInt getPlannedTimeMax() const
```

This method returns the end time of the time bucket in which the planning engine decided to plan the invoked production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedEndTime"` is set to zero.

```
public IloInt getPlannedTimeMin() const
```

This method returns the start time of the time bucket in which the planning engine decided to plan the invoked production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedStartTime"` is set to zero.

```
public ILOMSDEPRECATED IloNum getQuantity() const
```

This method returns the quantity of recipe planned in the time bucket by the planning engine. Note that this quantity is not a quantity of material, but must be understood as a quantity of work expressed in the same unit as the production order batch sizes.

```
public void setFirmBatchSizeMax(IloNum ub)
```

This method sets the batch size upper bound used by the planning engine for execution of the recipe in the corresponding time bucket.

```
public void setFirmBatchSizeMin(IloNum lb)
```

This method sets the batch size lower bound used by the planning engine for execution of the recipe in the corresponding time bucket.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking planned production.

```
public void setPlannedNumberOfBatches(IloInt number)
```

This method sets the number of planned batches of the recipe by the planning engine in the associated time bucket.

```
public void setPlannedTimeMax(IloInt value)
```

This method overrides the end time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedEndTime"` is set to zero.

```
public void setPlannedTimeMin(IloInt value)
```

This method overrides the start time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedStartTime"` is set to zero.

```
public ILOMSDEPRECATED void setQuantity(IloNum qty)
```

This method sets the quantity `qty` of the recipe to run in a specific time window.

```
public void setRecipe(IloMSRecipe recipe)
```

This method sets the `recipe` to produce.

# Class IloMSPlanningEngine

**Definition file:** ilplant/planengine.h
**Library:** plant


The `IloMSPlanningEngine` class is used to solve manufacturing planning problems represented by `IloMSModel` objects.
The Production Planning module of Plant PowerOps solves a planning problem with setups under finite capacity constraints.

**The model and planning problem**

An `IloMSModel` instance typically includes:

- A period of time divided into successive time buckets, possibly of different durations
- A set of recipes enabling the production of materials
- A set of resources required to be in particular setup states by each activity of each recipe
- Initial and maximal stocks for each material
- A set of unplanned demands, with each demand requesting a given quantity of a given material at a given due date
- A set of fixed production orders (with given start and end times for all activities) implementing a recipe
- A set of procurements (with receipt and production times) procuring material in the future (already planned arrival of material) or in the past (dated stock).

The planning problem then consists of deciding how many units of each recipe will be executed during each time bucket, so as to minimize a combination of the following costs:

- Processing costs; several recipes can be used to manufacture the same materials, at different costs
- Resource capacity costs; several levels of capacity may be available at different costs for each resource over each bucket
- Idle costs
- Setup costs
- Inventory costs
- Inventory deficit costs
- Nondelivery costs (also called unsatisfied demand costs)
- Earliness costs
- Tardiness costs

The linear combination of all costs is weighted by optimization criteria contained in the current optimization profile.

While minimizing these costs, an additional solution objective may be maximizing **revenue**.

The hard constraints taken into account are:

- Resource capacity constraints
- Minimal and maximal batch size constraints
- Batch size integrality constraints
- Setup time constraints
- Maximal inventory constraints
- Campaign duration constraint
- Shelf life and maturity constraints

**The planning module solution**

The planning solution is an instance of `IloMSPlanningSolution` composed of `IloMSPlannedProduction` and `IloMSPlannedDelivery` instances. The quantity of a planned production is expressed in the same unit as the batch size of production orders of the corresponding recipe. The planned demand delivery consists of a stock

exit to deliver part or all the amount of a specific demand. This delivery occurs in a time bucket that must be strictly included in a delivery window [delivery start min;delivery end max) of the demand.

**Automatic inference of the planning problem from the description of the scheduling problem**

In PPO there is a single description for the model: the detailed scheduling problem. Although the planning engine uses a very different technology as compared to the scheduling engine (mathematical programming versus constraint programming), there is no need to input the data model twice. For example, a specific mechanism infers the possible routing from the multimode recipes.

When considering the planning problem, just be aware of:

- Defining the time buckets
- Abstracting multiple resources into one super resource
- Defining the level of approximation you need with regard to setups.

**Definition of time buckets**

Defining time buckets is an important task. Remember that the planning engine makes rough approximations at the bucket level, enforcing constraints at time bucket granularity; while the scheduling engine enforces constraints at the time unit granularity. For example: The resource capacity is seen as an energy in the time bucket; the inventory max is enforced at each end of the bucket; the material balance is global to the bucket; the maturity and shelf life constraints are enforced pessimistically with respect to bucket size, and so forth.

Note that if a recipe requires a fixed processing time greater than the bucket size, the inference mechanism will automatically detect it and will enforce the resource capacity constraints on a gliding window of several buckets.

**Resource abstraction**

Multiple resources can be abstracted into one single super resource at the planning level to decrease the complexity of the problem. Only similar resources with the same connectivity can be grouped together.

**Setup approximation**

For each resource and each period of time different setup approximations are available:

- **Per bucket per recipe** means that the fixed capacity requirements including setups will be counted independently for each bucket and each recipe
- **Per bucket per setup feature** means that the fixed capacity requirements including setups will be counted independently for each bucket and each setup feature
- **Cross bucket per recipe** is similar to "Per bucket per recipe" except that the continuation of the same recipe from a bucket to the next will not necessitate redoing the setups in the second bucket
- **Cross bucket per setup feature** is similar to "Per bucket per setup feature" except that the continuation of the same setup features from a bucket to the next will not necessitate redoing the setups in the second bucket.

All these models are approximations of what really happens in the factory. A more precise model would require representation of the sequence of activities that occurs on each resource, that is, to go from a planning to a scheduling model.

**Time offset approximation**

The inference mechanism makes an estimate of when a material is produced or consumed. One can however override that inferred value by defining time offsets in the `IloMSMaterialProduction` instances attached to instances of `IloMSMode`.

**Complexity considerations**

The response time of the planning engine is a function of several characteristics of the problem. First, there is the size of the problem, which is mainly a function of:

- Number of buckets * number of recipes

- Number of buckets * number of resources
- Number of buckets * number of materials

Another complexity issue is the existence of noncontinuous variables, which depends upon:

- The type of setup approximation (adds binary variables)
- The existence of fixed processing times and costs (adds integer variables)
- The use of integral constraints for batch size (adds integer variables)

These variables complicate the problem and may transform it into a complex MILP (Mixed Integer Linear Problem).

**Relaxation window**

There are separate horizons for the planning and scheduling modules. The batching module shares the horizon of the scheduling module. When the scheduling horizon is shorter than the planning horizon, you can relax the integrality constraints after the scheduling and batching horizon by calling `setIntegralityAfterBatchingHorizonConstraintsEnabledForPlanning(false)` on the current `IloMSOptimizationProfile`. This defines a relaxation window between the scheduling and the planning horizons.

**Maturity and shelf life constraints**

Maturity and shelf life constraints are handled pessimistically. For instance if a material shelf life is less than the size of a bucket, it must be consumed in the same bucket as it is produced. Note that if you don't define a waste recipe to throw away obsolete stock, then the obsolete goods will remain in stock until the end of the horizon.

**Earliness and tardiness costs**

A demand may have a due date which defines the optimal time for delivery of demanded material. To define the optimal time, four costs may be defined on an `IloMSDueDate` object:

- **Earliness fixed cost**: a fixed cost to be paid per unit of material if the demand is delivered early. In the planning model, this cost will be incurred if the material is delivered in any bucket that precedes the due date.
- **Earliness variable cost**: a cost per unit of material and per unit of time that the delivery is early. In the planning model, this cost will be multiplied by the difference between the due date and the end time of the bucket in which the material is delivered.
- **Tardiness fixed cost**: a fixed cost to be paid per unit of material if the demand is delivered late. In the planning model, this cost will be incurred if the material is delivered in any bucket that follows the due date.
- **Tardiness variable cost**: a cost per unit of material and per unit of time that the delivery is late. In the planning model, this cost will be multiplied by the difference between the start time of the bucket in which the material is delivered and the due date.

**Mandatory demand and unsatisfied demand**

An infinite nondelivery cost for a demand makes it mandatory. A finite nondelivery cost defines the cost per unit of material not delivered to the customer. If there is neither revenue nor nondelivery cost then the planning engine may decide not to produce anything (except if some safety stock has been defined).

**Advanced use: Formula optimization and blending**

The production planning engine also supports recipes with flexible ratios of ingredients for blending purposes. When using this feature the planning engine may create recipe instances as a side effect on the model. For each flexible recipe, a recipe instance with fixed ratios of ingredients may be created for each time bucket. This features allows formula optimization based on quality of materials. The assumption is that qualities blend linearly. See examples/data/oil/refinery.csv for a blending example.

**See Also:** IloMSModel, IloMSBucket, IloMSMaterial, IloMSDemand, IloMSDueDate, IloMSRecipe, IloMSActivity, IloMSMode, IloMSMaterialProduction, IloMSResource, IloMSProductionOrder, IloMSProcurement, IloMSSetupMatrix, IloMSPlanningSetupModel, IloMSStorageUnit, IloMSQuality, IloMSPlannedDelivery,

| Constructor Summary | |
|---|---|
| public | IloMSPlanningEngine(IloMSModel plant) |

| Method Summary | |
|---|---|
| public IloMSPlanningSolution | getBestSolution() const |
| public void | setCheckForStop(IloMSCheckForStop checkForStop) |
| public void | setTimeLimit(IloInt timeLimit) |
| public IloBool | solve() |
| public void | whenSolution(IloMSSolutionHook whenSolutionHook) |

## Constructors

public **IloMSPlanningEngine**(IloMSModel plant)

This method creates a planning engine for the given `IloMSModel` object.

## Methods

public IloMSPlanningSolution **getBestSolution**() const

After solving, this method returns the best planning solution found by the invoking engine. This solution is not necessarily the optimal one.

public void **setCheckForStop**(IloMSCheckForStop checkForStop)

This method installs a stopping callback on the invoking engine.

public void **setTimeLimit**(IloInt timeLimit)

This method sets the `timeLimit` granted for finding a solution.

public IloBool **solve**()

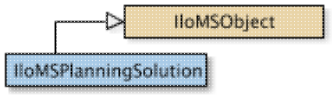
This method solves the planning problem. It returns true if a solution is found; false otherwise.

public void **whenSolution**(IloMSSolutionHook whenSolutionHook)

This method installs a solution callback on the invoking engine.

# Class IloMSPlanningSolution

**Definition file:** ilplant/plansolution.h
**Library:** plant




The `IloMSPlanningSolution` class is used to represent partial and complete planning solutions found by a planning engine.
All methods of the `IloMSPlanningSolution` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public void | display(ostream & stream) const |
| public void | end() |
| public IloNum | getActualCost() const |
| public IloNum | getAmountOfSatisfiedDemand(IloMSDemand demand, IloMSBucket bucket) const |
| public IloNum | getAmountOfSatisfiedDemand(IloMSDemand demand) const |
| public IloMSChecker | getChecker() |
| public IloNum | getInventoryLevel(IloMSMaterial material, IloMSBucket bucket) const |
| public IloNum | getLowerBound() const |
| public IloInt | getNumberOfPlannedDeliveries(const IloMSDemand demand) const |
| public IloInt | getNumberOfPlannedDeliveries() const |
| public IloInt | getNumberOfPlannedProductions() const |
| public IloNum | getObjectiveValue() const |
| public IloMSPlannedDelivery | getPlannedDelivery(const IloMSDemand demand, IloInt i) const |
| public IloMSPlannedDelivery | getPlannedDelivery(IloInt i) const |
| public IloMSPlannedProduction | getPlannedProduction(IloInt i) const |
| public IloNum | getProductionLevel(IloMSMaterial material, IloMSBucket bucket) const |
| public IloNum | getValue(IloMSOptimizationCriterion criterion) const |
| public IloNum | getValueInternal(IloMSOptimizationCriterion criterion) const |
| public IloNum | getWaste(IloMSMaterial material, IloMSBucket bucket) const |
| public IloMSPlannedDelivery | newPlannedDelivery(IloMSDemand demand, IloMSBucket bucket, IloNum quantity) |
| public IloMSPlannedProduction | newPlannedProduction(IloMSRecipe recipe, IloMSBucket bucket, IloNum quantity) |

| Inherited Methods from `IloMSObject` |
| --- |
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

```
public void display(ostream & stream) const
```

This method displays the solution in the stream passed as argument.

```
public void end()
```

This method destroys the planning solution and frees its memory.

```
public IloNum getActualCost() const
```

This method returns the value of the total cost, which is the linear combination of the weighted standard planning KPIs.

```
public IloNum getAmountOfSatisfiedDemand(IloMSDemand demand, IloMSBucket bucket)
const
```

This method returns the amount of the given demand that is satisfied in the specified bucket in the invoking solution.

```
public IloNum getAmountOfSatisfiedDemand(IloMSDemand demand) const
```

This method returns the amount of the given demand that is satisfied in the invoking solution.

```
public IloMSChecker getChecker()
```

This method returns the checker of the invoking solution.

```
public IloNum getInventoryLevel(IloMSMaterial material, IloMSBucket bucket) const
```

This method returns the stock quantity of the specified material at the end of the specified bucket with regard to the invoking planning solution.

```
public IloNum getLowerBound() const
```

This method returns the best lower bound of the objective found by solving the relaxed problem.

```
public IloInt getNumberOfPlannedDeliveries(const IloMSDemand demand) const
```

This method returns the number of planned demand deliveries for the specified demand owned by this planning solution.

```
public IloInt getNumberOfPlannedDeliveries() const
```

This method returns the number of planned demand deliveries owned by this planning solution.

```
public IloInt getNumberOfPlannedProductions() const
```

This method returns the number of planned productions owned by the invoking planning solution.

```
public IloNum getObjectiveValue() const
```

This method returns the value of the objective function found by the planning for this solution.

```
public IloMSPlannedDelivery getPlannedDelivery(const IloMSDemand demand, IloInt i)
const
```

This method returns the planned demand delivery with the specified index `i` for the specified demand in the invoking solution.

```
public IloMSPlannedDelivery getPlannedDelivery(IloInt i) const
```

This method returns the planned demand deliveries with the specified index `i` in the solution.

```
public IloMSPlannedProduction getPlannedProduction(IloInt i) const
```

This method returns the planned production with the specified index `i` in the solution.

```
public IloNum getProductionLevel(IloMSMaterial material, IloMSBucket bucket) const
```

This method returns the quantity of the specified material produced in the specified bucket with regard to the invoking planning solution.

```
public IloNum getValue(IloMSOptimizationCriterion criterion) const
```

This method returns the value of the given `criterion` in the objective function.

```
public IloNum getValueInternal(IloMSOptimizationCriterion criterion) const
```

internal (tests)

```
public IloNum getWaste(IloMSMaterial material, IloMSBucket bucket) const
```

This method returns the amount of material to throw away in the specified bucket.

```
public IloMSPlannedDelivery newPlannedDelivery(IloMSDemand demand, IloMSBucket
bucket, IloNum quantity)
```

This method creates, adds and returns a new demand delivery for the specified demand in the specified bucket.

```
public IloMSPlannedProduction newPlannedProduction(IloMSRecipe recipe, IloMSBucket
bucket, IloNum quantity)
```

This method creates, adds and returns a new planned production in the specified bucket for the specified recipe. The decision about modes (resource alternatives) must be specified by adding references to mode prototypes on the planned production object.

# Class IloMSPrecedence

**Definition file:** ilplant/precedence.h
**Library:** plant


The `IloMSPrecedence` class is used to represent precedence constraints between activities.
Four types of precedence constraints are distinguished: start-to-start precedence constraints relate the start times of the two activities; start-to-end constraints relate the start time of the predecessor with the end time of the successor; end-to-start constraints relate the end time of the predecessor with the start time of the successor; and end-to-end constraints relate the end times of the two activities.

You can define positive minimum (DELAY_MIN) and maximum (DELAY_MAX) delays between the relevant start or end points of the two activities. These delays are specified in TIME_UNITs. The special value -1 indicates that the corresponding delay constraint does not apply. In particular, when the minimum delay constraint does not apply, the relevant time point of the successor is not forced to follow the relevant time point of the predecessor.

All the methods of the `IloMSPrecedence` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSAbstractActivity

| Method Summary | |
|---|---|
| public IloInt | getDelayMax() const |
| public IloInt | getDelayMin() const |
| public IloMSAbstractActivity | getPredecessor() const |
| public IloMSAbstractActivity | getSuccessor() const |
| public IloMSPrecedenceType | getType() const |
| public IloBool | hasDelayMax() const |
| public IloBool | hasDelayMin() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloInt` **`getDelayMax`**`() const`

This method returns the maximal delay between the relevant time points of the two activities.

`public IloInt` **`getDelayMin`**`() const`

This method returns the minimal delay between the relevant time points of the two activities.

```
public IloMSAbstractActivity getPredecessor() const
```

This method returns the predecessor activity of the invoking precedence constraint.

```
public IloMSAbstractActivity getSuccessor() const
```

This method returns the successor activity of the invoking precedence constraint.

```
public IloMSPrecedenceType getType() const
```

This method returns the type of the precedence relation between the two activities.

**See Also:** IloMSPrecedenceType

```
public IloBool hasDelayMax() const
```

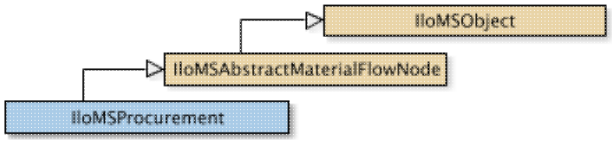
This method returns true if the maximal delay constraint applies, and false otherwise.

```
public IloBool hasDelayMin() const
```

This method returns true if the minimal delay constraint applies, and false otherwise.

# Class IloMSProcurement

**Definition file:** ilplant/procurement.h
**Library:** plant




The class `IloMSProcurement` is used to represent material procured from outside the plant or the initial stock. A procurement, represented by an instance of `IloMSProcurement`, is an amount of material with an age computable according to its production date. A procurement may override the default shelf life and maturity characteristics of the material.

| Method Summary | |
|---|---|
| public IloInt | getConsumptionTimeMax() const |
| public IloInt | getConsumptionTimeMin() const |
| public IloMSMaterial | getMaterial() const |
| public IloInt | getMaturity() const |
| public IloInt | getOverriddenMaturity() const |
| public IloInt | getOverriddenShelfLife() const |
| public IloInt | getProductionTime() const |
| public IloNum | getQuantity() const |
| public IloNum | getQuantityInDisplayUnit() const |
| public IloInt | getReceiptTime() const |
| public IloInt | getShelfLife() const |
| public IloMSStorageUnit | getStorageUnit() const |
| public IloBool | hasOverriddenMaturity() const |
| public IloBool | hasOverriddenShelfLife() const |
| public IloBool | hasStorageUnit() const |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setMaterial(IloMSMaterial material) |
| public void | setOverriddenMaturity(IloInt value) |
| public void | setOverriddenShelfLife(IloInt value) |
| public void | setProductionTime(IloInt value) |
| public void | setQuantity(IloNum quantity) |
| public void | setQuantityInDisplayUnit(IloNum quantityInDisplayUnit) |
| public void | setReceiptTime(IloInt time) |
| public void | setStorageUnit(IloMSStorageUnit storageUnit) |

| Inherited Methods from `IloMSAbstractMaterialFlowNode` |
|---|
| getCategory, getMaterialFlowNodeType, isFirm, setCategory, setFirm, toDemand, toProductionOrder |

| Inherited Methods from `IloMSObject` |
| --- |
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloInt` **`getConsumptionTimeMax`**`() const`

This method returns the latest time at which the material provided by this procurement may be consumed. It is the sum of the production time and the (possibly overridden) shelf life of the material. It can be infinite if no shelf life is defined for the material.

`public IloInt` **`getConsumptionTimeMin`**`() const`

This method returns the earliest time at which the material provided by this procurement may be consumed. It is either the receipt time of the procurement or the sum of the production time and the (possibly overridden) maturation time for the material.

`public IloMSMaterial` **`getMaterial`**`() const`

This method returns the procured material.

`public IloInt` **`getMaturity`**`() const`

This method returns the maturation time (overridden or not) of the material provided by this procurement.

`public IloInt` **`getOverriddenMaturity`**`() const`

This method returns the overridden maturation time of the material provided by this procurement. If the value returned is negative, it means that the maturation time of the material is not overridden.

`public IloInt` **`getOverriddenShelfLife`**`() const`

This method returns the overridden shelf life of the material provided by this procurement. If the value returned is negative, it means that the shelf life of the material is not overridden.

`public IloInt` **`getProductionTime`**`() const`

This method returns the time at which the material provided by this procurement was produced (for age computation).

`public IloNum` **`getQuantity`**`() const`

This method returns the quantity of the procured material expressed in the primary unit of the material.

