# Table of Contents

Welcome to Concert Technology ........................................... 11
Concepts ............................................................................. 15
Group optim.concert ........................................................... .26
ILOANYBINARYPREDICATENAME0 ....................................... 33
ILOANYTERNARYPREDICATENAME0 .................................... 34
ILOINTBINARYPREDICATENAME0 ......................................... 35
ILOINTTERNARYPREDICATENAME0 ....................................... 36
ILO_NO_MEMORY_MANAGER ................................................... 37
IloAbs ............................................................................... 38
IloAbstraction .................................................................... 39
IloAdd ............................................................................... 41
IloAlgorithm ................................................................. ................................. 42
IloAlgorithm::CannotExtractException ................................. 49
IloAlgorithm::CannotRemoveException ..................... ............................. 51
IloAlgorithm::Exception ......................................................... 53
IloAlgorithm::NotExtractedException ...................... ............................... 55
IloAlgorithm::Status .......................................................... 57
IloAllDiff ................................................................. ................................. 59
IloAllMinDistance ............................................................ 62
IloAllNullIntersect ............................................................... 65
IloAnd ............................................................................... 66
IloAny ............................................................................... 70
IloAnyArray ....................................................................... 71
IloAnyBinaryPredicate ......................................................... 73
IloAnySet ................................................................. ................................. 75
IloAnySet::Iterator .............................................................. 79
IloAnySetVar ................................................................. ............................. 80
IloAnySetVarArray ................................................................. 84
IloAnyTernaryPredicate ......................................................... 87
IloAnyTupleSet ................................................................. ............................. 89
IloAnyTupleSetIterator ......................................................... 92
# Table of Contents

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloAnyVar</td>
<td>94</td>
</tr>
<tr>
<td>IloAnyVarArray</td>
<td>96</td>
</tr>
<tr>
<td>IloArcCos</td>
<td>99</td>
</tr>
<tr>
<td>IloArray</td>
<td>100</td>
</tr>
<tr>
<td>IloBarrier</td>
<td>104</td>
</tr>
<tr>
<td>IloBaseEnvMutex</td>
<td>107</td>
</tr>
<tr>
<td>IloBool</td>
<td>108</td>
</tr>
<tr>
<td>IloBoolAbstraction</td>
<td>109</td>
</tr>
<tr>
<td>IloBoolArray</td>
<td>111</td>
</tr>
<tr>
<td>IloBoolVar</td>
<td>114</td>
</tr>
<tr>
<td>IloBoolVarArray</td>
<td>117</td>
</tr>
<tr>
<td>IloBox</td>
<td>120</td>
</tr>
<tr>
<td>IloCard</td>
<td>124</td>
</tr>
<tr>
<td>IloCell</td>
<td>125</td>
</tr>
<tr>
<td>IloCondition</td>
<td>126</td>
</tr>
<tr>
<td>IloConstraint</td>
<td>129</td>
</tr>
<tr>
<td>IloConstraintArray</td>
<td>132</td>
</tr>
<tr>
<td>IloConversion</td>
<td>135</td>
</tr>
<tr>
<td>IloDeleterMode</td>
<td>139</td>
</tr>
<tr>
<td>IloDiff</td>
<td>140</td>
</tr>
<tr>
<td>IloDisableNANDetection</td>
<td>143</td>
</tr>
<tr>
<td>IloDistribute</td>
<td>144</td>
</tr>
<tr>
<td>IloDiv</td>
<td>148</td>
</tr>
<tr>
<td>IloEmptyHandleException</td>
<td>149</td>
</tr>
<tr>
<td>IloEnableNANDetection</td>
<td>150</td>
</tr>
<tr>
<td>IloEndMT</td>
<td>151</td>
</tr>
<tr>
<td>IloEnv</td>
<td>152</td>
</tr>
<tr>
<td>IloEnvironmentMismatch</td>
<td>157</td>
</tr>
<tr>
<td>IloEqIntersection</td>
<td>158</td>
</tr>
<tr>
<td>IloEqPartition</td>
<td>159</td>
</tr>
<tr>
<td>IloEqUnion</td>
<td>160</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloException</td>
<td>161</td>
</tr>
<tr>
<td>IloExponent</td>
<td>163</td>
</tr>
<tr>
<td>IloExpr</td>
<td>164</td>
</tr>
<tr>
<td>IloExprArray</td>
<td>171</td>
</tr>
<tr>
<td>IloExpr::LinearIterator</td>
<td>173</td>
</tr>
<tr>
<td>IloExtractable</td>
<td>174</td>
</tr>
<tr>
<td>IloExtractableArray</td>
<td>177</td>
</tr>
<tr>
<td>IloFastMutex</td>
<td>180</td>
</tr>
<tr>
<td>IloFloatArray</td>
<td>184</td>
</tr>
<tr>
<td>IloFloatVar</td>
<td>185</td>
</tr>
<tr>
<td>IloFloatVarArray</td>
<td>186</td>
</tr>
<tr>
<td>IloFloor</td>
<td>187</td>
</tr>
<tr>
<td>IloFunction</td>
<td>188</td>
</tr>
<tr>
<td>IloGetClone</td>
<td>189</td>
</tr>
<tr>
<td>IloHalfPi</td>
<td>190</td>
</tr>
<tr>
<td>IloIfThen</td>
<td>191</td>
</tr>
<tr>
<td>IloInfinity</td>
<td>194</td>
</tr>
<tr>
<td>IloInitMT</td>
<td>195</td>
</tr>
<tr>
<td>IloInt</td>
<td>196</td>
</tr>
<tr>
<td>IloIntArray</td>
<td>197</td>
</tr>
<tr>
<td>IloIntBinaryPredicate</td>
<td>199</td>
</tr>
<tr>
<td>IloIntExpr</td>
<td>201</td>
</tr>
<tr>
<td>IloIntExprArg</td>
<td>204</td>
</tr>
<tr>
<td>IloIntExprArray</td>
<td>206</td>
</tr>
<tr>
<td>IloIntSet</td>
<td>209</td>
</tr>
<tr>
<td>IloIntSet::Iterator</td>
<td>213</td>
</tr>
<tr>
<td>IloIntSetVar</td>
<td>214</td>
</tr>
<tr>
<td>IloIntSetVarArray</td>
<td>218</td>
</tr>
<tr>
<td>IloIntTernaryPredicate</td>
<td>221</td>
</tr>
<tr>
<td>IloIntTupleSet</td>
<td>223</td>
</tr>
<tr>
<td>IloIntTupleSetIterator</td>
<td>226</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IloIntVar</td>
<td>228</td>
</tr>
<tr>
<td>IloIntVarArray</td>
<td>233</td>
</tr>
<tr>
<td>IloInverse</td>
<td>237</td>
</tr>
<tr>
<td>IloIsNAN</td>
<td>240</td>
</tr>
<tr>
<td>IloIterator</td>
<td>241</td>
</tr>
<tr>
<td>IloLog</td>
<td>243</td>
</tr>
<tr>
<td>IloMax</td>
<td>244</td>
</tr>
<tr>
<td>IloMaximize</td>
<td>245</td>
</tr>
<tr>
<td>IloMember</td>
<td>246</td>
</tr>
<tr>
<td>IloMin</td>
<td>247</td>
</tr>
<tr>
<td>IloMinimize</td>
<td>248</td>
</tr>
<tr>
<td>IloModel</td>
<td>249</td>
</tr>
<tr>
<td>IloModel::Iterator</td>
<td>252</td>
</tr>
<tr>
<td>IloMonotonicDecreasingNumExpr</td>
<td>254</td>
</tr>
<tr>
<td>IloMonotonicIncreasingNumExpr</td>
<td>255</td>
</tr>
<tr>
<td>IloMutexDeadlock</td>
<td>256</td>
</tr>
<tr>
<td>IloMutexNotOwner</td>
<td>257</td>
</tr>
<tr>
<td>IloMutexProblem</td>
<td>258</td>
</tr>
<tr>
<td>IloNot</td>
<td>260</td>
</tr>
<tr>
<td>IloNotMember</td>
<td>262</td>
</tr>
<tr>
<td>IloNullIntersect</td>
<td>263</td>
</tr>
<tr>
<td>IloNum</td>
<td>264</td>
</tr>
<tr>
<td>IloNumArray</td>
<td>265</td>
</tr>
<tr>
<td>IloNumColumn</td>
<td>267</td>
</tr>
<tr>
<td>IloNumColumnArray</td>
<td>270</td>
</tr>
<tr>
<td>IloNumExpr</td>
<td>272</td>
</tr>
<tr>
<td>IloNumExprArg</td>
<td>275</td>
</tr>
<tr>
<td>IloNumExprArray</td>
<td>277</td>
</tr>
<tr>
<td>IloNumExpr::NonLinearExpression</td>
<td>279</td>
</tr>
<tr>
<td>IloNumFunction</td>
<td>280</td>
</tr>
<tr>
<td>IloNumVar</td>
<td>281</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IloNumVarArray</td>
<td>287</td>
</tr>
<tr>
<td>IloNumVar::Type</td>
<td>292</td>
</tr>
<tr>
<td>IloObjective</td>
<td>293</td>
</tr>
<tr>
<td>IloObjective::Sense</td>
<td>299</td>
</tr>
<tr>
<td>IloOr</td>
<td>300</td>
</tr>
<tr>
<td>IloPack</td>
<td>304</td>
</tr>
<tr>
<td>IloPartition</td>
<td>307</td>
</tr>
<tr>
<td>IloPathLength</td>
<td>308</td>
</tr>
<tr>
<td>IloPathTransitFunction</td>
<td>313</td>
</tr>
<tr>
<td>IloPathTransIt</td>
<td>314</td>
</tr>
<tr>
<td>IloPi</td>
<td>316</td>
</tr>
<tr>
<td>IloPiecewiseLinear</td>
<td>317</td>
</tr>
<tr>
<td>IloPower</td>
<td>318</td>
</tr>
<tr>
<td>IloQuarterPi</td>
<td>319</td>
</tr>
<tr>
<td>IloRandom</td>
<td>320</td>
</tr>
<tr>
<td>IloRange</td>
<td>323</td>
</tr>
<tr>
<td>IloRangeArray</td>
<td>330</td>
</tr>
<tr>
<td>IloRound</td>
<td>337</td>
</tr>
<tr>
<td>IloSOS1</td>
<td>338</td>
</tr>
<tr>
<td>IloSOS1Array</td>
<td>341</td>
</tr>
<tr>
<td>IloSOS2</td>
<td>344</td>
</tr>
<tr>
<td>IloSOS2Array</td>
<td>347</td>
</tr>
<tr>
<td>IloScalProd</td>
<td>350</td>
</tr>
<tr>
<td>IloScalProd</td>
<td>351</td>
</tr>
<tr>
<td>IloScalProd</td>
<td>352</td>
</tr>
<tr>
<td>IloScalProd.</td>
<td>351</td>
</tr>
<tr>
<td>IloSemaphore</td>
<td>354</td>
</tr>
<tr>
<td>IloSemiContVar</td>
<td>357</td>
</tr>
<tr>
<td>IloSemiContVarArray</td>
<td>361</td>
</tr>
<tr>
<td>IloSequence</td>
<td>365</td>
</tr>
<tr>
<td>IloSolution</td>
<td>368</td>
</tr>
</tbody>
</table>
# Table of Contents