```
public IloNum getQuantityInDisplayUnit() const
```

This method returns the quantity of the procured material in the material display unit.

```
public IloInt getReceiptTime() const
```

This method returns the receipt time of the procurement.

```
public IloInt getShelfLife() const
```

This method returns the shelf life (overridden or not) of the material provided by this procurement.

```
public IloMSStorageUnit getStorageUnit() const
```

This method returns the storage unit in which the procured material is stored. As the storage unit is optional, a NULL object may be returned.

```
public IloBool hasOverriddenMaturity() const
```

This method returns true if the maturation time of the material provided by this procurement has been overridden.

```
public IloBool hasOverriddenShelfLife() const
```

This method returns true if the shelf life of the material provided by this procurement has been overridden.

```
public IloBool hasStorageUnit() const
```

This method returns true if a storage unit has been specified this procurement.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking procurement. An exception is thrown if the given `identifier` is already used for another procurement.

```
public void setMaterial(IloMSMaterial material)
```

This method sets the procured material.

```
public void setOverriddenMaturity(IloInt value)
```

This method sets the overridden maturation time of the material provided by this procurement.

```
public void setOverriddenShelfLife(IloInt value)
```

This method sets the overridden shelf life of the material provided by this procurement.

```
public void setProductionTime(IloInt value)
```

This method sets the production time of the material provided by this procurement (for age computation).

```
public void setQuantity(IloNum quantity)
```

This method sets the quantity of material provided by this procurement, expressed in the primary unit of the material.

```
public void setQuantityInDisplayUnit(IloNum quantityInDisplayUnit)
```

This method sets the quantity of material provided by this procurement. The provided quantity is expressed in the display unit of the material.

```
public void setReceiptTime(IloInt time)
```

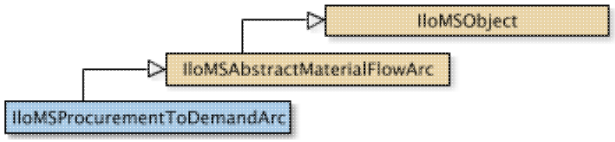
This method sets the receipt time of the procurement.

```
public void setStorageUnit(IloMSStorageUnit storageUnit)
```

This method sets the storage unit in which the procured material is stored.

# Class IloMSProcurementToDemandArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProcurementToDemandArc` class is used to represent flow of material between a procurement and a demand. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between a procurement and demand is modeled as a precedence constraint between the procurement and the demand, and may partially or entirely satisfy the demand.

All the methods of the `IloMSProcurementToDemandArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public IloMSDemand | getDemand() const |
| public IloMSProcurement | getProcurement() const |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloMSDemand getDemand() const
```

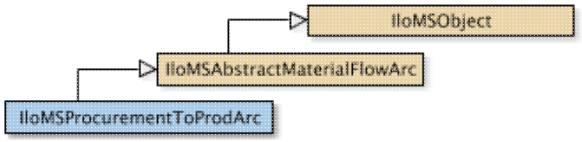
Returns the successor demand in this arc.

```
public IloMSProcurement getProcurement() const
```

Returns the predecessor procurement.

# Class IloMSProcurementToProdArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProcurementToProdArc` class is used to represent flow of material between a procurement and a production order. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between a procurement and a production order is modeled as a precedence constraint between the procurement and the consuming activity of the production order.

All the methods of the `IloMSProcurementToProdArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSProcurement` | `getProcurement() const` |
| `public IloMSProductionOrder` | `getProductionOrder() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSProcurement` **`getProcurement`**`() const`

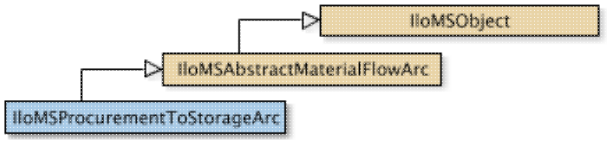
Returns the predecessor procurement.

`public IloMSProductionOrder` **`getProductionOrder`**`() const`

Returns the successor production order.

# Class IloMSProcurementToStorageArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProcurementToStorageArc` class is used to represent flow of material between a procurement and stock or storage. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between a procurement and storage is modeled as a precedence constraint between the procurement and the storage.

All the methods of the `IloMSProcurementToStorageArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSProcurement` | `getProcurement() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

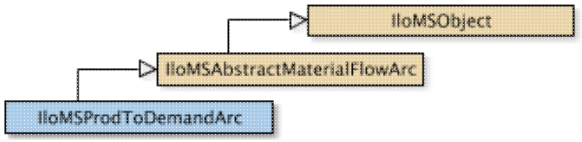
## Methods

`public IloMSProcurement` **`getProcurement`**`() const`

Returns the predecessor procurement.

# Class IloMSProdToDemandArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProdToDemandArc` class is used to represent flow of material between a production order and a demand. Note that a material flow between two nodes represents a precedence constraint between them. A flow of material between production and demand is modeled as a precedence constraint between the producing activity of a production order and a demand, and may partially or entirely satisfy the demand.

All the methods of the `IloMSProdToDemandArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSDemand` | `getDemand() const` |
| `public IloMSProductionOrder` | `getProductionOrder() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSDemand` **`getDemand`**`() const`

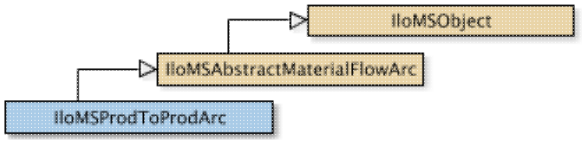
Returns the successor demand.

`public IloMSProductionOrder` **`getProductionOrder`**`() const`

Returns the production order predecessor.

# Class IloMSProdToProdArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProdToProdArc` class is used to represent flow of material between two production orders. Note that a material flow between production orders represents a precedence constraint between them.
A flow of material between two production orders is modeled as a precedence constraint between the producing and consuming activities of the corresponding production orders.

All the methods of the `IloMSProdToProdArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSProductionOrder` | `getPredecessorProductionOrder() const` |
| `public IloMSProductionOrder` | `getSuccessorProductionOrder() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSProductionOrder` **`getPredecessorProductionOrder`**`() const`

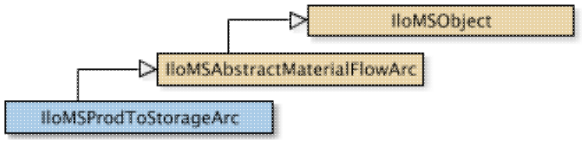
Returns the production order predecessor.

`public IloMSProductionOrder` **`getSuccessorProductionOrder`**`() const`

Returns the production order successor.

# Class IloMSProdToStorageArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSProdToStorageArc` class is used to represent flow of material between a production order and stock or storage. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between a production order and storage is modeled as a precedence constraint between the producing activity of a production order and stock.

All the methods of the `IloMSProdToStorageArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSProductionOrder` | `getProductionOrder() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

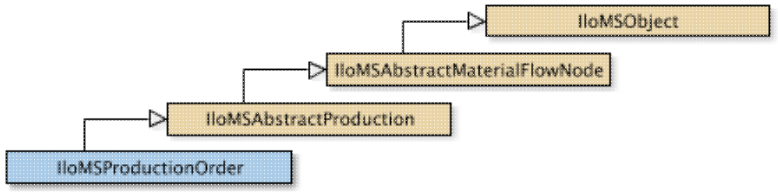
## Methods

`public IloMSProductionOrder` **`getProductionOrder`**`() const`

Returns the predecessor to the production order.

# Class IloMSProductionOrder

**Definition file:** ilplant/order.h
**Library:** plant




The class `IloMSProductionOrder` is used to represent production orders.
An instance of `IloMSProductionOrder` represents a batch using the recipe of a process. This order has a multiplication factor regarding the recipe called the batch size. The batch size is used to compute the quantity of material produced or consumed by the production order. Batch size is also used to adjust the processing time of generated activities (if a variable processing time has been specified on the activity prototypes of the recipe). A production order producing finished goods may partially or wholly satisfy a single demand, or may satisfy several demands. A production order producing only intermediates does not satisfy demand. The material flow between production orders can be modeled by adding material flow arcs between production orders.

**See Also:** IloMSRecipe, IloMSActivity, IloMSMaterial, IloMSAbstractMaterialFlowArc

| Method Summary | |
|---|---|
| public IloMSActivity | getActivity(IloMSActivity prototype) const |
| public IloMSActivity | getActivity(IloInt index) const |
| public IloMSAbstractActivity | getActivityPrototype(IloMSAbstractActivity activity) const |
| public IloMSBatchingSolution | getBatchingSolution() const |
| public const char * | getComment() const |
| public IloInt | getEndMax() const |
| public IloInt | getNumberOfActivities() const |
| public IloInt | getOverriddenExpirationTime() const |
| public IloInt | getPlannedTimeMax() const |
| public IloInt | getPlannedTimeMin() const |
| public IloInt | getStartMin() const |
| public IloBool | hasGenerated(IloMSActivity activity) const |
| public IloBool | isEditable() const |
| public IloBool | isFirmingEditable() const |
| public IloBool | isFixed() const |
| public IloBool | isSatisfying(IloMSDemand demand) const |
| public void | setComment(const char * comment) |
| public void | setEditable(IloBool editable) |
| public void | setEndMax(IloInt value) |
| public void | setFirmingEditable(IloBool firmingEditable) |
| public void | setFixed(IloBool fixed) |
| public void | setIdentifier(IloMSIdentifier identifier) |

| | |
|---|---|
| public void | setOverriddenExpirationTime(IloInt timeUnits) |
| public void | setPlannedTimeMax(IloInt value) |
| public void | setPlannedTimeMin(IloInt value) |
| public void | setStartMin(IloInt value) |

| Inherited Methods from IloMSAbstractProduction |
|---|
| addPlannedMode, getBatchSize, getNumberOfPlannedModes, getPlannedMode, getPlannedTimeMax, getPlannedTimeMin, getRecipe, isFirm, setBatchSize, setFirm, setPlannedTimeMax, setPlannedTimeMin |

| Inherited Methods from IloMSAbstractMaterialFlowNode |
|---|
| getCategory, getMaterialFlowNodeType, isFirm, setCategory, setFirm, toDemand, toProductionOrder |

| Inherited Methods from IloMSObject |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloMSActivity getActivity(IloMSActivity prototype) const
```

This method returns the generated activity corresponding to the `prototype` passed as argument.

```
public IloMSActivity getActivity(IloInt index) const
```

This method returns the generated activity specified by the `index` passed as argument.

```
public IloMSAbstractActivity getActivityPrototype(IloMSAbstractActivity activity)
const
```

This method returns the activity prototype corresponding to the generated `activity` passed as argument.

```
public IloMSBatchingSolution getBatchingSolution() const
```

This method returns the batching solution to which the invoking production order belongs.

```
public const char * getComment() const
```

This accessor returns the comment associated with the invoking production order.

```
public IloInt getEndMax() const
```

This method returns the latest end time of this production order. The time is expressed in the time unit of the model, from the date origin of the model.

```
public IloInt getNumberOfActivities() const
```

This method returns the number of generated activities.

```
public IloInt getOverriddenExpirationTime() const
```

This accessor returns the expiration time of the lot produced by this production order. This time is a function of the shelf life of the products produced by the order and can be overridden manually.

```
public IloInt getPlannedTimeMax() const
```

This method returns the end time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedEndTime"` is set to zero.

```
public IloInt getPlannedTimeMin() const
```

This method returns the start time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting `"slackOnPlannedStartTime"` is set to zero.

```
public IloInt getStartMin() const
```

This method returns the earliest start time of this production order. The time is expressed in the time unit of the model, from the date origin of the model.

```
public IloBool hasGenerated(IloMSActivity activity) const
```

This predicate method returns true if the activity passed as argument has been generated by this production order.

```
public IloBool isEditable() const
```

This method returns true if the production order can be edited.

```
public IloBool isFirmingEditable() const
```

This method returns true if the firming status of this production order can be edited.

```
public IloBool isFixed() const
```

This method returns the status of the invoking object (it is fixed if all the activity start times and modes are fixed).

```
public IloBool isSatisfying(IloMSDemand demand) const
```

This predicate method returns true if the `demand` passed as argument is satisfied partially or totally by this production order.

```
public void setComment(const char * comment)
```

This modifier associates a comment with the invoking production order.

```
public void setEditable(IloBool editable)
```

This method is used to allow or prevent any modification of the order.

```
public void setEndMax(IloInt value)
```

This method sets the latest end time of this production order. None of the activities of this production order will be allowed to end after the given time. The time is expressed in the time unit of the model, from the date origin of the model.

```
public void setFirmingEditable(IloBool firmingEditable)
```

This method is used to allow or prevent any modification of the firming status of the order.

```
public void setFixed(IloBool fixed)
```

This method fixes the start time, end time, performed status, and mode of all the activities belonging to the production order and firms the production order itself.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking production order. An exception is thrown if the given `identifier` is already used.

```
public void setOverriddenExpirationTime(IloInt timeUnits)
```

This modifier overrides the expiration time of the lot produced by this production order.

```
public void setPlannedTimeMax(IloInt value)
```

This method overrides the end time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting "slackOnPlannedEndTime" is set to zero.

```
public void setPlannedTimeMin(IloInt value)
```

This method overrides the start time of the time bucket in which the planning engine decided to plan the production.

This time is used as a hard constraint by the scheduling engine for all the activities of the production order if the setting "slackOnPlannedStartTime" is set to zero.

```
public void setStartMin(IloInt value)
```

This method sets the earliest start time of this production order. None of the activities of this production order will be allowed to start before the given time. The time is expressed in the time unit of the model, from the date origin of the model.

# Class IloMSQuality

**Definition file:** ilplant/quality.h
**Library:** plant


The class `IloMSQuality` is used to represent qualities of materials.
Recipes may linearly blend ingredients of different qualities to determine and constrain the quality of the recipe product.

**See Also:** IloMSRecipe, IloMSMaterial

| Method Summary | |
|---|---|
| `public void` | `setIdentifier(IloMSIdentifier identifier)` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public void` **`setIdentifier`**`(IloMSIdentifier identifier)`

This method associates an `identifier` with the invoking quality. An exception is thrown if the given `identifier` is already used.

# Class IloMSRecipe

**Definition file:** ilplant/recipe.h
**Library:** plant


The class `IloMSRecipe` is used to represent production recipes.
A recipe object models a production process and is composed of activity prototypes. It contains a template of activity prototypes and constraints that can be used as a mold to generate a set of dependent activities. To effectively use a recipe, create an instance of `IloMSProductionOrder` for that recipe, using the method `newProductionOrder`. The real activities are generated by cloning the activity prototypes of the recipe. This generation is done at the latest when calling the `solve` method of `IloMSSchedulingEngine`. It can also be done earlier by explicitly calling `generateActivities` of `IloMSModel`.

The creation of a new recipe starts by using `newRecipe` of `IloMSModel`. A new activity prototype is created using a call to `newActivity` of `IloMSModel`.

A recipe may produce or consume one or several products. For a given `IloMSMaterial`, at most one of the recipe prototype activities can produce or consume it.

A primary product and a primary ingredient may be declared. Also, the batch size of production orders can be expressed in the display unit of the main product.

**See Also:** IloMSProductionOrder, IloMSMaterial, IloMSModel, IloMSSchedulingEngine

| Method Summary | |
|---|---|
| public void | changeMaterial(IloMSMaterial oldMaterial, IloMSMaterial newMaterial, IloMSStorageUnit oldMaterialStorageUnit, IloMSStorageUnit newMaterialStorageUnit) |
| public IloMSRecipe | deepCopy(IloMSIdentifier identifier=0) const |
| public IloMSActivity | getActivityPrototype(IloInt index) const |
| public IloNum | getAllocationWeight() const |
| public IloNum | getBatchSizeMax() const |
| public IloNum | getBatchSizeMin() const |
| public IloMSUnit | getBatchSizeUnit() const |
| public IloInt | getCampaignCycle() const |
| public IloInt | getCampaignMaxNumberOfOrders() const |
| public IloNum | getConsumedFixedQuantity(IloMSMaterial material) const |
| public IloMSMaterial | getConsumedMaterial(IloInt index) const |
| public IloNum | getConsumedQuantity(IloMSMaterial material) const |
| public IloNum | getConsumedVariableQuantity(IloMSMaterial material) const |
| public IloMSAbstractActivity | getConsumingActivityPrototype(IloMSMaterial material) const |
| public IloInt | getEndMax() const |
| public IloInt | getNumberOfActivityPrototypes() const |

| | |
|---|---|
| public IloInt | getNumberOfConsumedMaterials() const |
| public IloInt | getNumberOfProducedMaterials() const |
| public IloInt | getNumberOfProductionOrders() const |
| public IloMSMaterial | getPrimaryIngredient() const |
| public IloMSMaterial | getPrimaryProduct() const |
| public IloNum | getProducedFixedQuantity(IloMSMaterial material) const |
| public IloMSMaterial | getProducedMaterial(IloInt index) const |
| public IloNum | getProducedQuantity(IloMSMaterial material) const |
| public IloNum | getProducedVariableQuantity(IloMSMaterial material) const |
| public IloMSAbstractActivity | getProducingActivityPrototype(IloMSMaterial material) const |
| public IloMSProductionOrder | getProductionOrder(IloInt index) const |
| public IloInt | getPrototypeIndex(IloMSActivity prototype) const |
| public IloMSRecipeType | getRecipeType() const |
| public IloNum | getSplitBatchSizeMin() const |
| public IloInt | getStartMin() const |
| public IloBool | hasCampaignConstraint() const |
| public IloBool | hasIntegerBatchSize() const |
| public IloBool | hasPrimaryIngredient() const |
| public IloBool | hasPrimaryProduct() const |
| public IloBool | isFlexibleInstance() const |
| public IloBool | isRequiring(IloMSResource resource) const |
| public IloBool | isWasteRecipe() const |
| public void | setAllocationWeight(IloNum ratio) |
| public void | setBatchSizeMax(IloNum limit) |
| public void | setBatchSizeMin(IloNum limit) |
| public void | setCampaignCycle(IloInt cycle) |
| public void | setCampaignMaxNumberOfOrders(IloInt maxOrders) |
| public void | setEndMax(IloInt time) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setIntegerBatchSize(IloBool isInteger) |
| public void | setPrimaryIngredient(IloMSMaterial primaryIngredient) |
| public void | setPrimaryProduct(IloMSMaterial primaryProduct) |
| public void | setRecipeType(IloMSRecipeType type) |
| public void | setSplitBatchSizeMin(IloNum limit) |
| public void | setStartMin(IloInt time) |
| public void | splitCrossBucketOrders() |

| **Inherited Methods from `IloMSObject`** |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, |

getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString

## Methods

```
public void changeMaterial(IloMSMaterial oldMaterial, IloMSMaterial newMaterial,
IloMSStorageUnit oldMaterialStorageUnit, IloMSStorageUnit newMaterialStorageUnit)
```

This method changes all occurrences of the couple `(oldMaterial,oldMaterialStorageUnit)` in the invoking recipe to the couple `(newMaterial,newMaterialStorageUnit)`. If `oldMaterialStorageUnit` is null then all occurrences of `oldMaterial` are replaced by `newMaterial`, not considering storage units. If the `newMaterial` is null, then all production or consumption of `oldMaterial` is removed.

```
public IloMSRecipe deepCopy(IloMSIdentifier identifier=0) const
```

This function implements a new recipe with exactly the same structure but different identifiers as the invoking recipe. If a null `identifier` is passed to this method, a UUID will be generated.

```
public IloMSActivity getActivityPrototype(IloInt index) const
```

This method returns a specific activity prototype contained in the recipe.

```
public IloNum getAllocationWeight() const
```

This method returns the allocation weight. When a material can be produced by using alternative recipes, this allocation weight controls distribution of the dependent demand for the semi-finished product across the different routings.

```
public IloNum getBatchSizeMax() const
```

This method returns the maximal size of production orders implementing this recipe.

```
public IloNum getBatchSizeMin() const
```

This method returns the minimal size of production orders implementing this recipe.

```
public IloMSUnit getBatchSizeUnit() const
```

This method returns the unit of measurement for the primary product per one unit of batch size in the invoking recipe. A null pointer may be returned if it is not possible to determine the correspondence between a unit of measure and a unit of recipe.

```
public IloInt getCampaignCycle() const
```

This method returns the campaign cycle time. This value is used in conjunction with the "campaignMaxNumberOfOrders" field to set an upper bound on the number of produced orders per time interval.

```
public IloInt getCampaignMaxNumberOfOrders() const
```

This method returns the maximum number of produced orders per cycle time. This value is used in conjunction with the "campaignCycle" field to set an upper bound on the number of produced orders per time interval.

```
public IloNum getConsumedFixedQuantity(IloMSMaterial material) const
```

This method returns the quantity of material consumed by one unit of recipe execution.

```
public IloMSMaterial getConsumedMaterial(IloInt index) const
```

This method returns the material consumed by the invoking recipe.

```
public IloNum getConsumedQuantity(IloMSMaterial material) const
```

This method returns the quantity of material consumed by one unit of recipe execution.

```
public IloNum getConsumedVariableQuantity(IloMSMaterial material) const
```

This method returns the quantity of material consumed by one unit of recipe execution.

```
public IloMSAbstractActivity getConsumingActivityPrototype(IloMSMaterial material)
const
```

This method returns the activity prototype of the invoking recipe that consumes the material passed as parameter.

```
public IloInt getEndMax() const
```

This method returns the valid end time of the recipe. The recipe cannot be used after the returned time.

```
public IloInt getNumberOfActivityPrototypes() const
```

This method returns the number of activity prototypes contained in the recipe.

```
public IloInt getNumberOfConsumedMaterials() const
```

This method returns the number of materials consumed by the invoking recipe.

```
public IloInt getNumberOfProducedMaterials() const
```

This method returns the number of materials produced by the invoking recipe.

```
public IloInt getNumberOfProductionOrders() const
```

This method returns the number of production orders of the invoking recipe.

```
public IloMSMaterial getPrimaryIngredient() const
```

This method returns the primary ingredient of this recipe.

```
public IloMSMaterial getPrimaryProduct() const
```

This method returns the primary product of this recipe.

```
public IloNum getProducedFixedQuantity(IloMSMaterial material) const
```

This method returns the quantity of `material` produced by one unit of recipe execution.

```
public IloMSMaterial getProducedMaterial(IloInt index) const
```

This method returns the material produced by the invoking recipe.

```
public IloNum getProducedQuantity(IloMSMaterial material) const
```

This method returns the quantity of `material` produced by one unit of recipe execution.

```
public IloNum getProducedVariableQuantity(IloMSMaterial material) const
```

This method returns the quantity of `material` produced by one unit of recipe execution.

```
public IloMSAbstractActivity getProducingActivityPrototype(IloMSMaterial material)
const
```

This method returns the activity prototype of the invoking recipe that produces the material passed as parameter.

```
public IloMSProductionOrder getProductionOrder(IloInt index) const
```

This method returns the production order of the invoking recipe with specified index.

```
public IloInt getPrototypeIndex(IloMSActivity prototype) const
```

This method returns the index of the corresponding production activity prototype within the invoking recipe, or throws an exception if the prototype is not found.

```
public IloMSRecipeType getRecipeType() const
```

This method returns the type of the invoking recipe.

**See Also:** IloMSRecipeType

```
public IloNum getSplitBatchSizeMin() const
```

This method returns the minimal possible size of production orders when splitting cross-bucket orders. If not defined then the minimal batch size (`setBatchSizeMin`) is used.

```
public IloInt getStartMin() const
```

This method returns the valid start time of the recipe. The recipe cannot be used before the returned time.

```
public IloBool hasCampaignConstraint() const
```

This method returns true if the recipe has a campaign constraint.

```
public IloBool hasIntegerBatchSize() const
```

This method returns true if any production orders created from this recipe must take an integer batch size.

```
public IloBool hasPrimaryIngredient() const
```

This method returns true if a primary ingredient has been declared for this recipe.

```
public IloBool hasPrimaryProduct() const
```

This method returns true if a primary product has been declared for this recipe.

```
public IloBool isFlexibleInstance() const
```

This method returns true if the invoking recipe comes from the instantiation of a flexible recipe after solving a blending problem with the planning module.

```
public IloBool isRequiring(IloMSResource resource) const
```

This method returns true if the recipe requires the resource `resource`.

```
public IloBool isWasteRecipe() const
```

This method returns true if the invoking recipe consumes one material and produces nothing.

```
public void setAllocationWeight(IloNum ratio)
```

This method sets the allocation weight. When a material can be produced by using alternative recipes, this allocation weight controls distribution of the dependent demand for the semi-finished product across the different routings.

```
public void setBatchSizeMax(IloNum limit)
```

This method sets the maximal size of production orders implementing this recipe.

```
public void setBatchSizeMin(IloNum limit)
```

This method sets the minimal size of production orders implementing this recipe.

```
public void setCampaignCycle(IloInt cycle)
```

This method sets the campaign cycle time of the recipe.

```
public void setCampaignMaxNumberOfOrders(IloInt maxOrders)
```

This method sets the campaign maximum number of orders per cycle time of the recipe.

```
public void setEndMax(IloInt time)
```

This method sets the valid end time of the recipe. The recipe cannot be used after the time `time`.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking recipe. An exception is thrown if the given `identifier` is already used.

```
public void setIntegerBatchSize(IloBool isInteger)
```

This method states that any production orders created from this recipe must take an integer batch size.

```
public void setPrimaryIngredient(IloMSMaterial primaryIngredient)
```

This method sets the primary ingredient of this recipe.

```
public void setPrimaryProduct(IloMSMaterial primaryProduct)
```

This method sets the primary product of this recipe.

```
public void setRecipeType(IloMSRecipeType type)
```

This method specifies the `type` of the invoking recipe.

**See Also:** IloMSRecipeType

```
public void setSplitBatchSizeMin(IloNum limit)
```

This method sets the minimal possible size of production orders when splitting cross-bucket orders using the splitCrossBucketOrders API. If not defined then the minimal batch size (`setBatchSizeMin`) is used.

```
public void setStartMin(IloInt time)
```

This method sets the valid start time of the recipe. The recipe cannot be used before the time `time`.

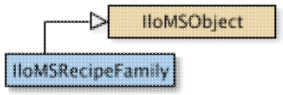
```
public void splitCrossBucketOrders()
```

This method splits all production orders of the invoking recipe that cross the boundaries of the buckets in the current solution.

Note that this is possible on recipes having a single activity prototype and for which the processing time is purely variable.

# Class IloMSRecipeFamily

**Definition file:** ilplant/recipefamily.h
**Library:** plant




The `IloMSRecipeFamily` class is used to represent recipe families. A recipe may be a member of several families. Families are a way to define submodels and to perform a partial database load of the data.

**See Also:** IloMSRecipe, IloMSRecipeFamilyFilter, IloMSScope

| Method Summary | |
|---|---|
| public void | add(IloMSRecipe recipe) |
| public void | display(ostream & stream) const |
| public IloInt | getNumberOfRecipes() const |
| public IloMSRecipe | getRecipe(IloInt index) const |
| public IloMSIdentifier | getType() const |
| public IloBool | isMember(IloMSRecipe recipe) const |
| public void | remove(IloMSRecipe recipe) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setType(IloMSIdentifier type) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`add`**`(IloMSRecipe recipe)`

This method adds the recipe to the invoking family.

`public void` **`display`**`(ostream & stream) const`

This method displays the recipe family in the stream passed as argument.

`public IloInt` **`getNumberOfRecipes`**`() const`

This method returns the number of recipes in this family.

`public IloMSRecipe` **`getRecipe`**`(IloInt index) const`

This method returns the recipe with the `index` in the invoking family.

```
public IloMSIdentifier getType() const
```

This method retrieves the type of the family.

```
public IloBool isMember(IloMSRecipe recipe) const
```

This method return true if the recipe is a member of the invoking family.

```
public void remove(IloMSRecipe recipe)
```

This method removes the recipe from the invoking family.

```
public void setIdentifier(IloMSIdentifier identifier)
```

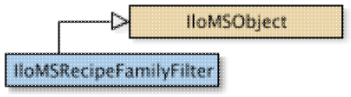
This modifier associates an `identifier` with the invoking recipe family. An exception is thrown if the given `identifier` is already used.

```
public void setType(IloMSIdentifier type)
```

This method registers the invoking family in the collection of families of the same type.

# Class IloMSRecipeFamilyFilter

**Definition file:** ilplant/recipefamilyfilter.h
**Library:** plant




The `IloMSRecipeFamilyFilter` class is used to filter a model to create a submodel that contains recipe families.
The submodel contains only the recipe families as defined in the filter. Two different sets of recipe families are created: A first set contains "frozen" recipe families, with the recipes and corresponding orders of this set fixed. A second set contains "planned" recipe families, with the recipes and corresponding orders that are planned and scheduled.

The scope permits creation of a submodel with this filter applied on the core model.

**See Also:** IloMSScope, IloMSRecipeFamily

| Method Summary | |
|---|---|
| public void | addFrozenRecipeFamily(IloMSRecipeFamily recipeFamily) |
| public void | addPlannedRecipeFamily(IloMSRecipeFamily recipeFamily) |
| public void | addRecipeFamily(IloMSRecipeFamily recipeFamily, IloMSRecipeFamilyStatus recipeFamilyStatus) |
| public void | display(ostream & stream) const |
| public IloMSRecipeFamily | getFrozenRecipeFamily(IloInt i) const |
| public IloInt | getNumberOfFrozenRecipeFamilies() const |
| public IloInt | getNumberOfPlannedRecipeFamilies() const |
| public IloMSRecipeFamily | getPlannedRecipeFamily(IloInt i) const |
| public IloMSScope | getScope() |
| public IloBool | isFrozenMember(IloMSRecipeFamily recipeFamily) |
| public IloBool | isMember(IloMSRecipeFamily recipeFamily) |
| public IloBool | isPlannedMember(IloMSRecipeFamily recipeFamily) |
| public void | removeFrozenRecipeFamily(IloMSRecipeFamily recipeFamily) |
| public void | removePlannedRecipeFamily(IloMSRecipeFamily recipeFamily) |
| public void | removeRecipeFamily(IloMSRecipeFamily recipeFamily, IloMSRecipeFamilyStatus recipeFamilyStatus) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setScope(IloMSScope scope) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public void addFrozenRecipeFamily(IloMSRecipeFamily recipeFamily)
```

This method adds a frozen recipe family from this filter.

To the associated filter, it adds the given recipe family from the list of frozen recipe families.

```
public void addPlannedRecipeFamily(IloMSRecipeFamily recipeFamily)
```

This method adds a planned recipe family from this filter.

To the associated filter, it adds a given recipe family from the list of planned recipe families.

```
public void addRecipeFamily(IloMSRecipeFamily recipeFamily, IloMSRecipeFamilyStatus
recipeFamilyStatus)
```

This method adds a recipe family from this filter, given the recipe family status.

To the associated filter, it adds the given recipe family either from the list of planned recipe families or from the list of frozen recipe families.

```
public void display(ostream & stream) const
```

This method displays the planned production in the `stream` passed as argument.

```
public IloMSRecipeFamily getFrozenRecipeFamily(IloInt i) const
```

This method returns the frozen recipe family at index `i`.

```
public IloInt getNumberOfFrozenRecipeFamilies() const
```

This method returns the number of frozen recipe families in this filter.

In the associated submodel, this is the number of recipe families for which recipes and corresponding orders will be fixed.

```
public IloInt getNumberOfPlannedRecipeFamilies() const
```

This method returns the number of planned recipe families in this filter.

In the associated submodel, this is the number of recipe families for which recipes and corresponding orders will be planned and scheduled.

```
public IloMSRecipeFamily getPlannedRecipeFamily(IloInt i) const
```

This method returns the planned recipe family at index `i`.

```
public IloMSScope getScope()
```

This method returns the scope to which the filter is associated.

The scope permits creation of a submodel with this filter applied on the core model.

```
public IloBool isFrozenMember(IloMSRecipeFamily recipeFamily)
```

This method returns true if the given recipe family belongs to this filter.

Only the set of frozen recipe families is searched.

```
public IloBool isMember(IloMSRecipeFamily recipeFamily)
```

This method returns true if the given recipe family belongs to this filter.

Both the set of frozen recipe families and the set of planned recipe families are searched.

```
public IloBool isPlannedMember(IloMSRecipeFamily recipeFamily)
```

This method returns true if the given recipe family belongs to this filter.

Only the set of planned recipe families is searched.

```
public void removeFrozenRecipeFamily(IloMSRecipeFamily recipeFamily)
```

This method removes a frozen recipe family from this filter.

In the associated filter, it removes a given recipe family from the list of frozen recipe families.

```
public void removePlannedRecipeFamily(IloMSRecipeFamily recipeFamily)
```

This method removes a planned recipe family from this filter.

In the associated filter, it removes a given recipe family from the list of planned recipe families.

```
public void removeRecipeFamily(IloMSRecipeFamily recipeFamily,
IloMSRecipeFamilyStatus recipeFamilyStatus)
```

This method removes a recipe family from this filter, given the recipe family status.

In the associated filter, it removes the given recipe family either from the list of planned recipe families or from the list of frozen recipe families.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking recipe family filter.

```
public void setScope(IloMSScope scope)
```

This method sets the scope to which the filter is applied.

The scope permits creation of a submodel with this filter applied on the core model.

# Class IloMSRepairAlgorithm

**Definition file:** ilplant/repair.h
**Library:** plant


The `IloMSRepairAlgorithm` class is used to change a scheduling solution while enforcing (or repairing) some constraints.
`IloMSRepairAlgorithm` is deprecated. Use Java(TM) class `IloMSCommandProcessorRepairCapacity`.

The `IloMSRepairAlgorithm` can be used either to 'repair' a solution in which some constraints are violated, or to change some activities in the solution while maintaining or repairing the constraints associated with those activities. The possibilities include: Changing the date of an activity, changing the activity mode, changing the activity primary resource, or changing the batch size of the associated production order.

Depending on the values for the 'scope' and the 'capacity' parameters, the repair algorithm will consider more or fewer constraints, for more or fewer activities.

The 'scope' parameter determines the set of activities that will have their constraints enforced. The 'capacity' parameter determines whether the capacity of the primary resource of the selected activities (it must be the same for all) is considered.

The usage pattern for `IloMSRepairAlgorithm` is the following: Get the instance using `IloMSModel::getRepairAlgorithm()`, then add one or more activities using `addSelectedActivity()`, possibly set a non-null delta using `setDelta()`, and call zero or one of `setNewMode()`, `setNewBatchSize()` or `setNewResource()`, then call `repair()`.

**See Also:** IloMSRepairCapacity, IloMSRepairExtent, IloMSModel

| Method Summary | |
|---|---|
| public void | addSelectedActivity(IloMSScheduledActivity pivot) |
| public IloBool | repair() |
| public void | setDelta(IloInt delta) |
| public void | setNewBatchSize(IloNum newBatchSize) |
| public void | setNewMode(IloMSMode newMode) |
| public void | setNewResource(IloMSResource newRes) |
| public void | setRepairCapacity(IloMSRepairCapacity repairCapacity) |
| public void | setRepairExtent(IloMSRepairExtent repairExtent) |

## Methods

`public void` **`addSelectedActivity`**`(IloMSScheduledActivity pivot)`

This method adds an activity to the set of activities that will be changed.

`public IloBool` **`repair`**`()`

This method changes the selected activities and repairs the constraints.

The selected activities are first changed as specified using `setDelta(int)`, `setNewResource(IloMSResource)`, `setNewMode(IloMSMode)` or `setNewBatchSize(double)`.

Then the constraints, as defined by the `IloMSRepairExtent` and `IloMSRepairCapacity` parameters, are enforced.

The method returns `true` if the repair algorithm succeeded in finding a new solution, and false otherwise.

```
public void setDelta(IloInt delta)
```

This method sets the amount of time units by which the selected activities must be shifted (to the future if > 0, to the past if < 0).

If the given value is not null, then the start time of the selected activities will be moved by at least the given value. Moreover, any other activity will be constrained to start at or after its original start time (if delta > 0) or to end at or before its original end time (if delta < 0).

```
public void setNewBatchSize(IloNum newBatchSize)
```

This method sets the new batch size of the production order of the selected activity.

This method can be called only for one selected activity at a time. This method can not be called together with `setNewResource(IloMSResource)` or `setNewMode(IloMSMode)`.

```
public void setNewMode(IloMSMode newMode)
```

This method sets the new mode of the selected activity.

This method can be called only for one selected activity at a time. This method can not be called together with `setNewResource(IloMSResource)` or `setNewBatchSize(double)`.

```
public void setNewResource(IloMSResource newRes)
```

This method sets the new primary resource used by the selected activities.

This method can be called only if all of the selected activities currently use the same primary resource. This method can not be called together with `setNewMode(IloMSMode)` or `setNewBatchSize(double)`.

```
public void setRepairCapacity(IloMSRepairCapacity repairCapacity)
```

This method defines how capacity constraints will be repaired.

If the value is `IloMSRepairNoResource`, capacity constraints are ignored.

If the value is `IloMSRepairOneResource`, then the capacity constraints of the primary resource of the selected activities will be enforced; activities that overlap will be changed to eliminate this overlap. Note that the selected activities should all have the same primary resource, and this resource should have a unitary capacity.

**See Also:** IloMSRepairCapacity

```
public void setRepairExtent(IloMSRepairExtent repairExtent)
```

This method defines the set of activities whose constraints will be repaired.

If the value is `IloMSRepairActivity`, then only the activities that are actually selected (e.g., in the Gantt chart) will have their constraints (duration, calendar) enforced. In particular, precedence constraints will not be considered.

If the value is `IloMSRepairProductionOrder`, then all the activities that belong to the production order(s) of the selected activities will have their constraints enforced. In particular, this means that precedence constraints between those activities will be enforced.

If the value is `IloMSRepairCluster`, then all the activities that belong to clusters of the production order(s) of the selected activities will have their constraints enforced. In particular, this means that pegging constraints between those production orders will be enforced.