- IloSolutionArray ................................................................. 379
- IloSolution::Iterator ............................................................. 380
- IloSolutionIterator ............................................................... 382
- IloSolutionManip ................................................................. 384
- IloSquare ........................................................................... 385
- IloSubset ............................................................................. 386
- IloSubsetEq .......................................................................... 387
- IloSum .................................................................................. 388
- IloTableConstraint ................................................................. 389
- IloThreeHalfPi ....................................................................... 391
- IloTimer ................................................................................ 392
- IloTwoPi ............................................................................... 394
- IloTypedList .......................................................................... 395
- IloTypedListManager ............................................................... 396
- operator && ................................................................. 397
- operator * ........................................................................... 398
- operator new ................................................................. 399
- operator! ........................................................................... 400
- operator== .......................................................................... 401
- operator+ ........................................................................... 402
- operator- ........................................................................... 403
- operator/ ........................................................................... 404
- operator< ........................................................................... 405
- operator<< .......................................................................... 406
- operator<= ........................................................................... 407
- operator== .......................................................................... 408
- operator> ........................................................................... 409
- operator>= .......................................................................... 410
- operator|> .......................................................................... 411
- operator|| .......................................................................... 412

**Group optim.concert.extensions** ............................................. 412
# Table of Contents

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloColumnHeaderNotFoundException</td>
<td>414</td>
</tr>
<tr>
<td>IloCsvLine</td>
<td>415</td>
</tr>
<tr>
<td>IloCsvReader</td>
<td>419</td>
</tr>
<tr>
<td>IloCsvReader::LineIterator</td>
<td>427</td>
</tr>
<tr>
<td>IloCsvReaderParameterException</td>
<td>430</td>
</tr>
<tr>
<td>IloCsvTableReader</td>
<td>432</td>
</tr>
<tr>
<td>IloCsvTableReader::LineIterator</td>
<td>436</td>
</tr>
<tr>
<td>IloDifference</td>
<td>438</td>
</tr>
<tr>
<td>IloIntervalList</td>
<td>445</td>
</tr>
<tr>
<td>IloIntervalListCursor</td>
<td>450</td>
</tr>
<tr>
<td>IloLineNotFoundException</td>
<td>453</td>
</tr>
<tr>
<td>IloMax</td>
<td>454</td>
</tr>
<tr>
<td>IloMin</td>
<td>455</td>
</tr>
<tr>
<td>IloNumToAnySetStepFunction</td>
<td>456</td>
</tr>
<tr>
<td>IloNumToAnySetStepFunctionCursor</td>
<td>465</td>
</tr>
<tr>
<td>IloNumToNumSegmentFunction</td>
<td>468</td>
</tr>
<tr>
<td>IloNumToNumSegmentFunctionCursor</td>
<td>475</td>
</tr>
<tr>
<td>IloNumToNumStepFunction</td>
<td>478</td>
</tr>
<tr>
<td>IloNumToNumStepFunctionCursor</td>
<td>484</td>
</tr>
<tr>
<td>IloTableNotFoundException</td>
<td>487</td>
</tr>
<tr>
<td>IloUnion</td>
<td>488</td>
</tr>
<tr>
<td>IloUnion::operator *</td>
<td>489</td>
</tr>
<tr>
<td>operator +</td>
<td>490</td>
</tr>
<tr>
<td>operator+</td>
<td>491</td>
</tr>
<tr>
<td>Operator</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>operator</td>
<td>492</td>
</tr>
<tr>
<td>operator&lt;&lt;</td>
<td>493</td>
</tr>
<tr>
<td>operator==</td>
<td>494</td>
</tr>
<tr>
<td>operator==</td>
<td>495</td>
</tr>
<tr>
<td>operator==</td>
<td>496</td>
</tr>
</tbody>
</table>

**Group optim.concert.xml** ........................................ 496

- IloXmlContext ........................................... 497
- IloXmlInfo .............................................. 503
- IloXmlReader ............................................ 513
- IloXmlWriter ............................................. 519
Welcome to Concert Technology

This reference manual documents the 2.0 beta release of ILOG Concert Technology.

What Is Concert Technology?

Concert Technology offers a C++ library of classes and functions that enable you to design models of problems for both math programming (including linear programming, mixed integer programming, quadratic programming, and network programming) and constraint programming solutions.

This library is not a new programming language: it lets you use data structures and control structures provided by C++. Thus, the Concert Technology part of an application can be completely integrated with the rest of that application (for example, the graphic interface, connections to databases, etc.) because it can share the same objects.

Furthermore, you can use the same objects to model your problem whether you choose a constraint programming or math programming approach. In fact, Concert Technology enables you to combine these technologies simultaneously.

What You Need to Know

This manual assumes that you are familiar with the operating system where you are using Concert Technology. Since Concert Technology is written for C++ developers, this manual assumes that you can write C++ code and that you have a working knowledge of your C++ development environment.

Notation

Throughout this manual, the following typographic conventions apply:

◆ Samples of code are written in this **typeface**.

◆ The names of constructors and member functions appear in this **typeface** in the section where they are documented.

◆ Important ideas are emphasized like this.