**See Also:** IloMSRepairExtent

# Class IloMSReplicateAlgorithm

**Definition file:** ilplant/replicate.h
**Library:** plant


The `IloMSReplicateAlgorithm` class is used to duplicate a set of production orders.
Use IloMSReplicateCommand instead.

The replicated set of production orders is moved away from the original orders depending upon the value of the **offset**. If the time duration of the offset is positive, then the replicated production orders are moved that amount of time to the future. If the offset is negative, the orders are moved by that duration to the past.

**See Also:** IloMSProductionOrder, IloMSScheduledActivity, IloMSProdToProdArc

| Method Summary | |
|---|---|
| public void | addActivityToReplicate(IloMSScheduledActivity act) |
| public void | computeModifiedSet() |
| public IloInt | getNumberOfPO() |
| public IloInt | getNumberOfReplicatedActivities() |
| public IloInt | getNumberOfReplicatedArcs() |
| public IloInt | getNumberOfReplicatedPO() |
| public IloInt | getOffset() |
| public IloMSScheduledActivityI * | getReplicatedActivity(IloInt i) |
| public IloMSProdToProdArcI * | getReplicatedArc(IloInt i) |
| public IloMSProductionOrderI * | getReplicatedPO(IloInt i) |
| public void | initialize() |
| public IloBool | replicate(IloBool replicateArcs=IloTrue) |
| public void | reset() |
| public void | setOffset(IloInt offset) |

## Methods

`public void` **`addActivityToReplicate`**`(IloMSScheduledActivity act)`

This method adds an activity to the set of duplicated activities.

`public void` **`computeModifiedSet`**`()`

This method computes the modified set of duplicated production orders.

`public IloInt` **`getNumberOfPO`**`()`

This method returns the number of production orders that are selected for duplication.

```
public IloInt getNumberOfReplicatedActivities()
```

This method returns the number of activities corresponding to the replicated production orders.

```
public IloInt getNumberOfReplicatedArcs()
```

This method returns the number of arcs that are replicated during the duplication of production orders.

```
public IloInt getNumberOfReplicatedPO()
```

This method returns the number of duplicated production orders.

```
public IloInt getOffset()
```

This method returns the offset between the original set of production orders and the duplicated set of production orders.

```
public IloMSScheduledActivityI * getReplicatedActivity(IloInt i)
```

This method returns the replicated activity.

```
public IloMSProdToProdArcI * getReplicatedArc(IloInt i)
```

This method returns the replicated arc.

```
public IloMSProductionOrderI * getReplicatedPO(IloInt i)
```

This method returns the replicated production order.

```
public void initialize()
```

This method initializes variables that are used in the algorithm used to replicate a set of production orders.

```
public IloBool replicate(IloBool replicateArcs=IloTrue)
```

This method is the core function of the replicate algorithm; it creates the duplicated production orders that are delayed by the offset.

```
public void reset()
```

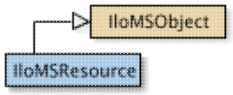
This method resets the algorithm used to replicate a set of production orders.

```
public void setOffset(IloInt offset)
```

This method sets the offset. The offset is the time duration between the original set of production orders and the duplicated set of production orders.

# Class IloMSResource

**Definition file:** ilplant/resource.h
**Library:** plant

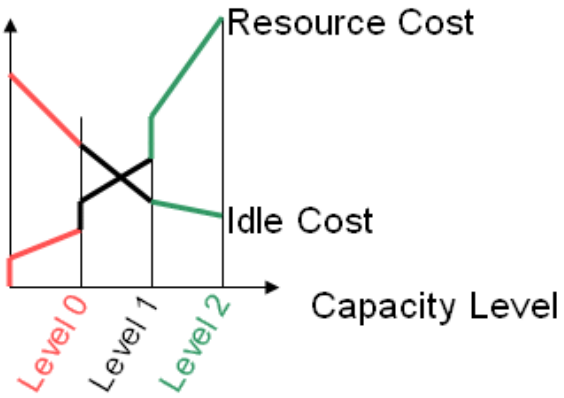



The `IloMSResource` class is used to describe production resources on which activities will be performed. A resource is typically used to model a machine, a tool, a vehicle or equipment. It can also be used to model workers. Do not use `IloMSResource` to model materials being consumed or produced; use `IloMSMaterial` for that purpose.

Resources can be aggregated together into larger groups; one reason to do so is to make solving the planning problem easier. Use of super resources allows the planning engine to consider within each time bucket the grouped capacity of all the resources of a super resource. Note that super resources must group only similar resources, which typically have identical connectivity and make the same products. To define super resources in order to lighten the planning problem, use methods such as `IloMSResource::setSuperResource`.

You can also group resources into resource families. This allows you to tailor and organize resource data for reporting or visibility purposes (for example, to improve visibility on the Gantt Diagram). To create resource families, use the class `IloMSResourceFamily`.

You can define resource usage and idle costs. The cost for using a resource often increases with the capacity used; this resource capacity cost can be defined as a piecewise linear function or as a stepwise linear function. Idle cost can be defined as a piecewise linear function.




You can use `IloMSResourceCapacityCostFunction` to create a penalty for idle resources.

All the methods of the `IloMSResource` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSMaterial, IloMSResourceCapacityCostFunction, IloMSResourceFamily

| Method Summary | |
|---|---|
| `public void` | `addConnectedResource(IloMSResource otherResource)` |
| `public IloMSCalendar` | `getCalendar() const` |
| `public IloNum` | `getCapacity(IloInt time, IloInt levelNumber) const` |

| | |
|---|---|
| public IloInt | getCapacity() const |
| public IloMSResourceCapacityCostFunction | getCapacityCostFunction(IloInt i) const |
| public IloMSResourceCapacityCostFunction | getCapacityCostFunctionAtTime(IloInt time) const |
| public IloInt | getCapacityCostFunctionValidityEnd(IloInt i) const |
| public IloInt | getCapacityCostFunctionValidityStart(IloInt i) const |
| public const char * | getCategory() const |
| public IloNum | getCleanupCost() const |
| public IloMSRecipe | getCleanupRecipe() const |
| public IloInt | getCleanupTime() const |
| public IloMSResource | getConnectedResource(IloInt index) const |
| public IloInt | getDisplayRank() const |
| public IloInt | getEndMax() const |
| public IloMSIdentifier | getFinalSetupState(IloMSIdentifier feature) const |
| public IloNum | getFixedCostOfCapacity(IloInt time, IloInt levelNumber) const |
| public IloNum | getIdleVariableCostOfCapacity(IloInt time, IloInt levelNumber) const |
| public IloMSIdentifier | getInitialSetupState(IloMSIdentifier feature) const |
| public IloMSIdentifier | getLineId() const |
| public IloInt | getMaxIdleTimeBeforeCleanup() const |
| public IloInt | getMaxNumberOfBatchesBeforeCleanup() const |
| public IloInt | getMaxTimeBeforeCleanup() const |
| public IloInt | getNumberOfBatchesSinceLastCleanup() const |
| public IloInt | getNumberOfCapacityCostFunctions() const |
| public IloInt | getNumberOfConnectedResources() const |
| public IloInt | getNumberOfLevelsOfCapacity(IloInt time) const |
| public IloInt | getNumberOfPlanningSetupModels() const |
| public IloInt | getNumberOfSubResources() const |
| public IloNum | getPlanningCapacityReductionFactor() const |
| public IloMSPlanningSetupModel | getPlanningSetupModel(IloInt i) const |
| public IloMSPlanningSetupModel | getPlanningSetupModelAtTime(IloInt time) const |
| public IloInt | getPlanningSetupModelValidityEnd(IloInt i) const |
| public IloInt | getPlanningSetupModelValidityStart(IloInt i) const |
| public IloInt | getRank() const |

| | |
|---|---|
| public IloMSSetupMatrix | getSetupMatrix(IloMSIdentifier feature) const |
| public IloInt | getStartMin() const |
| public IloMSStorageUnit | getStorageUnit() const |
| public IloMSResource | getSubResource(IloInt index) const |
| public IloMSResource | getSuperResource() const |
| public IloInt | getTimeFence() const |
| public IloInt | getTimeOfLastCleanup() const |
| public IloNum | getVariableCostOfCapacity(IloInt time, IloInt levelNumber) const |
| public IloNum | getX() const |
| public IloNum | getY() const |
| public IloBool | hasCalendar() const |
| public IloBool | hasCapacityCostFunctionAtTime(IloInt time) const |
| public IloBool | hasCleanupRecipe() const |
| public IloBool | hasFinalSetupState(IloMSIdentifier feature) const |
| public IloBool | hasInitialSetupState(IloMSIdentifier feature) const |
| public IloBool | hasSetupMatrix(IloMSIdentifier feature) const |
| public IloBool | hasStorageUnit() const |
| public IloBool | hasSuperResource() const |
| public IloBool | isConnectedWith(IloMSResource otherResource) const |
| public IloBool | isTimeFenceDefined() const |
| public void | removeCapacityCostFunction(IloInt i) |
| public void | removeCapacityCostFunction(IloMSResourceCapacityCostFun function, IloInt start, IloInt end) |
| public void | removeConnectedResource(IloMSResource otherResource) |
| public void | removePlanningSetupModel(IloInt i) |
| public void | removePlanningSetupModel(IloMSPlanningSetupModel model, IloInt start, IloInt end) |
| public void | setCalendar(IloMSCalendar calendar) |
| public void | setCapacityCostFunction(IloMSResourceCapacityCostFuncti function, IloInt validityStart, IloInt validityEnd) |
| public void | setCategory(const char * category) |
| public void | setCleanupRecipe(IloMSRecipe recipe) |
| public void | setDisplayRank(IloInt rank) |
| public void | setEndMax(IloInt endMax) |
| public void | setFinalSetupState(IloMSIdentifier feature, IloMSIdentifier initialSetupState) |

| | |
|---|---|
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInitialSetupState(IloMSIdentifier initialSetupState) |
| public void | setInitialSetupState(IloMSIdentifier feature, IloMSIdentifier initialSetupState) |
| public void | setLineId(IloMSIdentifier id) |
| public void | setMaxIdleTimeBeforeCleanup(IloInt value) |
| public void | setMaxNumberOfBatchesBeforeCleanup(IloInt value) |
| public void | setMaxTimeBeforeCleanup(IloInt value) |
| public void | setNumberOfBatchesSinceLastCleanup(IloInt numberOfBatches) |
| public void | setPlanningCapacityReductionFactor(IloNum factor) |
| public void | setPlanningSetupModel(IloMSPlanningSetupModel model, IloInt validityStart, IloInt validityEnd) |
| public void | setRank(IloInt rank) |
| public void | setSetupMatrix(IloMSSetupMatrix setupMatrix) |
| public void | setSetupMatrix(IloMSIdentifier feature, IloMSSetupMatrix setupMatrix) |
| public void | setStartMin(IloInt startMin) |
| public void | setSuperResource(IloMSResource superResource) |
| public void | setTimeFence(IloInt timeFence) |
| public void | setTimeOfLastCleanup(IloInt value) |
| public void | setX(IloNum x) |
| public void | setY(IloNum y) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public void **addConnectedResource**(IloMSResource otherResource)

This method creates a connection between the invoking resource and the resource specified as parameter. A connection between resources allows you to enforce the usage of two connected resources as the primary resources of two activities. There is no need to define the resource connections when all activities can use any resource as the primary resource.

public IloMSCalendar **getCalendar**() const

This method returns the calendar associated with the invoking resource. An exception is thrown if the resource has no calendar.

```
public IloNum getCapacity(IloInt time, IloInt levelNumber) const
```

This method returns the capacity defined at level `levelNumber` of the capacity cost function valid at the given time for the invoking resource. This feature is only used by the planning engine.

```
public IloInt getCapacity() const
```

This method returns the maximal instantaneous capacity of the invoking resource.

```
public IloMSResourceCapacityCostFunction getCapacityCostFunction(IloInt i) const
```

This method returns the capacity cost functions with the given index `i` of the invoking resource. One can use several capacity levels and associate increasing costs of resource usage. This feature is only used by the planning engine.

```
public IloMSResourceCapacityCostFunction getCapacityCostFunctionAtTime(IloInt time)
const
```

This method returns the capacity cost functions valid at `time` for the invoking resource. This feature is only used by the planning engine.

```
public IloInt getCapacityCostFunctionValidityEnd(IloInt i) const
```

This method returns the end time of the validity interval for the capacity cost functions with the given index `i` in the invoking resource. One can use several capacity levels and associate increasing costs of resource usage. This feature is only used by the planning engine.

```
public IloInt getCapacityCostFunctionValidityStart(IloInt i) const
```

This method returns the start time of the validity interval for the capacity cost functions with the given index `i` in the invoking resource. One can use several capacity levels and associate increasing costs of resource usage. This feature is only used by the planning engine.

```
public const char * getCategory() const
```

This method returns the category of this resource to be interpreted for symbolic representation in a specialized editor.

```
public IloNum getCleanupCost() const
```

This method returns the cost of a cleaning activity.

```
public IloMSRecipe getCleanupRecipe() const
```

This method returns the cleanup recipe associated with the resource that allows creating cleanup orders when required.

```
public IloInt getCleanupTime() const
```

This method returns the processing time of a cleaning activity.

```
public IloMSResource getConnectedResource(IloInt index) const
```

This method returns the connected resource with the given `index` for the invoking resource.

```
public IloInt getDisplayRank() const
```

This method returns the rank of the resource, as displayed in the **Gantt Diagram**.

```
public IloInt getEndMax() const
```

This method returns the time after which no production activity or setup can end on the resource.

It is the common latest end time shared by all modes requiring the resource.

```
public IloMSIdentifier getFinalSetupState(IloMSIdentifier feature) const
```

This method returns the required final setup state of the invoking resource for the given setup feature. An exception is thrown if the invoking resource has no initial setup.

```
public IloNum getFixedCostOfCapacity(IloInt time, IloInt levelNumber) const
```

This method returns the fixed cost defined at level `levelNumber` of the capacity cost function valid at the given time for the invoking resource. This feature is only used by the planning engine.

```
public IloNum getIdleVariableCostOfCapacity(IloInt time, IloInt levelNumber) const
```

This method returns the idle variable cost defined at level `levelNumber` of the capacity cost function valid at the given time for the invoking resource. This feature is only used by the planning engine.

```
public IloMSIdentifier getInitialSetupState(IloMSIdentifier feature) const
```

This method returns the initial setup state of the invoking resource for the given setup feature. An exception is thrown if the invoking resource has no initial setup.

```
public IloMSIdentifier getLineId() const
```

This method returns the line identifier of the invoking resource.

```
public IloInt getMaxIdleTimeBeforeCleanup() const
```

This method returns the maximum number of time units between two cleanups if the resource is idle.

```
public IloInt getMaxNumberOfBatchesBeforeCleanup() const
```

This method returns the maximum number of batches executable between two cleanups.

```
public IloInt getMaxTimeBeforeCleanup() const
```

This method returns the maximum number of time units between two cleanups.

```
public IloInt getNumberOfBatchesSinceLastCleanup() const
```

This method returns the number of batches executed since last clean up.

```
public IloInt getNumberOfCapacityCostFunctions() const
```

This method returns the number of capacity cost functions in the invoking resource. One can use several capacity levels and associate increasing costs of resource usage. This feature is only used by the planning engine.

```
public IloInt getNumberOfConnectedResources() const
```

This method returns the number of connected resources to the invoking resource.

```
public IloInt getNumberOfLevelsOfCapacity(IloInt time) const
```

This method returns the number of levels of the capacity cost function valid at the given time for the invoking resource. This feature is only used by the planning engine.

```
public IloInt getNumberOfPlanningSetupModels() const
```

This method returns the number of planning setup models in the invoking resource. This feature is only used by the planning engine.

```
public IloInt getNumberOfSubResources() const
```

This method returns the number of subresources of the invoking resource.

```
public IloNum getPlanningCapacityReductionFactor() const
```

This method returns the factor limiting the bucket capacity of the resource for the planning engine so that the planning optimization is not too optimistic with availability of the resource with respect to real scheduling constraints.

```
public IloMSPlanningSetupModel getPlanningSetupModel(IloInt i) const
```

This method returns the planning setup model with the given index `i` of the invoking resource. This feature is only used by the planning engine.

```
public IloMSPlanningSetupModel getPlanningSetupModelAtTime(IloInt time) const
```

This method returns the planning setup model valid at `time` for the invoking resource. This feature is only used by the planning engine.

```
public IloInt getPlanningSetupModelValidityEnd(IloInt i) const
```

This method returns the end time of the validity interval for the planning setup model with the given index `i` in the invoking resource. This feature is only used by the planning engine.

```
public IloInt getPlanningSetupModelValidityStart(IloInt i) const
```

This method returns the start time of the validity interval for the planning setup model with the given index `i` in the invoking resource. This feature is only used by the planning engine.

```
public IloInt getRank() const
```

This method returns the rank of the resource, as displayed in the **Gantt Diagram**.

```
public IloMSSetupMatrix getSetupMatrix(IloMSIdentifier feature) const
```

This method returns the setup matrix of the invoking resource for the given setup feature. An exception is thrown if the invoking resource has no setup matrix.

```
public IloInt getStartMin() const
```

This method returns the time before which no production activity or setup can start on the resource.

It is the common earliest start time shared by all modes requiring the resource.

```
public IloMSStorageUnit getStorageUnit() const
```

This method returns the associated storage unit.

```
public IloMSResource getSubResource(IloInt index) const
```

This method returns the subresource with the given `index`. An exception is thrown if the given `index` is out of bounds.

```
public IloMSResource getSuperResource() const
```

This method returns the super-resource to which the invoking resource belongs. An exception is thrown if the invoking resource has no super-resource.

```
public IloInt getTimeFence() const
```

This method retrieves the time fence to apply to the model start min at each database session to obtain this resource start min.

```
public IloInt getTimeOfLastCleanup() const
```

This method returns the end time of the last cleanup executed before the origin of the resource.

```
public IloNum getVariableCostOfCapacity(IloInt time, IloInt levelNumber) const
```

This method returns the variable cost defined at level `levelNumber` of the capacity cost function valid at the given time for the invoking resource. This feature is only used by the planning engine.

```
public IloNum getX() const
```

This method returns the X-coordinate in the plant layout representation.

```
public IloNum getY() const
```

This method returns the Y-coordinate in the plant layout representation.

```
public IloBool hasCalendar() const
```

This method returns true if the invoking resource has a calendar and false otherwise.

```
public IloBool hasCapacityCostFunctionAtTime(IloInt time) const
```

This method returns true if a capacity cost functions is valid at `time` for the invoking resource. It returns false otherwise. This feature is only used by the planning engine.

```
public IloBool hasCleanupRecipe() const
```

This method returns true if a cleanup recipe is associated with the resource.

```
public IloBool hasFinalSetupState(IloMSIdentifier feature) const
```

This method returns true if a final setup state has been specified for the invoking resource on the given setup feature `feature`, and false otherwise.

```
public IloBool hasInitialSetupState(IloMSIdentifier feature) const
```

This method returns true if an initial setup state has been specified for the invoking resource on the given setup feature `feature`, and false otherwise.

```
public IloBool hasSetupMatrix(IloMSIdentifier feature) const
```

This method returns true if the invoking resource has a setup matrix for the given setup feature `feature`, and false otherwise.

```
public IloBool hasStorageUnit() const
```

This method returns true if a storage unit is associated to this resource.

```
public IloBool hasSuperResource() const
```

This method returns true if the invoking resource has a super-resource. In such a case, the planning module considers globally the capacity of the super-resource.

```
public IloBool isConnectedWith(IloMSResource otherResource) const
```

This method returns true if a direct or indirect connection exists between the invoking resource and the resource specified as parameter.