Naming Conventions

The names of types, classes, and functions defined in the Concert Technology library begin with \texttt{Ilo}.

The names of classes are written as concatenated, capitalized words. For example, \texttt{IloNumVar}

A lower case letter begins the first word in names of arguments, instances, and member functions. Other words in such a name begin with a capital letter. For example, \texttt{IloNumVarArray::setBounds}

There are no public data members in Concert Technology.

Accessors begin with the keyword \texttt{get} followed by the name of the data member. Accessors for Boolean members begin with \texttt{is} followed by the name of the data member. Like other member functions, the first word in such a name begins with a lower case letter, and any other words in the name begin with a capital letter.

Modifiers begin with the keyword \texttt{set} followed by the name of the data member.

Related Documents

The Concert Technology 2.0 library comes with the following documentation. The online documentation, in HTML format, may be accessed through standard HTML browsers.

◆ The Reference Manual documents the predefined C++ classes, global functions, type definitions, and macros in the libraries. It also provides formal explanations of certain concepts, such as arrays, handles, notification, and column-wise modeling.

◆ The README file, delivered in the standard distribution, contains the most current information about platform prerequisites for Concert Technology.

◆ Source code for examples is located in the examples directory in the standard distribution.

For More Information

ILOG offers technical support, users' mailing lists, and comprehensive websites for its products, including Concert Technology, ILOG CPLEX, and ILOG Solver.

Technical Support

For technical support of Concert Technology, you should contact your local distributor, or, if you are a direct ILOG customer, contact the technical support center listed for your...
licensed ILOG product, whether CPLEX or Solver. We encourage you to use e-mail for faster, better service.

**Users' Mailing List**

The electronic mailing list `solver-list@ilog.fr` is available for you to share your development experience with other Concert Technology users. This list is not moderated, but subscription is subject to an ongoing maintenance contract. To subscribe to `solver-list`, send e-mail without any subject to `solver-list-owner@ilog.fr`, with the following contents:

```
subscribe solver-list
your e-mail address (if different from the From field)
first name, last name
your location (company and country)
maintenance contract number
maintenance contract owner's last name
```

**Web Sites**

There are two kinds of web pages available to users of Concert Technology: web pages restricted to owners of a paid maintenance contract; web pages freely available to all.

**Web Pages for a Paid Maintenance Contract**

The technical support pages on our world wide web sites contain FAQ (Frequently Asked/Answered Questions) and the latest patches for some of our products. Changes are posted in the product mailing list. Access to these pages is restricted to owners of an ongoing maintenance contract. The maintenance contract number and the name of the person this contract is sent to in your company will be needed for access, as explained on the login page.

All three of these sites contain the same information, but access is localized, so we recommend that you connect to the site corresponding to your location, and select the Services page from the home page.

- Americas: [http://www.ilog.com](http://www.ilog.com)
- Europe, Africa, and Middle East: [http://www.ilog.fr](http://www.ilog.fr)

**Web Pages for General Information**

In addition to those web pages for technical support of a paid maintenance contract, you will find other web pages containing additional information about Concert Technology, including technical papers that have also appeared at industrial and academic
conferences, models developed by ILOG and its customers, news about progress in optimization. This freely available information is located at these localized web sites:

◆ http://www.ilog.com/products/optimization/
◆ http://www.ilog.fr/products/optimization/
Arrays

For most basic classes (such as IloNumVar or IloConstraint) in Concert Technology, there is also a corresponding class of arrays where the elements of the array are instances of that basic class. For example, elements of an instance of IloConstraintArray are instances of the class IloConstraint.

Arrays in an Environment

Every array must belong to an environment (an instance of IloEnv). In other words, when you create a Concert Technology array, you pass an instance of IloEnv as a parameter to the constructor. All the elements of a given array must belong to the same environment.

Extensible Arrays

Concert Technology arrays are extensible. That is, you can add elements to the array dynamically. You add elements by means of the add member function of the array class.

You can also remove elements from an array by means of its remove member function. References to an array change whenever an element is added to or removed from the array.

Arrays as Handles

Like other Concert Technology objects, arrays are implemented by means of two classes: a handle class corresponding to an implementation class. An object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of the corresponding implementation class. As a Concert Technology user, you will be working primarily with handles.

Copying Arrays

Many handles may point to the same implementation object. This principle holds true for arrays as well as other handle classes in Concert Technology. When you want to create more than one handle for the same implementation object, you should use either the copy constructor or the assignment operator of the array class. For example,
IloNumArray array(env); // creates a handle pointing to new impl
IloNumArray array1(array); // creates a handle pointing to same impl
IloNumArray array2;      // creates an empty handle
array2 = array;          // sets impl of handle array2 to impl of array

Programming Hint: Using Arrays Efficiently

If your application only reads an array (that is, if your function does not modify an
element of the array), then we recommend that you pass the array to your function as a
const parameter. This practice forces Concert Technology to access the const
conversion of the index operator (that is, operator[]), which is faster.

Assert and NDEBUG

Most member functions of classes in Concert Technology are inline functions that
contain an assert statement. This statement asserts that the invoking object and the
member function parameters are consistent; in some member functions, the assert
statement checks that the handle pointer is non-null. These statements can be suppressed by
the macro NDEBUG. This option usually reduces execution time. The price you pay
for this choice is that attempts to access through null pointers are not trapped and usually
result in memory faults.

Compilation with assert statements will not prevent core dumps by incorrect code. Instead, compilation with assert statements moves the execution of the incorrect code (the core dump, for example) to a place where you can see what is causing the problem in a source code debugger. Correctly written code will never cause one of these Concert Technology assert statements to fail.

Column-Wise Modeling

Concert Technology supports column-wise modeling, a technique widely used in the math programming and operations research communities to build a model column by column. In Concert Technology, creating a new column is comparable to creating a new variable and adding it to a set of constraints. You use an instance of IloNumColumn to do so. An instance of IloNumColumn allows you to specify to which constraints or other extractable objects Concert Technology should add the new variable along with its data. For example, in a linear programming problem (an LP), if the new variable will appear in some linear constraints as ranges (instances of IloRange), you need to specify the list of such constraints along with the non-zero coefficients (a value of IloNum) for each of them.
You then create a new column in your model when you create a new variable with an instance of `IloNumColumn` as its parameter. When you create the new variable, Concert Technology will add it along with appropriate parameters to all the extractable objects you have specified in the instance of `IloNumColumn`.

Instead of building an instance of `IloNumColumn`, as an alternative, you can use a column expression directly in the constructor of the variable. You can also use instances of `IloNumColumn` within column expressions.

The following undocumented classes provide the underlying mechanism for column-wise modeling:

- `IloAddValueToObj`
- `IloAddValueToRange`

The following operators are useful in column-wise modeling:

- in the class `IloRange`,
  `IloAddValueToRange operator() (IloNum value);`
- in the class `IloObjective`,
  `IloAddValueToObj operator () (IloNum value);`

That is, the `operator()` in extractable classes, such as `IloRange` or `IloObjective`, creates descriptors of how Concert Technology should add the new, yet-to-be-created variable to the invoking extractable object.

You can use the `operator +` to link together the objects returned by `operator()` to form a column. You can then use an instance of `IloNumColumn` to build up column expressions within a programming loop and thus save them for later use or to pass them to functions.

Here is how to use an instance of `IloNumColumn` with operators from `IloRange` and `IloObjective` to create a column with a coefficient of 2 in the objective, with 10 in range1, and with 3 in range2. The example then uses that column when it creates newvar1, and it uses column expressions when it creates newvar2 and newvar3.

```plaintext
IloNumColumn col = obj(2) + range1(10) + range2(3);
IloNumVar newvar1(col);
IloNumVar newvar2(col + range3(17));
IloNumVar newvar3(range1(1) + range3(3));
```

In other words, given an instance `obj` of `IloObjective` and the instances `range1`, `range2`, and `range3` of `IloRange`, those lines create the new variables `newvar1`, `newvar2`, and `newvar3`. 
newvar2, and newvar3 and add them as linear terms to obj, range1, and range3 in the following way:

\[
\begin{align*}
\text{obj: } & + 2 \times \text{newvar1} + 2 \times \text{newvar2} \\
\text{range1: } & + 10 \times \text{newvar1} + 10 \times \text{newvar2} + 1 \times \text{newvar3} \\
\text{range2: } & + 3 \times \text{newvar1} + 3 \times \text{newvar2} \\
\text{range3: } & + 17 \times \text{newvar2} + 3 \times \text{newvar3}
\end{align*}
\]

For more information, refer to the documentation of IloNumColumn, IloObjective, and IloRange.