Note that a resource *A* is connected with a resource *B* if one of the following is true:

- *A* is directly connected with *B*
- *A* is directly connected with superResource(*B*)
- superResource(*A*) is directly connected with *B*
- superResource(*A*) is directly connected with superResource(*B*).

```
public IloBool isTimeFenceDefined() const
```

This method returns true if a valid time fence has been specified for the invoking resource. Any negative value for the time fence causes this method to return false.

```
public void removeCapacityCostFunction(IloInt i)
```

This method removes the capacity cost function of the given index `i` for the invoking resource. This feature is only used by the planning engine.

```
public void removeCapacityCostFunction(IloMSResourceCapacityCostFunction function,
IloInt start, IloInt end)
```

This method removes the capacity cost function `function` for the invoking resource previously defined between `start` and `end`. This feature is only used by the planning engine.

```
public void removeConnectedResource(IloMSResource otherResource)
```

This method removes a connection between the invoking resource and the resource specified as parameter.

```
public void removePlanningSetupModel(IloInt i)
```

This method removes the planning setup model of the given index `i` for the invoking resource. This feature is only used by the planning engine.

```
public void removePlanningSetupModel(IloMSPlanningSetupModel model, IloInt start,
IloInt end)
```

This method removes the planning setup model `model` from the invoking resource previously defined between `start` and `end`. This feature is only used by the planning engine.

```
public void setCalendar(IloMSCalendar calendar)
```

This method sets the calendar of the invoking resource. The calendar on the resource is the "default" calendar used when no calendar is specified on the mode.

```
public void setCapacityCostFunction(IloMSResourceCapacityCostFunction function,
IloInt validityStart, IloInt validityEnd)
```

This method sets the capacity cost function `function` for the invoking resource between `validityStart` and `validityEnd`. This feature is only used by the planning engine.

```
public void setCategory(const char * category)
```

This method sets the desired category on this resource to be interpreted for symbolic representation in a specialized editor.

The expected values are: `column`, `cooler`, `filter`, `mixer`, `operator`, `packer`, `pasteurizer`, `separator`, `sterilizer`, `tank` or `team`.

```
public void setCleanupRecipe(IloMSRecipe recipe)
```

This method sets the recipe on the invoking resource that allows creating cleanup orders when required.

```
public void setDisplayRank(IloInt rank)
```

This method sets the desired rank of the resource as displayed in the GUI: The smaller the rank value, the higher the resource appears in the **Gantt Diagram**.

```
public void setEndMax(IloInt endMax)
```

This method sets the time after which no production activity or setup can end on the resource.

It is the common latest end time shared by all modes requiring the resource.

```
public void setFinalSetupState(IloMSIdentifier feature, IloMSIdentifier
initialSetupState)
```

This method sets the required final setup state of the invoking resource for the given setup `feature`.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking resource. An exception is thrown if the given identifier is already used.

```
public void setInitialSetupState(IloMSIdentifier initialSetupState)
```

This method sets the initial setup state of the invoking resource. This is equivalent to `setInitialSetupState(IloMSNoFeature, initialSetupState)`.

```
public void setInitialSetupState(IloMSIdentifier feature, IloMSIdentifier
initialSetupState)
```

This method sets the initial setup state of the invoking resource for the given setup `feature`.

```
public void setLineId(IloMSIdentifier id)
```

This method sets the line identifier of the invoking resource to the given value. The parameter `id` is the line identifier of the resource. An exception is thrown if the given `id` is negative.

```
public void setMaxIdleTimeBeforeCleanup(IloInt value)
```

This method sets the maximum number of time units between two cleanups if the resource is idle.

```
public void setMaxNumberOfBatchesBeforeCleanup(IloInt value)
```

This method sets the maximum number of batches between two cleanups.

```
public void setMaxTimeBeforeCleanup(IloInt value)
```

This method sets the maximum number of time units between two cleanups.

```
public void setNumberOfBatchesSinceLastCleanup(IloInt numberOfBatches)
```

This method sets the number of batches executed since last clean up (before the start time of the model).

```
public void setPlanningCapacityReductionFactor(IloNum factor)
```

This method applies a `factor` between 0.0 and 1.0 to limit the bucket capacity for the planning engine so that the planning optimization is not too optimistic with availability of the resource with respect to real scheduling constraints.

```
public void setPlanningSetupModel(IloMSPlanningSetupModel model, IloInt
validityStart, IloInt validityEnd)
```

This method sets the planning setup model `model` for the invoking resource between `validityStart` and `validityEnd`. The default model is `NoSetup` which means that setup activities are not taken into account. This feature is only used by the planning engine.

```
public void setRank(IloInt rank)
```

This method sets the desired rank of the resource as displayed in the GUI: The smaller the rank value, the higher the resource appears in the **Gantt Diagram**.

```
public void setSetupMatrix(IloMSSetupMatrix setupMatrix)
```

This method sets the setup matrix of the invoking resource to the given `setupMatrix`. This is equivalent to `setSetupMatrix(IloMSNoFeature,setupMatrix)`.

An exception is thrown if the invoking resource already has a setup matrix.

```
public void setSetupMatrix(IloMSIdentifier feature, IloMSSetupMatrix setupMatrix)
```

This method sets the setup matrix of the invoking resource for the given setup feature to the given `setupMatrix`. An exception is thrown if the invoking resource already has a setup matrix.

```
public void setStartMin(IloInt startMin)
```

This method sets the time before which no production activity or setup can start on the resource.

It is the common earliest start time shared by all modes requiring the resource.

```
public void setSuperResource(IloMSResource superResource)
```

This method sets the super-resource of the invoking resource. Note that hierarchies with three levels or more are not supported. An exception is thrown if the invoking resource has a subresource or if the given `superResource` has a super-resource itself.

```
public void setTimeFence(IloInt timeFence)
```

This method sets the time fence to apply to the model start min at each database session to obtain this resource start min. The previous value of startMin of this resource will be conserved if it is greater than this computed value.

```
public void setTimeOfLastCleanup(IloInt value)
```

This method sets the end time of the last cleanup executed before the origin of the resource.

```
public void setX(IloNum x)
```

This method sets the desired X-coordinate for plant layout representation.

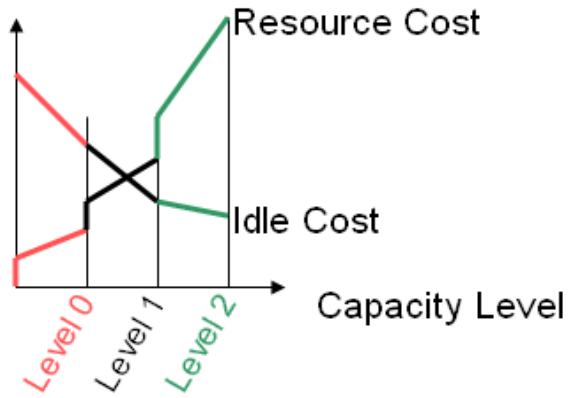
```
public void setY(IloNum y)
```

This method sets the desired Y-coordinate for plant layout representation.

# Class IloMSResourceCapacityCostFunction

**Definition file:** ilplant/costfunctions.h
**Library:** plant


The class `IloMSResourceCapacityCostFunction` is used to evaluate the cost of using a resource over time.
An instance of `IloMSResourceCapacityCostFunction` is used to represent a piecewise or stepwise linear cost function used to evaluate the cost of using a resource over time. It also allows penalizing idle resources by using a piecewise linear idle cost. This feature is only used by the planning engine.




**See Also:** IloMSRecipe, IloMSResource

| Method Summary | |
|---|---|
| public IloNum | getCapacity(IloInt level) const |
| public IloNum | getFixedCost(IloInt level) const |
| public IloNum | getIdleVariableCost(IloInt level) const |
| public IloInt | getNumberOfLevels() |
| public IloNum | getVariableCost(IloInt level) const |
| public void | setCapacity(IloInt level, IloNum value) |
| public void | setFixedCost(IloInt level, IloNum value) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setIdleVariableCost(IloInt level, IloNum value) |
| public void | setVariableCost(IloInt level, IloNum value) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloNum getCapacity(IloInt level) const
```

This method returns the capacity limit up to which this level applies.

```
public IloNum getFixedCost(IloInt level) const
```

This method returns the fixed cost incurred to enter this level of capacity; that is, when the capacity of the previous level is exceeded.

```
public IloNum getIdleVariableCost(IloInt level) const
```

This method returns the slope of the piecewise linear cost function when not using the capacity included in the [floor;ceil) segment of the level.

```
public IloInt getNumberOfLevels()
```

This modifier returns the number of levels associated with the invoking function.

```
public IloNum getVariableCost(IloInt level) const
```

This method returns the slope of the piecewise linear cost function for this level of capacity; that is, for capacity included in this level (from the capacity of the previous level to the capacity of this level).

```
public void setCapacity(IloInt level, IloNum value)
```

This method sets the capacity limit (between 0 and the capacity of the resource) up to which this `level` applies.

For example, for a resource of capacity 2, if we want the average capacity on a time bucket to be penalized above 66% usage, then we must create two levels; one level with capacity 1.32 and a zero variable cost, and one level with a capacity of 2.0 or higher and a positive variable cost.

```
public void setFixedCost(IloInt level, IloNum value)
```

This method sets the fixed cost incurred to enter this level of capacity; that is, when the capacity of the previous level is exceeded.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking function.

```
public void setIdleVariableCost(IloInt level, IloNum value)
```

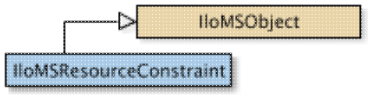
This method sets the slope of the piecewise linear cost function when not using the capacity included in the [floor;ceil) segment of the level.

```
public void setVariableCost(IloInt level, IloNum value)
```

This method sets the slope of the piecewise linear cost function for this level of capacity; that is, for capacity included in this level (from the capacity of the previous level to the capacity of this level).

# Class IloMSResourceConstraint

**Definition file:** ilplant/rescst.h
**Library:** plant




The class `IloMSResourceConstraint` is used to represent a resource requirement on an activity.
This constraint enables the mode of an activity to require a certain amount of capacity on a given resource. Note that a resource may be required as the primary or the secondary resource by an activity. Only one resource can be required as primary for a given mode; there can be many secondary resource constraints. When used as secondary, the setup and cleanup features are not available.

**See Also:** IloMSAbstractActivity, IloMSMode, IloMSResource

| Method Summary | |
|---|---|
| public IloMSMode | getMode() const |
| public IloInt | getRequiredCapacity() const |
| public IloMSResource | getResource() const |
| public IloBool | isPrimary() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

```
public IloMSMode getMode() const
```

This method returns the mode of an activity requiring a resource.

```
public IloInt getRequiredCapacity() const
```

This method returns the capacity required on the resource by the activity in the mode.

```
public IloMSResource getResource() const
```

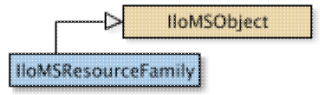
This method returns the required resource.

```
public IloBool isPrimary() const
```

This method returns true if the resource is required as primary resource.

# Class IloMSResourceFamily

**Definition file:** ilplant/resourcefamily.h
**Library:** plant




The `IloMSResourceFamily` class is used to represent resource families. A resource may be a member of several families. Families are grouped by their type in the GUI for aggregation purpose.

**See Also:** IloMSResource

| Method Summary | |
|---|---|
| public void | add(IloMSResource resource) |
| public void | display(ostream & stream) const |
| public IloInt | getNumberOfResources() const |
| public IloMSResource | getResource(IloInt index) const |
| public IloMSIdentifier | getType() const |
| public IloBool | isMember(IloMSResource resource) const |
| public void | remove(IloMSResource resource) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setType(IloMSIdentifier type) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`add`**`(IloMSResource resource)`

This method adds the resource to the invoking family.

`public void` **`display`**`(ostream & stream) const`

This method displays the resource family in the stream passed as argument.

`public IloInt` **`getNumberOfResources`**`() const`

This method returns the number of resources in this family.

`public IloMSResource` **`getResource`**`(IloInt index) const`

This method returns the resource with the `index` in the invoking family.

```
public IloMSIdentifier getType() const
```

This method retrieves the type of the family.

```
public IloBool isMember(IloMSResource resource) const
```

This method return true if the resource is a member of the invoking family.

```
public void remove(IloMSResource resource)
```

This method removes the resource from the invoking family.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking resource family. An exception is thrown if the given `identifier` is already used.

```
public void setType(IloMSIdentifier type)
```

This method registers the invoking family in the collection of families of same type.

# Class IloMSScheduledActivity

**Definition file:** ilplant/schedact.h
**Library:** plant


The class `IloMSScheduledActivity` is used to represent an activity as scheduled on the shop floor.

| Method Summary | |
|---|---|
| public void | addFirmPossibleMode(IloMSMode mode) |
| public void | addPossibleMode(IloMSMode mode) |
| public void | display(ostream & stream) const |
| public IloMSAbstractActivity | getActivityPrototype() const |
| public IloInt | getEndMax() const |
| public IloInt | getEndMin() const |
| public IloInt | getEndTime() const |
| public IloInt | getFirmEndMax() const |
| public IloInt | getFirmEndMin() const |
| public IloInt | getFirmModeNumber() const |
| public IloMSPerformedStatus | getFirmPerformedStatus() const |
| public IloInt | getFirmPossibleModeNumber(IloInt i) const |
| public IloInt | getFirmStartMax() const |
| public IloInt | getFirmStartMin() const |
| public IloMSAbstractActivity | getGeneratedActivity() const |
| public IloInt | getModeNumber() const |
| public IloInt | getNumberOfFirmPossibleModes() const |
| public IloInt | getNumberOfPossibleModes() const |
| public IloMSPerformedStatus | getPerformedStatus() const |
| public IloInt | getPossibleModeNumber(IloInt i) const |
| public IloMSProductionOrder | getProductionOrder() const |
| public IloMSSchedulingSolution | getSchedulingSolution() const |
| public IloInt | getStartMax() const |
| public IloInt | getStartMin() const |
| public IloInt | getStartTime() const |
| public void | removeFirmPossibleMode(IloMSMode mode) |
| public void | removePossibleMode(IloMSMode mode) |
| public void | setEndTime(IloInt endTime) |
| public void | setFirmEndMax(IloInt firmEndMax) |
| public void | setFirmEndMin(IloInt firmEndMin) |
| public void | setFirmModeNumber(IloInt firmModeNumber) |

| | |
|---|---|
| public void | setFirmPerformedStatus(IloMSPerformedStatus status) |
| public void | setFirmStartMax(IloInt firmStartMax) |
| public void | setFirmStartMin(IloInt firmStartMin) |
| public void | setModeNumber(IloInt modeNumber) |
| public void | setPerformedStatus(IloMSPerformedStatus status) |
| public void | setStartTime(IloInt startTime) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`addFirmPossibleMode`**`(IloMSMode mode)`

This method adds `mode` to the set of firm possible modes for the scheduled activity.

`public void` **`addPossibleMode`**`(IloMSMode mode)`

This method adds `mode` to the possible modes for the scheduled activity.

`public void` **`display`**`(ostream & stream) const`

This method displays the scheduled activity in the stream passed as argument.

`public IloMSAbstractActivity` **`getActivityPrototype`**`() const`

This method returns the activity prototype to which the invoking scheduled activity is associated.

`public IloInt` **`getEndMax`**`() const`

This method returns the maximal end time of the scheduled activity.

`public IloInt` **`getEndMin`**`() const`

This method returns the minimal end time of the scheduled activity.

`public IloInt` **`getEndTime`**`() const`

This method returns the end time that is recommended for the invoking scheduled activity.

```
public IloInt getFirmEndMax() const
```

This method returns the firm maximal end time of the invoking scheduled activity. This is usually known when the activity has been completed on the shop floor.

```
public IloInt getFirmEndMin() const
```

This method returns the firm minimal end time of the invoking scheduled activity.

```
public IloInt getFirmModeNumber() const
```

This method returns the number of the firm mode of the scheduled activity, if the firm mode is bound. It returns -1 if the firm mode is not bound.

```
public IloMSPerformedStatus getFirmPerformedStatus() const
```

This method returns the possible firm performed status of the scheduled activity.

**See Also:** IloMSPerformedStatus

```
public IloInt getFirmPossibleModeNumber(IloInt i) const
```

This method returns the number of the i-th firm possible mode for the scheduled activity.

```
public IloInt getFirmStartMax() const
```

This method returns the firm maximal start time of the invoking scheduled activity. This is usually known when the activity is about to start on the shop floor.

```
public IloInt getFirmStartMin() const
```

This method returns the firm minimal start time of the invoking scheduled activity.

```
public IloMSAbstractActivity getGeneratedActivity() const
```

This method returns the generated activity that corresponds to the invoking scheduling activity if it exists, and `null` otherwise.

```
public IloInt getModeNumber() const
```

This method returns the mode number of the scheduled activity, if the mode is bound. It returns -1 if the mode is not bound.

```
public IloInt getNumberOfFirmPossibleModes() const
```

This method returns the number of firm possible modes for the scheduled activity.

```
public IloInt getNumberOfPossibleModes() const
```

This method returns the number of possible modes for the scheduled activity.

```
public IloMSPerformedStatus getPerformedStatus() const
```

This method returns the possible performed status of the scheduled activity.

**See Also:** IloMSPerformedStatus

```
public IloInt getPossibleModeNumber(IloInt i) const
```

This method returns the number of the i-th possible mode for the scheduled activity.

```
public IloMSProductionOrder getProductionOrder() const
```

This method returns the production order to which the invoking scheduled activity is associated.

```
public IloMSSchedulingSolution getSchedulingSolution() const
```

This method returns the scheduling solution that manages the invoking scheduled activity.

```
public IloInt getStartMax() const
```

This method returns the maximal start time of the scheduled activity.

```
public IloInt getStartMin() const
```

This method returns the minimal start time of the scheduled activity.

```
public IloInt getStartTime() const
```

This method returns the start time that is recommended for the invoking scheduled activity.