---

### Exceptions, Errors

An exception is thrown; it is not allocated in a Concert Technology environment; it is not allocated on the C++ heap. It is not necessary for you as a programmer to delete an exception explicitly. Instead, the system calls the constructor of the exception to create it, and the system calls the destructor of the exception to delete it.

When exceptions are enabled on a platform that supports C++ exceptions, an instance of a class of Concert Technology is able to throw an exception in case of error. On platforms that do not support C++ exceptions, it is possible for Concert Technology to exit in case of error.

**Programming Hint: Throwing and Catching Exceptions**

Exceptions are thrown by value. They are not allocated on the C++ heap, nor in a Concert Technology environment. The correct way to catch an exception is to catch a reference to the error (indicated by the ampersand &), like this:

```cpp
catch(IloException& oops);
```

---

### Extraction

Concert Technology offers classes for you to design a model of your problem. You can then invoke an algorithm to extract information from your model to solve the problem. In this context, an algorithm is an instance of a class such as IloCplex, documented in the ILOG CPLEX Reference Manual, or IloSolver, documented in the ILOG Solver Reference Manual.

For details about what each algorithm extracts from a model, see the reference manual documenting that algorithm. For example, the ILOG CPLEX Reference Manual lists precisely which classes of Concert Technology are extracted by an instance of IloCplex. In general terms, an instance of IloCplex extracts a model as rows and columns, where the columns indicate decision variables of the model. Also in general terms, an instance of IloSolver extracts an instance of a class whose name begins
Ilo to a corresponding instance of a class whose name begins Ilc. For example, an instance of IloAllDiff is extracted by IloSolver as an instance of IlcAllDiff.

**Handle Class**

Most Concert Technology entities are implemented by means of two classes: a handle class and an implementation class, where an object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of the corresponding implementation class. As a Concert Technology user, you will be working primarily with handles.

As handles, these objects should be passed in either of these ways:

- as const by value (when no change is involved);
- by reference (when the function to which the handle is passed changes the implementation of that handle).

They should be created as automatic objects, where "automatic" has the usual C++ meaning.

Member functions of a handle class correspond to member functions of the same name in the implementation class.

**Lazy Copy**

Concert Technology makes a lazy copy when you use any of the following objects inside a predefined Concert Technology object:

- an expression (an instance of IloExpr or one of its subclasses),
- an array (such as an instance of IloNumArray),
- a column (an instance of IloNumColumn),
- or a set (such as an instance of IloIntSet).

That is, a physical copy of those objects is created only when needed.

In Concert Technology, expressions, arrays, columns, and sets are implemented by handle classes and corresponding implementation classes. One or more handles may point to the same implementation object. For example, many instances of the handle class IloNumVarArray may point to the same implementation object.

A handle may be empty; that is, it may point to 0 (zero). You can test whether a handle is empty by means of the member function handle.getImpl. If that member function returns 0, the handle points to a null implementation.
When you modify an expression, an array, a column, or a set that has been used in a Concert Technology object, Concert Technology considers whether the handle you are modifying is the sole reference to the corresponding implementation object. If so, Concert Technology simply makes the modification.

In contrast, if the handle you are modifying points to an implementation object that is used by other objects predefined in Concert Technology, Concert Technology first copies the implementation object for the handle you are modifying and then makes the modification. The other handles pointing to the original implementation object remain unchanged and your modification has no impact on them.

**Examples:**

Here is an example illustrating lazy copy of variables:

```cpp
IloNumVar a1(env, 0, 10);
IloNumVar a2(env, 0, 10);
IloNumVar a3(env, 0, 0);
IloNumVarArray A(env, 2, a1, a2);
IloConstraint ct = IloAllDiff(env, A);
A.add (a3);
```

Because of the lazy copy, even though `a3` was added to `A`, `ct` uses only `a1` and `a2`.

---

**Normalization: Reducing Linear Terms**

*Normalizing* is sometimes known as *reducing the terms* of a linear expression.

Linear expressions consist of terms made up of constants and variables related by arithmetic operations; for example, `x + 3y` is a linear expression of two terms consisting of two variables. In some expressions, a given variable may appear in more than one term, for example, `x + 3y + 2x`. Concert Technology has more than one way of dealing with linear expressions in this respect, and you control which way Concert Technology treats expressions from your application.

In one mode, Concert Technology analyzes linear expressions that your application passes it and attempts to reduce them so that a given variable appears in only one term in the linear expression. This is the default mode. You set this mode with the member function `IloEnv::setNormalizer(IloTrue)`.

In the other mode, Concert Technology assumes that no variable appears in more than one term in any of the linear expressions that your application passes to Concert Technology. We call this mode assume normalized linear expressions. You set this mode with the member function `IloEnv::setNormalizer(IloFalse)`.

In classes such as `IloExpr` or `IloRange`, there are member functions that check the setting of the member function `IloEnv::setNormalizer` in the environment and
behave accordingly. The documentation of those member functions indicates how they behave with respect to normalization.

When you set `IloEnv::setNormalizer(IloFalse)`, those member functions assume that no variable appears in more than one term in a linear expression. This mode may save time during computation, but it entails the risk that a linear expression may contain one or more variables, each of which appears in one or more terms. Such a case may cause certain assertions in member functions of a class to fail if you do not compile with the flag `-DNDEBUG`.

By default, those member functions attempt to reduce expressions. This mode may require more time during preliminary computation, but it avoids the possibility of a failed assertion in case of duplicates.

For more information, refer to the documentation of `IloEnv`, `IloExpr`, and `IloRange`.

**Notification**

You may modify the elements of a model in Concert Technology. For example, you may add or remove constraints, change the objective, add or remove columns, add or remove rows, and so forth.

In order to maintain consistency between a model and the algorithms that may use it, Concert Technology notifies algorithms about changes in the objects that the algorithms have extracted. In this manual, member functions that are part of this notification system are indicated like this:

```
Note: This member function notifies Concert Technology algorithms about this change of this invoking object.
```

**Deletion of Extractables**

As a modeling layer, Concert allows the creation and destruction of extractables. This is accessible through the method `IloExtractable::end()` and `IloExtractableArray::endElements()` method. The goal of these methods is to reclaim memory associated with the deleted objects while maintaining the safest possible Concert environment. In this context, a safe Concert environment is defined by the property that no object points to a deleted object; this is referred to as a dangling pointer in C++.

There exist two paradigms to ensure the safeness of the delete operation. The first, linear mode, comes from math programming and is possible only on extractables and
objects used in linear programming. The second, safe generic mode, is more strict and is valid on all Concert extractables.

You can access both paradigms by calling `IloEnv::setDeleter(IloDeleterMode mode)`, where mode may be `IloLinearDeleterMode` or `IloSafeDeleterMode`.

**Linear Mode**

To use linear mode, you must either

- call `IloEnv::setDeleter(IloLinearDeleterMode)`, or
- refrain from calling `IloEnv::setDeleter()`, as it is the default mode.

In linear mode, the following behavior is implemented:

- If a range constraint is deleted, it is removed from the models that contain it.
- If a variable is deleted, its coefficient is set to 0 in the ranges, expressions, and objectives where it appears. The variable is removed from the `SOS1`, `SOS2`, and `IloConversion` where it appears.

**Example**

This example tests the linear mode deletion of a variable x.

```cpp
void TestLinearDeleter() {
  IloEnv env;
  env.out() << "TestLinearDeleter" << endl;
  try {
    IloModel model(env);
    IloNumVar x(env, 0, 10, "x");
    IloNumVar y(env, 0, 10, "y");
    IloConstraint con = (x + y <= 0);
    IloConstraint con2 = y >= 6;
    IloNumVarArray ar(env, 2, x, y);
    IloSOS1 sos(env, ar, "sos");
    model.add(con);
    model.add(con2);
    model.add(sos);
    env.out() << "Before Delete" << endl;
    env.out() << model << endl;
    x.end();
    con2.end();
    env.out() << "After Delete" << endl;
    env.out() << model << endl;
  } catch (IloException & e) {
    cout << "Error : " << e << endl;
  }
  env.end();
}
```
The example produces the following output:

```
TestLinearDeleter
Before Delete
IloModel model10 = {
    IloRange rng3( 1 * x + 1 * y ) <= 0
    IloRange rng46 <=( 1 * y )
    IloSOS1I (sos)
        _varArray [x(F)[0..10], y(F)[0..10]]
        _valArray []
}

After Delete
IloModel model10 = {
    IloRange rng3( 1 * y ) <= 0
    IloSOS1I (sos)
        _varArray [y(F)[0..10]]
        _valArray []
}
```

Safe Generic Mode

To use safe generic mode, you must:

◆ call `IloEnv::setDeleter(IloSafeDeleterMode)` and
◆ add `#include <ilconcert/ilodeleter.h>` to your program.

In this mode, the environment builds a dependency graph between all extractables. This graph contains all extractables created

◆ after a call to `IloEnv::setDeleter(IloSafeDeleterMode)` and
◆ before a call to `IloEnv::unsetDeleter()`.

Objects not managed by this dependency graph are referred to here as "nondeletable". An attempt to delete a nondeletable object will throw an exception.

We recommended that you create this graph just after the creation of the environment and that you refrain from using `IloEnv::unsetDeleter`. We make these recommendations because building an incomplete dependency graph is very error prone and should only be attempted by advanced users. A good example of this incomplete graph is the separation of a model between a nondeletable base model and deletable extensions of this base model.
Calling `IloExtractable::end()` on extractable `xi` will succeed only if no other extractable uses extractable `xi`. If this is not the case, a call to `IloExtractable::end()` will throw an exception `IloDeleter::RequiresAnotherDeletionException` indicating which extractable uses the extractable that you want to delete.

**Example**

This example shows an attempt to delete one extractable that is used by another.

```cpp
void TestSafeDeleter() {
    IloEnv env;
    env.out() << "TestSafeDeleter" << endl;
    env.setDeleter(IloSafeDeleterMode);
    try {
        IloModel model(env);
        IloNumVar x(env, 0, 10);
        IloNumVar y(env, 0, 10);
        IloConstraint con = (x + y <= 0);
        try {
            x.end();
        } catch (IloDeleter::RequiresAnotherDeletionException &e) {
            cout << "Caught " << e << endl;
            e getUsers()[0].end();
            e.end();
        }
        x.end();
    } catch (IloException & e) {
        cout << "Error : " << e << endl;
    }
    env.unsetDeleter();
    env.end();
}
```

The example produces the following output:

```
TestSafeDeleter
Caught You cannot end x1(F)[0..10] before IloRange rng3( 1 * x1 + 1 * x2 ) <= 0
```

To address this, you should use the `IloExtractableArray::endElements()` method. With this method, all extractables appearing in the array are deleted one after another. Thus, if an extractable is used by another extractable and this other extractable is deleted before the first one, the system will not complain and will not throw an exception.
Example

This example illustrates the use of the endElements() method

```cpp
void TestSafeDeleterWithArray() {
  IloEnv env;
  env.out() << "TestSafeDeleterWithArray" << endl;
  env.setDeleter(IloSafeDeleterMode);
  try {
    IloModel model(env);
    IloNumVar x(env, 0, 10);
    IloNumVar y(env, 0, 10);
    IloConstraint con = (x + y <= 0);
    IloExtractableArray ar(env, 2, con, x);
    ar.endElements();
  } catch (IloException& e) {
    cout << "Error : " << e << endl;
  }
  env.unsetDeleter();
  env.end();
}
```

The example will not throw an exception.

Note: Please note that in this last example, the constraint con must appear before the variable x as it will be deleted before the variable x.

Obsolete Functions & Classes

<table>
<thead>
<tr>
<th>Obsolete Function or Class</th>
<th>Replaced By</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloNumSet</td>
<td>IloIntSet</td>
</tr>
<tr>
<td>IloNumSetVar</td>
<td>IloIntSetVar</td>
</tr>
<tr>
<td>IloNumSetVarArray</td>
<td>IloIntSetVar Array</td>
</tr>
<tr>
<td>IloExprBase</td>
<td>IloNumExprArg and IloIntExprArg</td>
</tr>
<tr>
<td>IloExprNode</td>
<td>IloNumExprArg and IloIntExprArg</td>
</tr>
<tr>
<td>IloExprArg</td>
<td>IloNumExprArg and IloIntExprArg</td>
</tr>
<tr>
<td>IloExprI</td>
<td>IloNumLinTermI (undocumented)</td>
</tr>
</tbody>
</table>
Group optim.concert

The ILOG Concert API.