```
public void removeFirmPossibleMode(IloMSMode mode)
```

This method removes `mode` from the set of firm possible modes for the scheduled activity.

`public void` **`removePossibleMode`**`(IloMSMode mode)`

This method removes `mode` from the set of possible modes for the scheduled activity.

`public void` **`setEndTime`**`(IloInt endTime)`

This method sets the end time of the invoking scheduled activity.

`public void` **`setFirmEndMax`**`(IloInt firmEndMax)`

This method sets the firm maximal end time of the invoking scheduled activity.

`public void` **`setFirmEndMin`**`(IloInt firmEndMin)`

This method sets the firm minimal end time of the invoking scheduled activity.

`public void` **`setFirmModeNumber`**`(IloInt firmModeNumber)`

This method sets the firm mode number of the invoking scheduled activity.

`public void` **`setFirmPerformedStatus`**`(IloMSPerformedStatus status)`

This method sets the firm performed status of the scheduled activity to `status`.

**See Also:** IloMSPerformedStatus

`public void` **`setFirmStartMax`**`(IloInt firmStartMax)`

This method sets the firm maximal start time of the invoking scheduled activity.

`public void` **`setFirmStartMin`**`(IloInt firmStartMin)`

This method sets the firm minimal start time of the invoking scheduled activity.

`public void` **`setModeNumber`**`(IloInt modeNumber)`

This method sets the number of the mode that is recommended for executing the invoking scheduled activity.

`public void` **`setPerformedStatus`**`(IloMSPerformedStatus status)`

This method sets the performed status of the scheduled activity to `status`.

**See Also:** IloMSPerformedStatus

```
public void setStartTime(IloInt startTime)
```

This method sets the start time of the invoking scheduled activity.

# Class IloMSSchedulingEngine

**Definition file:** ilplant/schedengine.h
**Library:** plant


The `IloMSSchedulingEngine` class implements a scheduling engine that uses the current batching solution (`getCurrentBatchingSolution` of `IloMSModel`) to compute a scheduling solution. The solution can be accessed with `getCurrentSchedulingSolution` of `IloMSModel`.
In a scheduling solution, the activities for the production orders have been created (instances of `IloMSScheduledActivity`), and they have been given an execution status, a mode and start and end dates. In other words, the scheduling engine always returns a complete solution unless there is a failure.

The usage pattern is the following: First create an instance of `IloMSSchedulingEngine` (using the static method `newSchedulingEngine` in Java(TM) or the constructor in C++). If needed, configure the engine using the methods `setFirstSolutionTimeLimit`, `setNumberOfSolutionsLimit`, and `setTimeLimit`. If you want to be notified each time a solution is found by the engine, set a call-back using `whenSolution`. If you want to be able to stop the engine before the end of the search, use `setCheckForStop`. Call `solve`. The engine starts computing solutions, and always keeps the best one with respect to the KPIs defined in the current optimization profile (`IloMSModel::getCurrentOptimizationProfile`).

**See Also:** IloMSModel

| Constructor Summary | |
|---|---|
| public | IloMSSchedulingEngine(IloMSModel plant) |

| Method Summary | |
|---|---|
| public void | end() |
| public void | setCheckForStop(IloMSCheckForStop checkForStop) |
| public void | setFirstSolutionTimeLimit(IloInt timeLimit) |
| public void | setNumberOfSolutionsLimit(IloInt limit) |
| public void | setTimeLimit(IloInt timeLimit) |
| public IloBool | solve() |
| public void | whenSolution(IloMSSolutionHook whenSolutionHook) |

## Constructors

`public` **`IloMSSchedulingEngine`**`(IloMSModel plant)`

This constructor creates a scheduling engine for the given `IloMSModel` object.

**See Also:** IloMSModel

## Methods

`public void` **`end`**`()`

This method releases the memory allocated by the invoking scheduling engine.

```
public void setCheckForStop(IloMSCheckForStop checkForStop)
```

This method sets the stopping conditions defined by the object `checkForStop`.

```
public void setFirstSolutionTimeLimit(IloInt timeLimit)
```

This method sets the maximal amount of time allowed to find the first solution. By default, it is equal to the global time limit. It can exceed the global time limit. If a first solution is found before the global time limit, then the search continues and stops at the global time limit. Otherwise, it stops either as soon as a first solution is found or when the "first solution time limit" is reached.

```
public void setNumberOfSolutionsLimit(IloInt limit)
```

This method sets the maximal number of solutions allowed during the solving process. By default, there is no limit in the number of solutions.

```
public void setTimeLimit(IloInt timeLimit)
```

This method sets the maximal amount of time (in seconds) allowed during the solving process. The default time limit is ten seconds.

```
public IloBool solve()
```

This method solves the scheduling problem using the default strategy. It returns true if a solution is found; false otherwise.

```
public void whenSolution(IloMSSolutionHook whenSolutionHook)
```

This method sets a call-back that is executed every time a new solution is found.

# Class IloMSSchedulingSolution

**Definition file:** ilplant/schedsolution.h
**Library:** plant


The `IloMSSchedulingSolution` class is used to represent partial and complete solutions to the scheduling problem defined in the corresponding instance of `IloMSModel`.
A solution is complete (`isBound()` returns `true`) if and only if all the activities have a defined performed status, mode, and start and end times.

Note that a scheduling solution reference obtained before solving is no longer valid after solving.

All methods of the `IloMSSchedulingSolution` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public void | addPossibleMode(IloMSMode mode) |
| public IloBool | computeDynamicPegging() |
| public IloBool | containsPossibleMode(IloMSMode mode) const |
| public IloNum | getAmountOfSatisfiedDemand(IloMSDemand demand) const |
| public IloMSChecker | getChecker() |
| public IloMSCleaningStatus | getCleaningStatus(IloMSSetupActivity activity) const |
| public IloNum | getEarlinessCost(IloMSAbstractActivity activity) const |
| public IloInt | getEndMax(IloMSAbstractActivity activity) const |
| public IloInt | getEndMin(IloMSAbstractActivity activity) const |
| public IloInt | getEndTime(IloMSAbstractActivity activity) const |
| public IloMSActivity | getFirstActivity(IloMSResource res) const |
| public IloNum | getInventoryLevel(IloMSMaterial material, IloInt time) const |
| public IloMSMode | getMode(IloMSAbstractActivity activity) const |
| public IloNum | getModeCost(IloMSActivity activity) const |
| public IloMSActivity | getNextActivity(IloMSActivity act) const |
| public IloInt | getNumberOfScheduledActivities() const |
| public IloMSPerformedStatus | getPerformedStatus(IloMSAbstractActivity activity) const |
| public IloMSScheduledActivity | getScheduledActivity(IloMSAbstractActivity act) const |
| public IloMSScheduledActivity | getScheduledActivity(IloInt i) const |
| public IloNum | getSetupCost(IloMSAbstractActivity activity) const |
| public IloInt | getSetupTime(IloMSAbstractActivity activity) const |
| public IloNum | |

| | |
|---|---|
| | getShippingQuantity(IloMSProcurementToDemandArc arc) const |
| public IloNum | getShippingQuantity(IloMSProdToDemandArc arc) const |
| public IloInt | getStartMax(IloMSAbstractActivity activity) const |
| public IloInt | getStartMin(IloMSAbstractActivity activity) const |
| public IloInt | getStartTime(IloMSAbstractActivity activity) const |
| public IloNum | getTardinessCost(IloMSAbstractActivity activity) const |
| public IloNum | getValue(IloMSOptimizationCriterion criterion) const |
| public IloBool | hasBoundMode(IloMSAbstractActivity activity) const |
| public IloBool | isBound() const |
| public void | removePossibleMode(IloMSMode mode) |
| public void | setCleaningStatus(IloMSSetupActivity activity, IloMSCleaningStatus status) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setMode(IloMSMode mode) |
| public void | setPerformedStatus(IloMSAbstractActivity activity, IloMSPerformedStatus status) |
| public void | setValue(IloMSOptimizationCriterion criterion, IloNum value) |
| public void | unsetMode(IloMSAbstractActivity activity) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`addPossibleMode`**`(IloMSMode mode)`

This method adds the given `mode` to the possible modes of the activity in the invoking solution.

`public IloBool` **`computeDynamicPegging`**`()`

This methods recomputes a pegging for all the materials that use the dynamic pegging strategy.

All existing pegging arcs are removed and new ones are created. The arcs with material having a `Static` pegging type are untouched.

`public IloBool` **`containsPossibleMode`**`(IloMSMode mode) const`

This method returns true if the given `mode` is possible according to the invoking solution.

`public IloNum` **`getAmountOfSatisfiedDemand`**`(IloMSDemand demand) const`

This method returns the quantity of shipped material for the specified `demand`.

```
public IloMSChecker getChecker()
```

This method returns an instance of `IloMSChecker` that can be used to verify the validity of the solution.

```
public IloMSCleaningStatus getCleaningStatus(IloMSSetupActivity activity) const
```

This method returns the possible cleaning status of `activity` in the invoking solution.

**See Also:** IloMSCleaningStatus

```
public IloNum getEarlinessCost(IloMSAbstractActivity activity) const
```

This method returns the earliness cost of the `activity`.

```
public IloInt getEndMax(IloMSAbstractActivity activity) const
```

This method returns the maximal end time of `activity` in the invoking solution.

```
public IloInt getEndMin(IloMSAbstractActivity activity) const
```

This method returns the minimal end time of `activity` in the invoking solution.

```
public IloInt getEndTime(IloMSAbstractActivity activity) const
```

This method returns the end time of `activity` in the invoking solution. An exception is thrown if the given activity has an unbound end time in the invoking solution.

```
public IloMSActivity getFirstActivity(IloMSResource res) const
```

This method returns the first production activity scheduled on the argument resource. It returns an empty handle if no activity is scheduled on this resource.

```
public IloNum getInventoryLevel(IloMSMaterial material, IloInt time) const
```

This method returns the quantity in inventory at the specified `time` for the specified `material`.

```
public IloMSMode getMode(IloMSAbstractActivity activity) const
```

This method returns the mode selected for `activity` in the invoking solution. An exception is thrown if the mode is not fixed (that is, more than one mode is possible).

```
public IloNum getModeCost(IloMSActivity activity) const
```

This method returns the mode cost of the `activity`.

```
public IloMSActivity getNextActivity(IloMSActivity act) const
```

This method returns the next production activity scheduled just after the argument activity, on the same resource. It returns an empty handle if there is no such activity.

```
public IloInt getNumberOfScheduledActivities() const
```

This method returns number of scheduled activities managed by the invoking solution.

```
public IloMSPerformedStatus getPerformedStatus(IloMSAbstractActivity activity)
const
```

This method returns the performed status of `activity` in the invoking solution.

**See Also:** IloMSPerformedStatus

```
public IloMSScheduledActivity getScheduledActivity(IloMSAbstractActivity act) const
```

This method returns the scheduled activity managed by the invoking solution that corresponds to the given generated activity. It returns null if there is no such activity.

Note that a scheduled activity obtained before solving is no longer valid after solving.

```
public IloMSScheduledActivity getScheduledActivity(IloInt i) const
```

This method returns the *i*-th scheduled activity managed by the invoking solution. It throws an exception if there is no such activity.

Note that a scheduled activity obtained before solving is no longer valid after solving.

```
public IloNum getSetupCost(IloMSAbstractActivity activity) const
```

This method returns the setup cost of `activity` in the invoking solution. An exception is thrown if the given `activity` is a production activity without a setup.

```
public IloInt getSetupTime(IloMSAbstractActivity activity) const
```

This method returns the setup time of `activity` in the invoking solution. An exception is thrown if the given `activity` is a production activity without a setup.

```
public IloNum getShippingQuantity(IloMSProcurementToDemandArc arc) const
```

internal

```
public IloNum getShippingQuantity(IloMSProdToDemandArc arc) const
```

internal

```
public IloInt getStartMax(IloMSAbstractActivity activity) const
```

This method returns the maximal start time of `activity` in the invoking solution.

```
public IloInt getStartMin(IloMSAbstractActivity activity) const
```

This method returns the minimal start time of `activity` in the invoking solution.

```
public IloInt getStartTime(IloMSAbstractActivity activity) const
```

This method returns the start time of `activity` in the invoking solution. An exception is thrown if the given activity has an unbound start time in the invoking solution.

```
public IloNum getTardinessCost(IloMSAbstractActivity activity) const
```

This method returns the tardiness cost of the `activity`.

```
public IloNum getValue(IloMSOptimizationCriterion criterion) const
```

This method returns the value of the given `criterion` in the objective function.

```
public IloBool hasBoundMode(IloMSAbstractActivity activity) const
```

This method returns true if there is exactly one mode for `activity` in the invoking solution.

```
public IloBool isBound() const
```

This method returns true if and only if all the decision variables for all the activities in the invoking solution are fixed (activity start and end times, modes and performed status).

```
public void removePossibleMode(IloMSMode mode)
```

This method removes the given `mode` from the possible modes of the activity in the invoking solution.

```
public void setCleaningStatus(IloMSSetupActivity activity, IloMSCleaningStatus
status)
```

This method sets the possible cleaning status of `activity` in the invoking solution.

**See Also:** IloMSCleaningStatus

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking scheduling solution. An exception is thrown if the given `identifier` is already used.

```
public void setMode(IloMSMode mode)
```

This method sets the `mode` as the one selected (that is, the only one possible) for the corresponding activity in the invoking solution.

```
public void setPerformedStatus(IloMSAbstractActivity activity, IloMSPerformedStatus
status)
```

This method sets the performed status of `activity` in the invoking solution.

**See Also:** IloMSPerformedStatus

```
public void setValue(IloMSOptimizationCriterion criterion, IloNum value)
```

This method sets the `value` of the given `criterion` in the solution.

```
public void unsetMode(IloMSAbstractActivity activity)
```

This method unsets the selection of modes: All modes of the `activity` become possible in the solution.

# Class IloMSScope

**Definition file:** ilplant/scope.h
**Library:** plant


The `IloMSScope` class represents *scopes*, used in the GUI as a means to control optimization.
The scope is linked to an optimization profile and is a container of filters. In particular, a scope can contain a recipe family filter (which is composed of one set of frozen recipe families and one set of planned recipe families). You can use a scope to create a submodel based on the core model. The associated filters allow selection of which part of the core model we want to solve in this particular scope. The associated optimization profile permits determination of how to solve the subproblem.

**See Also:** IloMSRecipeFamilyFilter

| Method Summary | |
|---|---|
| public void | addFrozenRecipe(IloMSRecipe recipe) |
| public void | addPlannedRecipe(IloMSRecipe recipe) |
| public IloMSModel | buildSubModel() |
| public void | display(ostream & stream) const |
| public IloInt | getPositionIndex() |
| public IloMSRecipeFamilyFilter | getRecipeFamilyFilter() |
| public IloBool | hasRecipeFamilyFilter() |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setPositionIndex(IloInt scopeIndex) |
| public void | setRecipeFamilyFilter(IloMSRecipeFamilyFilter recipeFamilyFilter) |
| public void | transferResults(IloMSModel subModel, IloMSModel mainModelModel) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public void` **`addFrozenRecipe`**`(IloMSRecipe recipe)`

This method adds a recipe to the list of frozen recipes of the invoking scope.

The list of frozen recipes is used to build the submodel. The recipes and corresponding orders are fixed in the submodel.

`public void` **`addPlannedRecipe`**`(IloMSRecipe recipe)`

This method adds a recipe to the list of planned recipes of the invoking scope.

The list of planned recipes is used to build the submodel. The recipes and corresponding orders are planned and scheduled in the submodel.

```
public IloMSModel buildSubModel()
```

This method builds the submodel associated with the invoking scope.

The scope permits creation of a submodel based on the core model. The associated filters control which part of the core model solved in this particular scope.

- If the scope has an associated recipe family filter, the submodel is generated from the core model, with the filter applied.
- If the scope doesn't have an associated recipe family filter, then the core model is not filtered, and the generated submodel is the same as the core model.

```
public void display(ostream & stream) const
```

This method displays the scope in the `stream` passed as argument.

```
public IloInt getPositionIndex()
```

This method returns the index associated with the invoking scope.

The index is used to display scopes in the desired order in the GUI. It is also used to consecutively solve each subproblem corresponding to each scope.

```
public IloMSRecipeFamilyFilter getRecipeFamilyFilter()
```

This modifier retrieves the recipe family filter in the invoking scope.

```
public IloBool hasRecipeFamilyFilter()
```

This modifier returns true if a recipe family filter has been associated with the invoking scope.

- If the scope has an associated recipe family filter, the submodel is generated from the core model, with this filter applied.
- If the scope doesn't have an associated recipe family filter, then the core model is not filtered, and the generated submodel is the same as the core model.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking scope.

```
public void setPositionIndex(IloInt scopeIndex)
```

This modifier associates an index to the invoking scope.

The index is used to display scopes in the desired order in the GUI. It is also used to consecutively solve each subproblem corresponding to each scope.

```
public void setRecipeFamilyFilter(IloMSRecipeFamilyFilter recipeFamilyFilter)
```

This modifier sets the recipe family filter in the invoking scope.

The recipe family filter is used to filter the core model in order to create a submodel, in order to solve only the filtered part of the process.

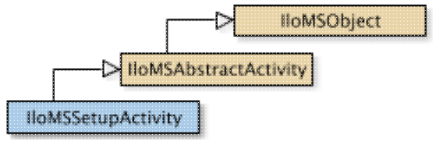
```
public void transferResults(IloMSModel subModel, IloMSModel mainModelModel)
```

This method transfers the results obtained in the submodel to the main (or core) model.

When you finish solving the submodel, it is necessary to transfer the results from the submodel to the core model. Only orders from the list of planned recipes are transferred from the submodel to the core model.

# Class IloMSSetupActivity

**Definition file:** ilplant/setupactivity.h
**Library:** plant




The `IloMSSetupActivity` class is used to **explicitly** represent setup activities.
All the methods of the `IloMSSetupActivity` class throw an exception if an empty handle (that is, an uninitialized object) is used.

**See Also:** IloMSAbstractActivity

| Method Summary | |
|---|---|
| `public IloMSActivity` | `getProductionActivity() const` |

| Inherited Methods from `IloMSAbstractActivity` |
|---|
| `getActivityChain, getActivityCompatibilityConstraint, getCleaningStatus, getDueDate, getIncomingPrecedence, getMode, getModePrototype, getNumberOfActivityCompatibilityConstraints, getNumberOfDueDates, getNumberOfIncomingPrecedences, getNumberOfModes, getNumberOfOutgoingPrecedences, getOutgoingPrecedence, getProductionOrder, getRecipe, hasActivityChain, hasProductionOrder, hasRecipe, isActivityPrototype, isProductionActivity, isSetupActivity, setCleaningStatus, setIdentifier, toProductionActivity, toSetupActivity` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

## Methods

`public IloMSActivity` **`getProductionActivity`**`() const`

This method returns the production activity associated with the invoking setup activity.

# Class IloMSSetupMatrix

**Definition file:** ilplant/setupmtx.h
**Library:** plant


The `IloMSSetupMatrix` class is used to store the setup time and cost incurred when two production activities follow each other on the same resource.
A setup type can be associated with each activity. A setup time and a setup cost will be incurred between any two production activities directly following each other on the same unary resource. This setup time and this setup cost depend on the setup types of the two activities. The `IloMSSetupMatrix` class is used to store the setup time and cost between any two setup states `i` and `j`.

PPO supports the notion of setup feature. An activity may require several features to be each in a specific state. The setup times and costs incurred by each feature are additive.

All the methods of the `IloMSSetupMatrix` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| public IloInt | getDefaultSetupCost(IloMSIdentifier state) const |
| public IloInt | getDefaultSetupTime(IloMSIdentifier state) const |
| public IloInt | getNumberOfSetupStates() const |
| public IloInt | getSetupCost(IloMSIdentifier i, IloMSIdentifier j) const |
| public IloMSIdentifier | getSetupState(IloInt index) const |
| public IloInt | getSetupTime(IloMSIdentifier i, IloMSIdentifier j) const |
| public IloBool | isCleanup(IloMSIdentifier i, IloMSIdentifier j) const |
| public void | setDefaultSetup(IloMSIdentifier state, IloInt time, IloInt cost, IloBool cleanup=IloFalse) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setSetup(IloMSIdentifier i, IloMSIdentifier j, IloInt time, IloInt cost, IloBool cleanup=IloFalse) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloInt` **`getDefaultSetupCost`**`(IloMSIdentifier state) const`

This method returns the approximated cost for switching from an unknown state to `state`.

`public IloInt` **`getDefaultSetupTime`**`(IloMSIdentifier state) const`

This method returns the approximated time for switching from an unknown state to `state`.

```
public IloInt getNumberOfSetupStates() const
```

Returns the number of setup states supported by this matrix; that is, the number of lines and columns of the matrix.

```
public IloInt getSetupCost(IloMSIdentifier i, IloMSIdentifier j) const
```

This method returns the setup cost between the given setup states `i` and `j`. An exception is thrown if one of the two given setup types is out of bounds.

```
public IloMSIdentifier getSetupState(IloInt index) const
```

Returns the setup state identifier with the given index.

```
public IloInt getSetupTime(IloMSIdentifier i, IloMSIdentifier j) const
```

This method returns the setup time between the given setup states `i` and `j`. An exception is thrown if one of the two given setup types is out of bounds.