<p>| Classes Summary |
|------------------|---------------------------------------------------------------|
| IloAlgorithm     | The base class of algorithms in Concert Technology.          |
| IloAlgorithm::CannotExtractException | The class of exceptions thrown if an object cannot be extracted from a model. |
| IloAlgorithm::CannotRemoveException | The class of exceptions thrown if an object cannot be removed from a model. |
| IloAlgorithm::Exception | The base class of exceptions thrown by classes derived from IloAlgorithm. |
| IloAlgorithm::NotExtractedException | The class of exceptions thrown if an extractable object has no value in the current solution of an algorithm. |
| IloAllDiff       | A constraint that forces constrained integer variables to assume different values in a model. |
| IloAllMinDistance | A constraint on the minimum absolute distance between a pair of variables in an array. |
| IloAnd           | A conjunctive constraint that defines a logical conjunctive-AND among other constraints. |
| IloAnyArray      | The array class of the enumerated type definition IloAny.     |
| IloAnyBinaryPredicate | A class to define binary predicates on objects in a model. |
| IloAnySet        | A class whose instances represent a set of enumeration values. |
| IloAnySet::Iterator | An iterator to traverse the elements of IloAnySet. |
| IloAnySetVar     | Class to represent a set of enumerated values as a constrained variable. |</p>
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloAnySetVarArray</td>
<td>The array class of the set variable class IloAnySetVar.</td>
</tr>
<tr>
<td>IloAnyTernaryPredicate</td>
<td>A class to define ternary predicates on objects in a model.</td>
</tr>
<tr>
<td>IloAnyTupleSet</td>
<td>A class to represent set of enumerated tuples in a model.</td>
</tr>
<tr>
<td>IloAnyTupleSetIterator</td>
<td>An iterator to traverse the elements of a finite set of tuples of enumerated values.</td>
</tr>
<tr>
<td>IloAnyVar</td>
<td>A class to represent an enumerated variable.</td>
</tr>
<tr>
<td>IloAnyVarArray</td>
<td>A class to represent an array of enumerated variables.</td>
</tr>
<tr>
<td>IloArray</td>
<td>A template to create classes of arrays for elements of a given class.</td>
</tr>
<tr>
<td>IloBarrier</td>
<td>A system class to synchronize threads at a specified number.</td>
</tr>
<tr>
<td>IloBaseEnvMutex</td>
<td>A class to initialize multithreading in an application.</td>
</tr>
<tr>
<td>IloBoolArray</td>
<td></td>
</tr>
<tr>
<td>IloBoolVar</td>
<td></td>
</tr>
<tr>
<td>IloBoolVarArray</td>
<td></td>
</tr>
<tr>
<td>IloBox</td>
<td>A class of multidimensional boxes for multidimensional placement problems.</td>
</tr>
<tr>
<td>IloCondition</td>
<td></td>
</tr>
<tr>
<td>IloConstraint</td>
<td></td>
</tr>
<tr>
<td>IloConstraintArray</td>
<td></td>
</tr>
<tr>
<td>IloConversion</td>
<td></td>
</tr>
<tr>
<td>IloDiff</td>
<td></td>
</tr>
<tr>
<td>IloDistribute</td>
<td></td>
</tr>
<tr>
<td>IloEmptyHandleException</td>
<td>The class of exceptions thrown if an empty handle is passed.</td>
</tr>
<tr>
<td>IloEnv</td>
<td>The class of environments for models or algorithms in Concert Technology.</td>
</tr>
<tr>
<td>IloEnvironmentMismatch</td>
<td></td>
</tr>
<tr>
<td>IloException</td>
<td></td>
</tr>
<tr>
<td>IloExpr</td>
<td></td>
</tr>
<tr>
<td>IloExprArray</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IloExpr::LinearIterator</td>
<td>An iterator over the linear part of an expression.</td>
</tr>
<tr>
<td>IloExtractable</td>
<td></td>
</tr>
<tr>
<td>IloExtractableArray</td>
<td></td>
</tr>
<tr>
<td>IloFastMutex</td>
<td></td>
</tr>
<tr>
<td>IloFunction</td>
<td></td>
</tr>
<tr>
<td>IloIfThen</td>
<td></td>
</tr>
<tr>
<td>IloIntArray</td>
<td></td>
</tr>
<tr>
<td>IloIntBinaryPredicate</td>
<td></td>
</tr>
<tr>
<td>IloIntExpr</td>
<td>The class of integer expressions in Concert Technology 2.0.</td>
</tr>
<tr>
<td>IloIntExprArg</td>
<td>A class used internally in Concert Technology.</td>
</tr>
<tr>
<td>IloIntExprArray</td>
<td>The array class of IloIntExpr.</td>
</tr>
<tr>
<td>IloIntSet</td>
<td></td>
</tr>
<tr>
<td>IloIntSet::Iterator</td>
<td></td>
</tr>
<tr>
<td>IloIntSetVar</td>
<td>The class IloIntSetVar.</td>
</tr>
<tr>
<td>IloIntSetVarArray</td>
<td>The array class of the set variable class for integer values.</td>
</tr>
<tr>
<td>IloIntTernaryPredicate</td>
<td></td>
</tr>
<tr>
<td>IloIntTupleSet</td>
<td></td>
</tr>
<tr>
<td>IloIntTupleSetIterator</td>
<td></td>
</tr>
<tr>
<td>IloIntVar</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray</td>
<td>The array class of IloIntVar.</td>
</tr>
<tr>
<td>IloInverse</td>
<td></td>
</tr>
<tr>
<td>IloIterator</td>
<td>A template to create iterators for a class of extractable objects.</td>
</tr>
<tr>
<td>IloModel</td>
<td></td>
</tr>
<tr>
<td>IloModel::Iterator</td>
<td></td>
</tr>
<tr>
<td>IloMutexDeadlock</td>
<td>The class of exceptions thrown due to mutex deadlock.</td>
</tr>
<tr>
<td>IloMutexNotOwner</td>
<td>The class of exceptions thrown.</td>
</tr>
<tr>
<td>IloMutexProblem</td>
<td></td>
</tr>
<tr>
<td>IloNot</td>
<td></td>
</tr>
<tr>
<td>IloNumArray</td>
<td></td>
</tr>
<tr>
<td>IloNumColumn</td>
<td></td>
</tr>
<tr>
<td>IloNumColumnArray</td>
<td></td>
</tr>
<tr>
<td>IloNumExpr</td>
<td>The class of numeric expressions in a Concert model.</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IloNumExprArg</td>
<td>A class used internally in Concert Technology.</td>
</tr>
<tr>
<td>IloNumExprArray</td>
<td>The class IloNumExprArray.</td>
</tr>
<tr>
<td>IloNumExpr::NonLinearExpression</td>
<td>The class of exceptions thrown if a numerical constant of a non-linear expression is set or queried.</td>
</tr>
<tr>
<td>IloNumVar</td>
<td></td>
</tr>
<tr>
<td>IloNumVarArray</td>
<td>The array class of IloNumVar.</td>
</tr>
<tr>
<td>IloObjective</td>
<td></td>
</tr>
<tr>
<td>IloOr</td>
<td></td>
</tr>
<tr>
<td>IloPack</td>
<td></td>
</tr>
<tr>
<td>IloPathLength</td>
<td></td>
</tr>
<tr>
<td>IloPathTransitI</td>
<td></td>
</tr>
<tr>
<td>IloRandom</td>
<td></td>
</tr>
<tr>
<td>IloRange</td>
<td></td>
</tr>
<tr>
<td>IloRangeArray</td>
<td></td>
</tr>
<tr>
<td>IloSOS1</td>
<td></td>
</tr>
<tr>
<td>IloSOS1Array</td>
<td></td>
</tr>
<tr>
<td>IloSOS2</td>
<td></td>
</tr>
<tr>
<td>IloSOS2Array</td>
<td></td>
</tr>
<tr>
<td>IloSemaphore</td>
<td></td>
</tr>
<tr>
<td>IloSemiContVar</td>
<td></td>
</tr>
<tr>
<td>IloSemiContVarArray</td>
<td></td>
</tr>
<tr>
<td>IloSequence</td>
<td></td>
</tr>
<tr>
<td>IloSolution</td>
<td></td>
</tr>
<tr>
<td>IloSolution::Iterator</td>
<td></td>
</tr>
<tr>
<td>IloSolutionIterator</td>
<td></td>
</tr>
<tr>
<td>IloSolutionManip</td>
<td></td>
</tr>
<tr>
<td>IloTimer</td>
<td></td>
</tr>
<tr>
<td>IloTypedList</td>
<td></td>
</tr>
<tr>
<td>IloTypedListManager</td>
<td></td>
</tr>
</tbody>
</table>

**Macros Summary**

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOANYBINARYPREDICATENAME0</td>
<td></td>
</tr>
<tr>
<td>ILOANYTERNARYPREDICATENAME0</td>
<td></td>
</tr>
<tr>
<td>IloFloatArray</td>
<td></td>
</tr>
<tr>
<td>IloFloatVar</td>
<td></td>
</tr>
<tr>
<td>IloFloatVarArray</td>
<td>The array class of IloFloatVar.</td>
</tr>
</tbody>
</table>
### Enumerations Summary

<table>
<thead>
<tr>
<th>Enumerations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloAlgorithm::Status</td>
<td>An enumeration for the class IloAlgorithm.</td>
</tr>
<tr>
<td>IloDeleterMode</td>
<td>An enumeration to set the mode of an IloDeleter.</td>
</tr>
<tr>
<td>IloNumVar::Type</td>
<td></td>
</tr>
<tr>
<td>IloObjective::Sense</td>
<td></td>
</tr>
</tbody>
</table>

### Type Definitions Summary

<table>
<thead>
<tr>
<th>Type Definitions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IloAny</td>
<td></td>
</tr>
<tr>
<td>IloBool</td>
<td></td>
</tr>
<tr>
<td>IloInt</td>
<td></td>
</tr>
<tr>
<td>IloNum</td>
<td></td>
</tr>
<tr>
<td>IloNumFunction</td>
<td></td>
</tr>
<tr>
<td>IloPathTransitFunction</td>
<td></td>
</tr>
<tr>
<td>IloSolutionArray</td>
<td></td>
</tr>
</tbody>
</table>

### Global Functions Summary

<table>
<thead>
<tr>
<th>Global Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloAbs</td>
<td></td>
</tr>
<tr>
<td>IloAbstraction</td>
<td>Returns a constraint that abstracts the values of one array into the abstract value of another array.</td>
</tr>
<tr>
<td>IloAdd</td>
<td></td>
</tr>
<tr>
<td>IloAllNullIntersect</td>
<td></td>
</tr>
<tr>
<td>IloArcCos</td>
<td></td>
</tr>
<tr>
<td>IloBoolAbstraction</td>
<td>A function to create a constraint that abstracts an array of Boolean variables.</td>
</tr>
<tr>
<td>IloCard</td>
<td></td>
</tr>
<tr>
<td>IloCeil</td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>IloDisableNANDetection</td>
<td></td>
</tr>
<tr>
<td>IloDiv</td>
<td></td>
</tr>
<tr>
<td>IloEnableNANDetection</td>
<td></td>
</tr>
<tr>
<td>IloEndMT</td>
<td></td>
</tr>
<tr>
<td>IloEqIntersection</td>
<td></td>
</tr>
<tr>
<td>IloEqPartition</td>
<td></td>
</tr>
<tr>
<td>IloEqUnion</td>
<td></td>
</tr>
<tr>
<td>IloExponent</td>
<td></td>
</tr>
<tr>
<td>IloFloor</td>
<td></td>
</tr>
<tr>
<td>IloGetClone</td>
<td></td>
</tr>
<tr>
<td>IloInitMT</td>
<td></td>
</tr>
<tr>
<td>IloIsNAN</td>
<td></td>
</tr>
<tr>
<td>IloLog</td>
<td></td>
</tr>
<tr>
<td>IloMax</td>
<td></td>
</tr>
<tr>
<td>IloMaximize</td>
<td></td>
</tr>
<tr>
<td>IloMember</td>
<td></td>
</tr>
<tr>
<td>IloMin</td>
<td></td>
</tr>
<tr>
<td>IloMinimize</td>
<td></td>
</tr>
<tr>
<td>IloMonotonicDecreasingNumExp</td>
<td></td>
</tr>
<tr>
<td>IloMonotonicIncreasingNumExp</td>
<td></td>
</tr>
<tr>
<td>IloNotMember</td>
<td></td>
</tr>
<tr>
<td>IloNullIntersect</td>
<td></td>
</tr>
<tr>
<td>IloPartition</td>
<td></td>
</tr>
<tr>
<td>IloPiecewiseLinear</td>
<td></td>
</tr>
<tr>
<td>IloPower</td>
<td></td>
</tr>
<tr>
<td>IloRound</td>
<td></td>
</tr>
<tr>
<td>IloScalProd</td>
<td></td>
</tr>
<tr>
<td>IloScalProd</td>
<td></td>
</tr>
<tr>
<td>IloScalProd</td>
<td></td>
</tr>
<tr>
<td>IloScalProd</td>
<td></td>
</tr>
<tr>
<td>IloSquare</td>
<td></td>
</tr>
<tr>
<td>IloSubset</td>
<td></td>
</tr>
<tr>
<td>IloSubsetEq</td>
<td></td>
</tr>
<tr>
<td>IloSum</td>
<td></td>
</tr>
<tr>
<td>IloTableConstraint</td>
<td></td>
</tr>
<tr>
<td>operator &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>operator *</td>
<td></td>
</tr>
<tr>
<td>operator new</td>
<td></td>
</tr>
<tr>
<td>operator!</td>
<td></td>
</tr>
<tr>
<td>operator!=</td>
<td></td>
</tr>
</tbody>
</table>
**Description**

Concert Technology offers a C++ library of classes and functions that enable you to design models of problems for both math programming (including linear programming, mixed integer programming, quadratic programming, and network programming) and constraint programming solutions.

<table>
<thead>
<tr>
<th>Global Variables Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILO_NO_MEMORY_MANAGER</td>
</tr>
<tr>
<td>IloHalfPi</td>
</tr>
<tr>
<td>IloInfinity</td>
</tr>
<tr>
<td>IloPi</td>
</tr>
<tr>
<td>IloQuarterPi</td>
</tr>
<tr>
<td>IloThreeHalfPi</td>
</tr>
<tr>
<td>IloTwoPi</td>
</tr>
</tbody>
</table>
This macro defines a predicate class named `nameI` with `n` data members for use in a model. When `n` is greater than 0, the types and names of the data members must be supplied as arguments to the macro. Each data member is defined by its type `T_i` and a name `data_i`. The call to the macro must be followed immediately by the body of the `isTrue` member function of the predicate class being defined. Besides the definition of the class `nameI`, this macro also defines a function named `name` that creates an instance of the class `nameI` and that returns an instance of the class `IloAnyBinaryPredicate` that points to it.

You are not obliged to use this macro to define binary predicates on arbitrary objects. When the macro seems too restrictive for your purposes, we recommend that you define a predicate class directly by subclassing `IlcAnyPredicateI`, documented in the `ILOG Solver Reference Manual`.

Since the argument `name` is used to name the predicate class, it is not possible to use the same name for several predicate definitions.

See Also

`IloAnyBinaryPredicate`
**Category**  
Macro

**Synopsis**  
- ILOANYTERNARYPREDICATENAME0(name, envName)
- ILOANYTERNARYPREDICATENAME1(name, envName, type1, nameArg1)
- ILOANYTERNARYPREDICATENAME2(name, envName, type1, nameArg1, type2, nameArg2)
- ILOANYTERNARYPREDICATENAME3(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3)
- ILOANYTERNARYPREDICATENAME4(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4)
- ILOANYTERNARYPREDICATENAME5(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5)
- ILOANYTERNARYPREDICATENAME6(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5, type6, nameArg6)

**Description**  
This macro defines a predicate class named nameI with n data members for use in a model. When n is greater than 0, the types and names of the data members must be supplied as arguments to the macro. Each data member is defined by its type Ti and a name datai. The call to the macro must be followed immediately by the body of the isTrue member function of the predicate class being defined. Besides the definition of the class nameI, this macro also defines a function named name that creates an instance of the class IloAnyTernaryPredicate that points to it.

You are not obliged to use this macro to define ternary predicates on arbitrary objects. When the macro seems too restrictive for your purposes, we recommend that you define a predicate class directly by subclassing IlcAnyPredicateI (documented in the ILOG Solver Reference Manual).

Since the argument name is used to name the predicate class, it is not possible to use the same name for several predicate definitions.

**See Also**  
IloAnyTernaryPredicate
ILOINT_BINARY_PREDICATE_NAME0

ILOINT_BINARY_PREDICATE_NAME0

Category  Macro

Synopsis  
ILOINT_BINARY_PREDICATE_NAME0(name, envName)
ILOINT_BINARY_PREDICATE_NAME1(name, envName, type1, nameArg1)
ILOINT_BINARY_PREDICATE_NAME2(name, envName, type1, nameArg1, type2, nameArg2)
ILOINT_BINARY_PREDICATE_NAME3(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3)
ILOINT_BINARY_PREDICATE_NAME4(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4)
ILOINT_BINARY_PREDICATE_NAME5(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5)
ILOINT_BINARY_PREDICATE_NAME6(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5, type6, nameArg6)

Description  This macro defines a predicate class named nameI with n data members for use in a model. When n is greater than 0, the types and names of the data members must be supplied as arguments to the macro. Each data member is defined by its type Ti and a name datai. The call to the macro must be followed immediately by the body of the isTrue member function of the predicate class being defined. Besides the definition of the class nameI, this macro also defines a function named name that creates an instance of the class nameI and that returns an instance of the class IloIntBinaryPredicate that points to it.
You are not obliged to use this macro to define binary predicates on arbitrary objects. When the macro seems too restrictive for your purposes, we recommend that you define a predicate class directly by subclassing IlcIntPredicateI, documented in the ILOG Solver Reference Manual.
Since the argument name is used to name the predicate class, it is not possible to use the same name for several predicate definitions.

See Also  IloIntBinaryPredicate
### ILOINTTERNARYPREDICATENAME0

<table>
<thead>
<tr>
<th>Category</th>
<th>Macro</th>
</tr>
</thead>
</table>

#### Synopsis

- `ILOINTTERNARYPREDICATENAME0(name, envName)`
- `ILOINTTERNARYPREDICATENAME1(name, envName, type1, nameArg1)`
- `ILOINTTERNARYPREDICATENAME2(name, envName, type1, nameArg1, type2, nameArg2)`
- `ILOINTTERNARYPREDICATENAME3(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3)`
- `ILOINTTERNARYPREDICATENAME4(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4)`
- `ILOINTTERNARYPREDICATENAME5(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5)`
- `ILOINTTERNARYPREDICATENAME6(name, envName, type1, nameArg1, type2, nameArg2, type3, nameArg3, type4, nameArg4, type5, nameArg5, type6, nameArg6)`

#### Description

This macro defines a predicate class named `nameI` with `n` data members for use in a model. When `n` is greater than 0, the types and names of the data members must be supplied as arguments to the macro. Each data member is defined by its type `T_i` and a name `data_i`. The call to the macro must be followed immediately by the body of the `isTrue` member function of the predicate class being defined. Besides the definition of the class `nameI`, this macro also defines a function named `name` that creates an instance of the class `IloIntTernaryPredicate` that points to it.

You are not obliged to use this macro to define ternary predicates on arbitrary objects. When the macro seems too restrictive for your purposes, we recommend that you define a predicate class directly by subclassing `IlcIntPredicateI` (documented in the *ILOG Solver Reference Manual*).

Since the argument `name` is used to name the predicate class, it is not possible to use the same name for several predicate definitions.