```
public IloBool isCleanup(IloMSIdentifier i, IloMSIdentifier j) const
```

This method returns true if the transition between the given setup states `i` and `j` requires a cleanup. An exception is thrown if one of the two given setup types is out of bounds.

```
public void setDefaultSetup(IloMSIdentifier state, IloInt time, IloInt cost,
IloBool cleanup=IloFalse)
```

This method sets the `time` and `cost` for switching from an unknown state to `state`. When defined, these values override those inferred by the planning engine as setup approximations. Note that the other engines do not use these values.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking setup matrix. An exception is thrown if the given identifier is already used.

```
public void setSetup(IloMSIdentifier i, IloMSIdentifier j, IloInt time, IloInt
cost, IloBool cleanup=IloFalse)
```

This method sets the setup `time` and the setup `cost` between two setup states `i` and `j`. If `cleanup` is true, then this transition requires a major cleanup.

An exception is thrown if the given `i` or `j` are out of bounds, or if `time` is negative.

# Class IloMSSolutionHook

**Definition file:** ilplant/control.h
**Library:** plant


The `IloMSSolutionHook` class provides a call-back mechanism that is activated every time a new solution is found.
The `IloMSSolutionHookI` and `IloMSSolutionHook` classes enable you to write code that will be executed every time a new solution is found by one of engines of Plant PowerOps. The defined `IloMSSolutionHook` object must be set to the engines in order for it to be considered.

The `IloMSSolutionHook` class is the handle class and contains a pointer to an object of type `IloMSSolutionHookI`. The `IloMSSolutionHookI` class is a pure virtual class. You need to redefine the function `execute` in order to specify actions that must be taken every time the engines of Plant PowerOps finds a new solution. The newly found solution is given in the function argument `sol` of the `execute` callback.

**See Also:** IloMSPlanningSolution, IloMSPlanningEngine, IloMSBatchingSolution, IloMSBatchingEngine, IloMSSchedulingSolution, IloMSSchedulingEngine

| Method Summary | |
|---|---|
| public void | endBatching(IloMSModel model) const |
| public void | endPlanning(IloMSModel model) const |
| public void | endScheduling(IloMSModel model) const |
| public void | execute(IloMSBatchingSolution sol) const |
| public void | execute(IloMSPlanningSolution sol) const |
| public void | execute(IloMSSchedulingSolution sol) const |
| public IloNum | getElapsedTime() const |
| public void | startBatching(IloMSModel model) const |
| public void | startBatchingPolishing(IloMSModel model) const |
| public void | startPlanning(IloMSModel model) const |
| public void | startPlanningPolishing(IloMSModel model) const |
| public void | startScheduling(IloMSModel model) const |
| public void | startSchedulingPolishing(IloMSModel model) const |

## Methods

`public void` **`endBatching`**`(IloMSModel model) const`

Notification method.

`public void` **`endPlanning`**`(IloMSModel model) const`

Notification method.

`public void` **`endScheduling`**`(IloMSModel model) const`

Notification method.

```
public void execute(IloMSBatchingSolution sol) const
```

This method calls the virtual method of the implementation class `IloMSSolutionHookI`.

```
public void execute(IloMSPlanningSolution sol) const
```

This method calls the virtual method of the implementation class `IloMSSolutionHookI`.

```
public void execute(IloMSSchedulingSolution sol) const
```

This method calls the virtual method of the implementation class `IloMSSolutionHookI`.

```
public IloNum getElapsedTime() const
```

This method returns the time, in seconds, elapsed since the call to the `solve()` method.

```
public void startBatching(IloMSModel model) const
```

Notification method.

```
public void startBatchingPolishing(IloMSModel model) const
```

Notification method.

```
public void startPlanning(IloMSModel model) const
```

Notification method.

```
public void startPlanningPolishing(IloMSModel model) const
```

Notification method.

```
public void startScheduling(IloMSModel model) const
```

Notification method.

```
public void startSchedulingPolishing(IloMSModel model) const
```

Notification method.

# Class IloMSSolutionHookI

**Definition file:** ilplant/control.h
**Library:** plant


The `IloMSSolutionHookI` class provides a call-back mechanism that is activated every time a new solution is found.
The `IloMSSolutionHookI` and `IloMSSolutionHook` classes enable you to write code that will be executed every time a new solution is found by one of the engines of Plant PowerOps. The defined `IloMSSolutionHook` object must be set to the engines in order for it to be considered.

The `IloMSSolutionHook` class is the handle class and contains a pointer to an object of type `IloMSSolutionHookI`. The `IloMSSolutionHookI` class is a pure virtual class. You need to redefine the function `execute` in order to specify actions that must be taken every time one of the engines of Plant PowerOps finds a new solution. The newly found solution is given in the function argument `sol` of the `execute` callback.

Note that for intermediate solutions, only the costs are given; other values may be omitted by the engines.

**See Also:** IloMSPlanningSolution, IloMSPlanningEngine, IloMSBatchingSolution, IloMSBatchingEngine, IloMSSchedulingSolution, IloMSSchedulingEngine, IloMSSolutionHook

| Constructor Summary | |
|---|---|
| `public` | `IloMSSolutionHookI()` |

| Method Summary | |
|---|---|
| `public virtual void` | `endBatching(IloMSModel model) const` |
| `public virtual void` | `endPlanning(IloMSModel model) const` |
| `public virtual void` | `endScheduling(IloMSModel model) const` |
| `public virtual void` | `execute(IloMSPlanningSolution sol) const` |
| `public virtual void` | `execute(IloMSBatchingSolution sol) const` |
| `public virtual void` | `execute(IloMSSchedulingSolution sol) const` |
| `public virtual void` | `startBatching(IloMSModel model) const` |
| `public virtual void` | `startBatchingPolishing(IloMSModel model) const` |
| `public virtual void` | `startPlanning(IloMSModel model) const` |
| `public virtual void` | `startPlanningPolishing(IloMSModel model) const` |
| `public virtual void` | `startScheduling(IloMSModel model) const` |
| `public virtual void` | `startSchedulingPolishing(IloMSModel model) const` |

## Constructors

`public` **`IloMSSolutionHookI`**`()`

This constructor creates a new instance of the class.

## Methods

```
public virtual void endBatching(IloMSModel model) const
```

This notifies that batching has ended.

```
public virtual void endPlanning(IloMSModel model) const
```

This notifies that planning has ended.

```
public virtual void endScheduling(IloMSModel model) const
```

This notifies that scheduling has ended.

```
public virtual void execute(IloMSPlanningSolution sol) const
```

This method is the virtual function to be redefined for reacting to when the optimizer finds a new planning solution.

```
public virtual void execute(IloMSBatchingSolution sol) const
```

This method is the virtual function to be redefined for reacting to when the optimizer finds a new batching solution.

```
public virtual void execute(IloMSSchedulingSolution sol) const
```

This method is the virtual function to be redefined for reacting to when the optimizer finds a new scheduling solution.

```
public virtual void startBatching(IloMSModel model) const
```

This notifies that batching has started.

```
public virtual void startBatchingPolishing(IloMSModel model) const
```

This is a notification method.

```
public virtual void startPlanning(IloMSModel model) const
```

This notifies that planning has started.

```
public virtual void startPlanningPolishing(IloMSModel model) const
```

This is a notification method.

```
public virtual void startScheduling(IloMSModel model) const
```

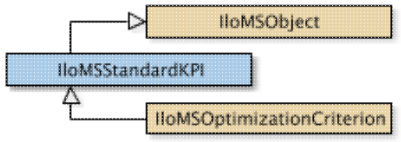
This notifies that scheduling has started.

```
public virtual void startSchedulingPolishing(IloMSModel model) const
```

This is a notification method.

# Class IloMSStandardKPI

**Definition file:** ilplant/kpi.h
**Library:** plant




The `IloMSStandardKPI` class is used to represent the Key Performance Indicators that are defined as *standard* in PPO.
A standard KPI can also be an optimization criterion.

| Method Summary | |
|---|---|
| public IloNum | getValue(IloMSSchedulingSolution solution) const |
| public IloNum | getValue(IloMSBatchingSolution solution) const |
| public IloNum | getValue(IloMSPlanningSolution solution) const |
| public IloBool | isEditable() const |
| public IloBool | isVisible() const |
| public void | setVisible(IloBool visible) const |
| public IloBool | toMinimize() const |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloNum` **`getValue`**`(IloMSSchedulingSolution solution) const`

This method returns the value of the KPI for the `solution` specified as argument.

`public IloNum` **`getValue`**`(IloMSBatchingSolution solution) const`

This method returns the value of the KPI for the `solution` specified as argument.

`public IloNum` **`getValue`**`(IloMSPlanningSolution solution) const`

This method returns the value of the KPI for the `solution` specified as argument.

`public IloBool` **`isEditable`**`() const`

This method returns true if the weight of KPI is editable.

```
public IloBool isVisible() const
```

This method returns true if the KPI is visible.

```
public void setVisible(IloBool visible) const
```

This method enables (makes visible) the KPI.

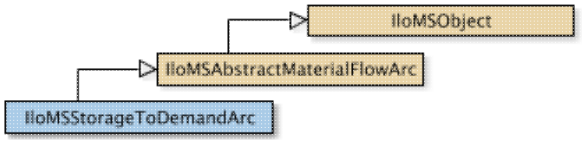
The KPI Comparison Panel displays the visible custom KPIs and the visible standard KPIs with non-null weights. In the KPIs Summary view, the initial KPIs tab shows all visible standard and custom KPIs. The other KPI tabs (such as Standard Scheduling KPIs) display the KPIs with non-null weight regardless of visibility.

```
public IloBool toMinimize() const
```

This method returns true if the goal or best result is to minimize the KPI.

# Class IloMSStorageToDemandArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSStorageToDemandArc` class is used to represent flow of material between stock or storage and a demand. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between storage or stock and a demand is modeled as a precedence constraint between storage and the demand. The material may partially or entirely satisfy the demand.

All the methods of the `IloMSStorageToDemandArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSDemand` | `getDemand() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

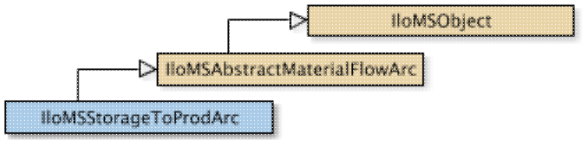
## Methods

`public IloMSDemand` **`getDemand`**`() const`

Returns the successor demand in this arc.

# Class IloMSStorageToProdArc

**Definition file:** ilplant/matflowarc.h
**Library:** plant




The `IloMSStorageToProdArc` class is used to represent flow of material between stock or storage and a production order. Note that a material flow between two nodes represents a precedence constraint between them.
A flow of material between storage or stock and a production order is modeled as a precedence constraint between storage and the consuming activity of a production order.

All the methods of the `IloMSStorageToProdArc` class throw an exception if an empty handle (that is, an uninitialized object) is used.

| Method Summary | |
|---|---|
| `public IloMSProductionOrder` | `getProductionOrder() const` |

| Inherited Methods from `IloMSAbstractMaterialFlowArc` |
|---|
| `getEndConsumptionTime, getEndProductionTime, getFirmQuantityMax, getFirmQuantityMin, getMaterial, getMaterialFlowType, getMaterialFlowTypeName, getPredecessor, getQuantity, getQuantityInDisplayUnit, getStartConsumptionTime, getStartProductionTime, getSuccessor, isFirm, setFirm, setFirmQuantityMax, setFirmQuantityMin, setQuantity, toProcurementToDemandArc, toProcurementToProdArc, toProcurementToStorageArc, toProdToDemandArc, toProdToProdArc, toProdToStorageArc, toStorageToDemandArc, toStorageToProdArc` |

| Inherited Methods from `IloMSObject` |
|---|
| `display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString` |

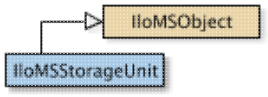
## Methods

`public IloMSProductionOrder` **`getProductionOrder`**`() const`

This method returns the successor to the production order.

# Class IloMSStorageUnit

**Definition file:** ilplant/storageunit.h
**Library:** plant




The `IloMSStorageUnit` class represents a location or storage facility for materials.
Storage units can store one or several materials.

Activities may produce to or consume from a storage unit. Particular storage units can be specified for material procurement or storage. Note that if a storage unit is specified for a material, then any demand, procurement or material production in the recipes referring to that material must also specify the storage unit.

Storage units are useful to model multiple locations in multi-site planning. A longitude, latitude and a category (warehouse, factory) can be specified for representation in the GUI.

If not attached to a unary resource, a storage unit is simply a location (such as a warehouse) with a possible maximum storage capacity. A storage unit attached to a unary resource can model a storage tank. The unary resource ensures the sequencing of batches in the tank.

A group of tanks (or *super storage unit*) can be created by defining a storage unit *S* and attaching it to a super resource *R*. Then each storage unit associated to a subresource of the super resource *R* is a substorage unit of *S*. Such grouping simplifies the planning problem by considering globally the super resource capacities and the super storage unit aggregated volume. It is also useful for specifying the production or consumption of a material in a recipe. Note that the storage unit destination of a material must be consistent with the primary resource required by the activity producing it (if the storage unit is attached to a resource).

When using super storage units, be aware that any material exit (demand, activity consumption) must never refer to a subunit. That is, consumption must remain "fuzzy". Note, however, that initial stock or procurements must refer to *precise* subunits. When a material production on a mode of an activity prototype refers to a super unit at the recipe level, the mode generation on corresponding production orders will generate material productions on precise subunits. Thus any production in the transactional data is precise, and any consumption is fuzzy.

**See Also:** IloMSMaterial, IloMSMaterialProduction, IloMSDemand, IloMSProcurement

| Method Summary | |
|---|---|
| public void | addStorableMaterial(IloMSMaterial material) |
| public IloBool | canStore(IloMSMaterial material) const |
| public const char * | getCategory() const |
| public IloNum | getInitialQuantity(IloMSMaterial material) const |
| public IloNum | getLatitude() const |
| public IloNum | getLongitude() const |
| public IloInt | getMergingLimit(IloMSMaterial material) const |
| public IloInt | getNumberOfStorableMaterials() const |
| public IloNum | getQuantityMax() const |
| public IloNum | getRackingQuantityMin(IloMSMaterial material) const |
| public IloMSResource | getResource() const |
| public IloMSMaterial | getStorableMaterial(IloInt i) const |
| public IloBool | isAutomatic() const |

| | |
|---|---|
| public IloBool | isMultipurpose() const |
| public void | removeStorableMaterial(IloMSMaterial material) |
| public void | setCategory(const char * category) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setInitialQuantity(IloMSMaterial material, IloNum quantity) |
| public void | setLatitude(IloNum latitude) |
| public void | setLongitude(IloNum longitude) |
| public void | setMergingLimit(IloMSMaterial material, IloInt numberMaxOfMixableBatches) |
| public void | setQuantityMax(IloNum maxLevel) |
| public void | setRackingQuantityMin(IloMSMaterial material, IloNum minLevel) |
| public void | setResource(IloMSResource resource) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

public void **addStorableMaterial**(IloMSMaterial material)

This method adds `material` to the list of storable materials of the invoking storage unit.

public IloBool **canStore**(IloMSMaterial material) const

This method returns true if `material` can be stored by the invoking storage unit, and false otherwise.

public const char * **getCategory**() const

This method retrieves the category of the storage unit for graphical interpretation in the **Distribution Planning** view. Expected values are `warehouse` or `factory`.

public IloNum **getInitialQuantity**(IloMSMaterial material) const

This method returns the stocked quantity of `material` in the invoking storage unit at the start time of the planning.

public IloNum **getLatitude**() const

This method retrieves the latitude of the storage unit expressed in degrees [-90;+90] from the equator (negative for Southern Hemisphere).

```
public IloNum getLongitude() const
```

This method retrieves the longitude of the storage unit expressed in degrees [-180;+180] from the Greenwich Meridian (negative for Eastern Hemisphere).

```
public IloInt getMergingLimit(IloMSMaterial material) const
```

This method returns the merging limit for batches of the specified `material` filling this storage unit.

```
public IloInt getNumberOfStorableMaterials() const
```

This method returns the number of materials that can be stored in the invoking storage unit.

```
public IloNum getQuantityMax() const
```

This method returns the maximal level over all the horizon.

```
public IloNum getRackingQuantityMin(IloMSMaterial material) const
```

This method returns the minimum level that must be reached before starting to rack `material` from the invoking storage unit.

```
public IloMSResource getResource() const
```

This method returns the resource associated with a storage unit, if one exists.

```
public IloMSMaterial getStorableMaterial(IloInt i) const
```

This method returns material number `i` from the list of storable materials of the invoking storage unit.

```
public IloBool isAutomatic() const
```

This method returns true if the storage unit has been automatically created by PPO to deal with a material not explicitly used with storage units.

```
public IloBool isMultipurpose() const
```

This method returns true if the storage unit can store several materials.

```
public void removeStorableMaterial(IloMSMaterial material)
```

This method removes `material` from the list of storable materials of the invoking storage unit.

```
public void setCategory(const char * category)
```

This method sets the category of the storage unit for graphical interpretation in the **Distribution Planning** view. Expected values are `warehouse` or `factory`.

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an identifier with the invoking storage unit.

```
public void setInitialQuantity(IloMSMaterial material, IloNum quantity)
```

This method sets the stocked quantity of `material` in the invoking storage unit to `quantity` at the start time of the planning.

```
public void setLatitude(IloNum latitude)
```

This method sets the latitude of the storage unit expressed in degrees [-90;+90] from the equator (negative for Southern Hemisphere).

```
public void setLongitude(IloNum longitude)
```

This method sets the longitude of the storage unit expressed in degrees [-180;+180] from the Greenwich Meridian (negative for Eastern Hemisphere).

```
public void setMergingLimit(IloMSMaterial material, IloInt
numberMaxOfMixableBatches)
```

This method sets the merging limit for batches of the specified `material` filling this storage unit. To prohibit merging batches for a `material`, set `numberMaxOfMixableBatches` to one.

```
public void setQuantityMax(IloNum maxLevel)
```

This method sets the maximal level over all the horizon.

```
public void setRackingQuantityMin(IloMSMaterial material, IloNum minLevel)
```

This method sets the minimal level that must be reached before starting to rack this material from the invoking storage unit.

```
public void setResource(IloMSResource resource)
```

This method assigns a resource to the storage unit to make it behave as a storage tank.

# Class IloMSUnit

**Definition file:** ilplant/unit.h
**Library:** plant


The `IloMSUnit` class is used to represent units of measure.
This class is used to define the standard conversion of a unit of measure to its dimension (if so desired). Some units of measure do not have a dimension or a standard conversion factor. For example, the pallet, the cup, and the box may have different conversion factors, depending on the contained material. Other units of measure do have a dimension, and thus a standard conversion to the metric system, such as length, surface area, volume, and mass. For example, the metric ton is a "mass" dimension, and its conversion factor from the standard unit of the dimension in the metric system (kilogram) is always 1/1000; it does not depend on the material being measured. The standard units are kilogram for mass, meter for length, squared meter for surface area, cubic meter for volume.

To define a secondary (display) unit and its conversion factors, see `IloMSMaterial`.

**See Also:** IloMSMaterial

| Method Summary | |
|---|---|
| public IloNum | getCheckingTolerance() const |
| public static IloNum | getDefaultCheckingTolerance() |
| public IloMSDimension | getDimension() const |
| public IloNum | getStandardConversion() const |
| public void | setCheckingTolerance(IloNum factor) |
| public void | setDimension(IloMSDimension dimension) |
| public void | setIdentifier(IloMSIdentifier identifier) |
| public void | setStandardConversion(IloNum factor) |

| Inherited Methods from `IloMSObject` |
|---|
| display, getIdentifier, getIntProperty, getModel, getName, getNumProperty, getObject, getStringProperty, hasIdentifier, isPropertyDefined, removeAllProperties, removeProperty, setIntProperty, setName, setNumProperty, setObject, setStringProperty, toString |

## Methods

`public IloNum` **`getCheckingTolerance`**`() const`

This modifier returns the tolerance to be used in the checkers, when this unit is used as display unit.

`public static IloNum` **`getDefaultCheckingTolerance`**`()`

This modifier returns the default tolerance to be used in the checkers, when this unit is used as display unit.

`public IloMSDimension` **`getDimension`**`() const`

This accessor returns the dimension of the invoking unit.

**See Also:** IloMSDimension

```
public IloNum getStandardConversion() const
```

This accessor returns the standard conversion factor from the standard unit (of the same dimension) to the invoking unit. An exception is thrown if the invoking unit has no dimension.

**See Also:** IloMSDimension

```
public void setCheckingTolerance(IloNum factor)
```

This modifier associates a tolerance to be used in the checkers when this unit is used as display unit.

Note that if a recipe has a primary product, then the tolerance used to check the batch size of any production orders of the recipe will use this tolerance. The tolerance on the batch size is then equal to the checking tolerance of the display unit of the primary product converted in primary unit and divided by the canonical quantity produced by the recipe.

This methods throws an exception if you try to set a negative tolerance.

**See Also:** IloMSDimension

```
public void setDimension(IloMSDimension dimension)
```

This modifier associates a standard `dimension` with the invoking unit. For units which have different masses depending on the material, use `NoDimension`.

**See Also:** IloMSDimension

```
public void setIdentifier(IloMSIdentifier identifier)
```

This modifier associates an `identifier` with the invoking unit.

```
public void setStandardConversion(IloNum factor)
```

This modifier associates a standard conversion to this unit (instance of `IloMSUnit`) from the standard dimension. The expected value is the multiplication factor to convert a value from the standard unit into this unit instance. For example, if the unit represents the ton, then the standard conversion is 0.001; that is you must multiply by 0.001 any units expressed in the standard unit of mass (kilograms) to convert into tons. An exception is thrown if the invoking unit has no dimension.

The standard units are kilogram for mass, meter for length, squared meter for surface area, cubic meter for volume, and seconds for time.

# Enumeration IloMSActivityCompatibilityType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSActivityCompatibilityType` is used to select a type used to enforce a compatibility constraint. Some of the possible types are **Same Line Id** which constrains the two modes (of the two given activities) to have the same line identifier; **Same Primary Resource** which constrains the two modes to have the same primary resource; **Same Primary Resource And Capacity** constrains the two modes to have the same primary resource and capacity requirement; **Same Performed Status** means that the two activities have to be in the same performed status (either both performed or both unperformed), and **Different Performed Status** means the opposite.

The types **Performed Implies Performed**, **Unperformed Implies Unperformed**, **Performed Implies Unperformed**, and **Unperformed Implies Performed** state that the status of `activity1` logically implies the status of `activity2`.

**Connected Primary Resources** enforces that the primary resource of the two activities have a connection.

**See Also:** IloMSActivityCompatibilityConstraint, IloMSResource, IloMSMode, IloMSActivity

**Fields:**

```
IloMSSameLineId = 0

IloMSSamePrimaryResource = 1

IloMSSamePrimaryResourceAndCapacity = 2

IloMSSamePerformedStatus = 3

IloMSDifferentPerformedStatus = 4

IloMSPerformedImpliesPerformed = 5

IloMSUnperformedImpliesUnperformed = 6

IloMSPerformedImpliesUnperformed = 7

IloMSUnperformedImpliesPerformed = 8

IloMSConnectedPrimaryResources = 9
```

# Enumeration IloMSBatchingAlgorithm

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSBatchingAlgorithm` selects the batching algorithm used by the batching engine.
The **heuristic** engine batches first, then pegs the production orders. **Advanced heuristic** is "smarter" than the simple heuristic in the sense it batches on the fly while pegging orders. **Constraint based** uses constraint programming to solve batching and is able deal with all cases of pegging cardinality constraints (maximum number of incoming or outgoing material flow arcs). **Automatic** tries successively the previous engines, until one succeeds.

**Fields:**

```
IloMSBatchingAlgorithmHeuristic = -1

IloMSBatchingAlgorithmAutomatic = 0

IloMSBatchingAlgorithmConstraintBased = 3

IloMSBatchingAlgorithmAdvancedHeuristic = 4
```

# Enumeration IloMSBucketPeriodUnit

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSBucketPeriodUnit` is used to select the time period over which the buckets will be generated. Period unit is a calendar unit. This period unit must be greater than or equal to the bucket type.
The possible types are:

- **Hour**
- **EightHShift**
- **Day**
- **Week** which starts on Monday
- **Month**
- **Quarter**
- **Year**.

**See Also:** IloMSBucketType, IloMSBucketTemplate, IloMSBucketSequence

**Fields:**

```
IloMSHourPeriod = 0

IloMS8hShiftPeriod = 1

IloMSDayPeriod = 2

IloMSWeekPeriod = 3

IloMSMonthPeriod = 4

IloMSQuarterPeriod = 5

IloMSYearPeriod = 6
```

# Enumeration IloMSBucketType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSBucketType` is used to select the type (the default duration) of buckets defined by bucket templates. The possible types are:

- **Hour**
- **EightHShift** for eight hour buckets starting at midnight. To change the start time use the **OFFSET** of the related `IloMSBucketSequence` object.
- **Day**
- **Week**
- **Month**
- **Quarter**
- **Year**.

**See Also:** IloMSBucketPeriodUnit, IloMSBucketTemplate, IloMSBucketSequence

**Fields:**

```
IloMSHourType = 0

IloMS8hShiftType = 1

IloMSDayType = 2

IloMSWeekType = 3

IloMSMonthType = 4

IloMSQuarterType = 5

IloMSYearType = 6
```

# Enumeration IloMSCheckerMessageLevel

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSCheckerMessageLevel` is used to identify the severity of the checker message.

**Fields:**

```
IloMSCheckerOk = 0
IloMSCheckerWarning = 1
IloMSCheckerError = 2
IloMSCheckerFatal = 3
```

# Enumeration IloMSCleaningStatus

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSCleaningStatus` identifies the cleaning status of an activity.

**Fields:**

```
IloMSCleaning = 0
IloMSNotCleaning = 1
IloMSCanBeCleaning = 2
```

# Enumeration IloMSDay

**Definition file:** ilplant/types.h
**Library:** plant

The enumerated type `IloMSDay` is used to identify the day of the week.

**Fields:**

```
IloMSSunday = 0
IloMSMonday = 1
IloMSTuesday = 2
IloMSWednesday = 3
IloMSThursday = 4
IloMSFriday = 5
IloMSSaturday = 6
```

# Enumeration IloMSDemandCompatibilityType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSDemandCompatibilityType` table places logical constraints on demand deliveries in the planning model.
This table is taken into account only in the Planning module and is usable only with unsplitable demand. Unsplitable demand are specified by putting the max number of pegging arcs of a demand to one.

**Covered Implies Covered** means that if *demand1* is fully satisfied then *demand2* must also be fully satisfied.

**Same Covering Status** means that *demand1* and *demand2* are either both fully satisfied or both unsatisfied.

**Fields:**

```
IloMSSameCoveringStatus = 0
```

```
IloMSCoveredImpliesCovered = 1
```

# Enumeration IloMSDimension

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSDimension` is used to identify the dimension type of the unit.
A unit of measure (an instance of the class `IloMSUnit`) may either have no dimension (such as pallets or boxes), or it can be linked to the international system of measure.

**See Also:** IloMSUnit

**Fields:**

```
IloMSDimensionNoDimension = 0

IloMSDimensionLength = 1

IloMSDimensionSurface = 2

IloMSDimensionVolume = 3

IloMSDimensionTime = 4

IloMSDimensionMass = 5

IloMSNumberOfDimensions = 6
```

# Enumeration IloMSMaterialFlowNodeType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSMaterialFlowNodeType` is used to define the node type of the material flow.

**Fields:**

```
IloMSProductionNode = 0

IloMSStorageNode = 1

IloMSDemandNode = 2

IloMSProcurementNode = 3

IloMSUndefinedNode = 4

IloMSPlannedProductionNode = 5
```

# Enumeration IloMSMaterialFlowType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSMaterialFlowType` is used to define the path of material flow.

**Fields:**

```
IloMSStorageToProduction = 0
IloMSStorageToDemand = 1
IloMSProductionToProduction = 2
IloMSProductionToDemand = 3
IloMSProductionToStorage = 4
IloMSProcurementToProduction = 5
IloMSProcurementToDemand = 6
IloMSProcurementToStorage = 7
IloMSPlannedProductionToDemand = 8
```

# Enumeration IloMSPeggingStrategy

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSPeggingStrategy` is used to identify the pegging strategy to be used with arcs carrying a given `IloMSMaterial`.
Pegging strategy is used by the GUI and can be either *static* or one of two types of *dynamic*. Dynamic pegging strategy allows for the pegging in the **Gantt Diagram** to be updated after a modification is made there. With the dynamic strategy you let PPO recompute the pegging arcs between production orders (instances of `IloMSProdToProdArc`).

The pegging strategies:

- The static strategy `IloMSPeggingStrategyStatic` means that the pegging is calculated by the batching engine or defined interactively and cannot be modified dynamically when moving an activity on the Gantt.
- `IloMSPeggingStrategyDynamicEarliestEndMax` means that the algorithm tries first the producing orders that must be consumed in priority.
- `IloMSPeggingStrategyDynamicFirstInFirstOut` selects from among the different possible producing orders the one that is scheduled the earliest.

**Fields:**

```
IloMSPeggingStrategyStatic = 0

IloMSPeggingStrategyDynamicEarliestEndMax = 1

IloMSPeggingStrategyDynamicFirstInFirstOut = 2

IloMSNumberOfPeggingStrategies = 3
```

# Enumeration IloMSPerformedStatus

**Definition file:** ilplant/types.h
**Library:** plant

The enumerated type `IloMSPerformedStatus` is used to identify the status of an activity in the scheduling solution as performed or unperformed. To let the scheduling engine decide whether or not to perform an activity, the activity must have the status `IloMSPerformedOrUnperformed`.

**Fields:**

```
IloMSUnperformed = 0

IloMSPerformed = 1

IloMSPerformedOrUnperformed = 2
```

# Enumeration IloMSPlanningAlgorithm

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSPlanningAlgorithm` sets the planning algorithm used by the planning engine.
The planning algorithms are useful to deal with infeasibilities in input data, providing different approaches to deal with problems due to the balance of intermediate materials using inflow recipes or waste recipes.

**OnePass** solves a single mathematical programming model with a single weighted objective function, combining business objectives and "technical" costs. With this approach, the costs of processing these recipes must be carefully computed so that the optimizer uses them only as a fallback position, in case of infeasibilities. The drawback of this approach is the potentially wide range in numerical values that coexist during optimization, which can lead to numerical stability issues. The business objectives could get diluted in an objective function containing high technical costs. A bad solution with respect to the business objective may result if the relative gap limit stops optimization when the solution is "good enough."

**MultiPass** uses goal programming to deal with infeasible material flow. The business objective is kept separate from technical costs such as inflow and waste recipe costs or nondelivery costs. This approach assumes it is always better to deliver a product, as compared to other considerations.

The **automatic** selection lets PPO decide which of the two algorithms is used.

**Fields:**

```
IloMSPlanningAlgorithmAutomatic = 0

IloMSPlanningAlgorithmOnePass = 1

IloMSPlanningAlgorithmMultiPass = 2
```

# Enumeration IloMSPlanningSetupModel

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSPlanningSetupModel` is used to define which approximation of the setup model the planning engine must take into account for each resource in a given time interval.
For each resource and each period of time different setup approximations are available.

**Per bucket per recipe** means that the fixed capacity requirements including setups will be counted independently for each bucket and each recipe.

**Per bucket per setup feature** means that the fixed capacity requirements including setups will be counted independently for each bucket and each setup feature.

**Cross bucket per recipe** is similar to "Per bucket per recipe" except that the continuation of the same recipe from a bucket to the next will not necessitate redoing the setups in the second bucket.

**Cross bucket per setup feature** is similar to "Per bucket per setup feature" except that the continuation of the same setup features from one bucket to the next will not necessitate redoing the setups in the second bucket.

**Fields:**

```
IloMSPlanningSetupModelNoSetup = -1

IloMSPlanningSetupModelPerBucketPerRecipe = 0

IloMSPlanningSetupModelPerBucketPerSetupFeature = 1

IloMSPlanningSetupModelCrossBucketPerRecipe = 2

IloMSPlanningSetupModelCrossBucketPerSetupFeature = 3

IloMSNumberOfPlanningSetupModels = 4
```

# Enumeration IloMSPrecedenceType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSPrecedenceType` is used to define the type of precedence constraint between two activities.

**Fields:**

```
IloMSStartToStart = 0

IloMSStartToEnd = 1

IloMSEndToStart = 2

IloMSEndToEnd = 3
```

# Enumeration IloMSRecipeFamilyStatus

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSRecipeFamilyStatus` is used to give more semantics to an `IloMSRecipeFamilyFilter`. `IloMSRecipeFamilyStatus` is used to tag the corresponding `IloMSRecipeFamilyStatus` as one of the following: **Undefined**, **Frozen** or **Planned**. If the type is **Frozen** then recipes of this family and its production orders are fixed into the submodel. If the type is **Planned** then recipes of this family and its production orders are planned into the submodel.

**Fields:**

```
IloMSRecipeFamilyStatusUndefined = -1

IloMSRecipeFamilyStatusFrozen = 0

IloMSRecipeFamilyStatusPlanned = 1
```

# Enumeration IloMSRecipeType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSRecipeType` is used to provide additional semantics to a recipe.
`IloMSRecipeType` is used to tag the corresponding recipe as one of the following: **Fixed**, **Transport**, **Automatic**, **Order**, **Make**, or **Undefined**.

If the type is **Fixed** then the recipe can be used only in manual mode; the planning engine will not be allowed to use it, and only already-existing fixed production orders or orders created interactively can use this recipe. If the type is **Transport** then it means that consuming an SKU and producing another is just a matter of transportation of the same item to a different location. This is used in the GUI to compute the stock in transit (**Stock Coverage** view) and the transported quantity (**Warehouse Summary** view). The value **Automatic** tags the technical waste and inflow recipes created and deleted by PPO to deal with infeasibility.

The other values are purely informative, but can be used in plug-ins or in future versions of PPO. For instance, the type **Order** can be used for recipes simulating a good or material ordered from outside, typically acquired from a third party, as opposed to the type **Make** indicating an in-house production. The "make or order" decision can be taken by the planning engine based on capacity of the plant and recipe costs. Note that if the quantity and receipt date of procured goods are already known, no recipe has to be defined, only the procurements.

**Fields:**

```
IloMSRecipeTypeUndefined = -1

IloMSRecipeTypeMake = 0

IloMSRecipeTypeOrder = 1

IloMSRecipeTypeFixed = 2

IloMSRecipeTypeTransport = 3

IloMSRecipeTypeAutomatic = 4
```

# Enumeration IloMSRepairCapacity

**Definition file:** ilplant/types.h
**Library:** plant

The enumerated type `IloMSRepairCapacity` is used to identify the capacity of repair.
If the value is `IloMSRepairNoResource`, capacity constraints are ignored.

If the value is `IloMSRepairOneResource`, then the capacity constraints of the primary resource of the selected activities will be enforced; activities that overlap will be changed to eliminate this overlap. Note that the selected activities should all have the same primary resource, and it should have a unary capacity.

**See Also:** IloMSRepairAlgorithm

**Fields:**

```
IloMSRepairNoResource = 0
```

```
IloMSRepairOneResource = 1
```

```
IloMSRepairAllResources = 2
```

# Enumeration IloMSRepairExtent

**Definition file:** ilplant/types.h
**Library:** plant

The enumerated type `IloMSRepairExtent` is used to identify the extent of repair with regards to constraints. If the value is `IloMSRepairActivity`, then only the selected activities will have their constraints (duration, calendar) enforced. In particular, precedence constraints will not be considered.

If the value is `IloMSRepairProductionOrder`, then all the activities that belong to the production order(s) of the selected activities will have their constraints enforced. In particular, this means that precedence constraints between those activities will be enforced.

If the value is `IloMSRepairCluster`, then all the activities that belong to clusters of the production order(s) of the selected activities will have their constraints enforced. In particular, this means that pegging constraints between those production orders will be enforced.

**See Also:** IloMSRepairAlgorithm

**Fields:**

```
IloMSRepairSetup = -1
```

```
IloMSRepairActivity = 0
```

```
IloMSRepairProductionOrder = 1
```

```
IloMSRepairCluster = 2
```

# Enumeration IloMSServiceLevelType

**Definition file:** ilplant/types.h
**Library:** plant

`IloMSServiceLevelType` sets the service level.
The service level is used to determine the acceptable level of unsatisfied demand that may occur due to insufficient available stock or inventory. The default value is `IloMSServiceLevelDisabled`, for which no service level stock computations are performed. You can set the service type based on either the probability of a stock-out event, or on overall demand quantity that is satisfied.

The alpha level measures the probability that a stock-out event will not happen. A stock-out event occurs in a given time bucket if the total demand due in the bucket exceeds the inventory quantity that is available in the bucket. This service level is used to set a limit on the frequency (on average) of stock out events. This type is also known as the type 1, cycle, or event-based service level.

The beta level is based on the overall demand quantity which is satisfied. It sets a lower limit on the expected demand satisfaction ratio based on overall demand quantity, irrespective of how many cycles (time buckets) are involved. The beta type is also known as the type 2, fill rate, or quantity-based service level.

**See Also:** IloMSMaterial

**Fields:**

```
IloMSServiceLevelDisabled = 0

IloMSServiceLevelAlpha = 1

IloMSServiceLevelBeta = 2

IloMSServiceLevelAlphaDynamic = 3

IloMSServiceLevelBetaDynamic = 4

IloMSNumberOfServiceLevelTypes = 5
```

# Global function operator<<

```
public ILOMSEXPORTED ostream & operator<<(ostream & s, const IloMSObject & obj)
```

**Definition file:** ilplant/object.h
**Library:** plant

This operator directs output to an output stream, usually standard output.

# Macro ILOMSCHECKFORSTOP0

**Definition file:** ilplant/control.h
**Library:** plant

**ILOMSCHECKFORSTOP0**(_this)

This macro is provided in order to facilitate the definition of the classes `IloMSCheckForStopI` and `IloMSCheckForStop`.
The `IloMSCheckForStopI` and `IloMSCheckForStop` classes enable you to write code that will regularly be executed by the scheduling engine of Plant PowerOps. The defined `IloMSCheckForStop` object must be set to the scheduling engine (`IloMSSchedulingEngine`) in order for it to be considered.

**See Also:** IloMSCheckForStopI, IloMSCheckForStop, IloMSSchedulingEngine

# Macro IloMSIdentifier

**Definition file:** ilplant/types.h
**Library:** plant

**`IloMSIdentifier`**

`IloMSIdentifier` is used to provide a name for programming elements.

# Macro IloMSIntMinusInfinity

**Definition file:** ilplant/types.h
**Library:** plant

**`IloMSIntMinusInfinity`**

`IloMSIntMinusInfinity` represents negative infinity as `-999999999`.

# Macro IloMSIntPlusInfinity

**Definition file:** ilplant/types.h
**Library:** plant

**`IloMSIntPlusInfinity`**

`IloMSIntPlusInfinity` represents positive infinity as `999999999`.

# Macro IloMSNoFeature

**Definition file:** ilplant/types.h
**Library:** plant

**`IloMSNoFeature`**

`IloMSNoFeature` is used to represent that no feature is selected.