#### See Also

- `IloIntTernaryPredicate`
ILO_NO_MEMORY_MANAGER

Category: Global Variable

Definition File: ilconcert/ilosys.h

This operating-system environment variable enables you to control the memory manager of Concert Technology.

Concert Technology uses its own memory manager to provide faster memory allocation for certain Concert Technology objects. The use of this memory manager can hide memory problems normally detected by memory usage applications (such as Rational Purify, for example). If you are working in a software development environment capable of detecting bad memory access, you can use this operating-system environment variable to turn off the Concert Technology memory manager in order to detect such anomalies during software development.

For example, if you are working in such a development environment on a personal computer running Microsoft NT, use this statement:

```
set ILO_NO_MEMORY_MANAGER=1
```

If you are working on a UNIX platform, using a C-shell, here is one way of setting this environment variable:

```
setenv ILO_NO_MEMORY_MANAGER
```
IloAbs

Category: Global Function

Definition File: ilconcert/iloexpression.h

Synopsis

public IloNumExprArg IloAbs(const IloNumExprArg arg)
public IloNum IloAbs(IloNum val)
public IloNum IloPower(IloNum val1, IloNum val2)
public IloIntExprArg IloAbs(const IloIntExprArg arg)

Description

Concert Technology offers predefined functions that return an expression from an algebraic function on expressions. These predefined functions also return a numeric value from an algebraic function on numeric values as well.

IloAbs returns the absolute value of its argument.

What Is Extracted

IloAbs is extracted by an instance of IloSolver (documented in the ILOG Solver Reference Manual) as an instance of IlcAbs.
IloAbstraction

Category  Global Function
Definition File  ilconcert/iloany.h
Synopsis  

public IloConstraint IloAbstraction(const IloEnv env,
const IloAnyVarArray y,
const IloAnyVarArray x,
const IloAnyArray values,
IloAny abstractValue)

Summary  Returns a constraint that abstracts the values of one array into the abstract value of another array.

Description  This function returns a constraint that abstracts the values of the elements of one array of constrained variables (called \( x \)) in a model into the abstract value of another array of constrained variables (called \( y \)). In other words, for each element \( x[i] \), there is a variable \( y[i] \) corresponding to the abstraction of \( x[i] \) with respect to an array of numeric values. That is,

\[
x[i] = v \text{ with } v \in \text{values if and only if } y[i] = v;
\]

\[
x[i] = v \text{ with } v \notin \text{values if and only if } y[i] = \text{abstractValue}.
\]

This constraint maintains a many-to-one mapping that makes it possible to define constraints that impinge only on a particular set of values from the domains of constrained variables. The abstract value (indicated by \( \text{abstractValue} \)) must not be in the domain of \( x[i] \).

Adding a Constraint to a Model, Extracting a Model for an Algorithm

In order for the constraint returned by IloAbstraction to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

Exceptions  
If the arrays \( x \) and \( y \) are not the same size, this function throws the exception IloAbstraction::InvalidArraysException.

Example  
For simplicity, let’s assume that our array \( x \) consists of three elements with the domains \{3\}, \{4\}, and \{5\}. We will also assume that our abstract value is 7, and the values we are interested in are \{4, 8, 12, 16\}. Then IloAbstraction produces these elements in the array \( y \):
**IloAdd**

**Category**  
Global Function

**Definition File**  
ilconcert/ilomodel.h

**Synopsis**  
```cpp
public X IloAdd(IloModel & mdl,  
                 X x)
```

**Description**  
This C++ template helps when you want to add elements to a model. In those synopses, X represents a class, x is an instance of the class X. The class X must be IloExtractable, IloExtractableArray, or one of their subclasses.

If model is an instance of IloModel, derived from IloExtractable, then x will be added to the top level of that model.

As an alternative to this way of adding extractable objects to a model, you may also use IloModel::add.

This template preserves the original type of its argument x when it returns x. This feature of the template may be useful, for example, in cases like this:

```cpp
IloRange rng = IloAdd(model, 3 * x + y == 17);
```

For a comparison of these two ways of adding extractable objects to a model, see Adding Extractable Objects in the documentation of IloExtractable.

**See Also**  
IloAnd, IloExtractable, IloExtractableArray, IloModel, IloOr
### IloAlgorithm

**Category**
- Class

**Inheritance Path**

**Definition File**
- ilconcert/iloalg.h

**Summary**
- The base class of algorithms in Concert Technology.

#### Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloAlgorithm(IloAlgorithmI *)</td>
</tr>
</tbody>
</table>

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>clear()</td>
</tr>
<tr>
<td>public void</td>
<td>end()</td>
</tr>
<tr>
<td>public ostream &amp;</td>
<td>error()</td>
</tr>
<tr>
<td>public void</td>
<td>IloAlgorithm::extract(const IloModel)</td>
</tr>
<tr>
<td>public IloEnv</td>
<td>getEnv()</td>
</tr>
<tr>
<td>public IloInt</td>
<td>getIntValue(const IloIntVar)</td>
</tr>
<tr>
<td>public void</td>
<td>getIntValues(const IloIntVarArray, IloIntArray)</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getObjValue()</td>
</tr>
<tr>
<td>public IloAlgorithm::Status</td>
<td>getStatus()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getTime()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValue(const IloNumExprArg)</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValue(const IloObjective)</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValue(const IloIntVar)</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValue(const IloNumVar)</td>
</tr>
<tr>
<td>public void</td>
<td>getValues(const IloIntVarArray, IloNumArray)</td>
</tr>
<tr>
<td>public void</td>
<td>getValues(const IloNumVarArray, IloNumArray)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isExtracted(const IloExtractable)</td>
</tr>
</tbody>
</table>
**IloAlgorithm**

**Description**

`IloAlgorithm` is the base class of algorithms in Concert Technology. An instance of this class represents an algorithm in Concert Technology.

In general terms, you define a model, and Concert Technology extracts objects from it for your target algorithm and then solves for solutions.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

**Status**

The member function `getStatus` returns a status indicating information about the currently extracted model and the solution (if there is one). For explanations of the status, see the nested enumeration `IloAlgorithm::Status`.

```
public ostream &    out ()
public void        printTime ()
public void        resetTime ()
public void        setError (ostream &)
public void        setOut (ostream &)
public void        setWarning (ostream &)
public IloBool     solve ()
```

**Inner Enumeration**

<table>
<thead>
<tr>
<th><code>IloAlgorithm::Status</code></th>
<th>An enumeration for the class <code>IloAlgorithm</code>.</th>
</tr>
</thead>
</table>

**Inner Class**

<table>
<thead>
<tr>
<th><code>IloAlgorithm::IloAlgorithm::CannotRemoveException</code></th>
<th>The class of exceptions thrown if an object cannot be removed from a model.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloAlgorithm::IloAlgorithm::CannotExtractException</code></td>
<td>The class of exceptions thrown if an object cannot be extracted from a model.</td>
</tr>
<tr>
<td><code>IloAlgorithm::IloAlgorithm::Exception</code></td>
<td>The base class of exceptions thrown by classes derived from <code>IloAlgorithm</code>.</td>
</tr>
<tr>
<td><code>IloAlgorithm::IloAlgorithm::NotExtractedException</code></td>
<td>The class of exceptions thrown if an extractable object has no value in the current solution of an algorithm.</td>
</tr>
</tbody>
</table>
Exceptions

The class `IloAlgorithm::Exception`, derived from the class `IloException`, is the base class of exceptions thrown by classes derived from `IloAlgorithm`. For an explanation of exceptions thrown by instances of `IloAlgorithm`, see `IloAlgorithm::Exception`.

Streams and Output

The class `IloAlgorithm` supports these communication streams:

- `ostream& out() const;` for general output.
- `ostream& warning() const;` for warning messages about non-fatal conditions.

See Also


Constructors

`public IloAlgorithm(IloAlgorithmI * impl)`

This constructor creates an algorithm in Concert Technology from its implementation object. This is the default constructor.

Methods

`public void clear()`

This member function clears the current model from the algorithm.

`public void end()`

This member function deletes the invoking algorithm. That is, it frees memory associated with the invoking algorithm.

`public ostream & error()`

This member function returns a reference to the stream currently used for warnings from the invoking algorithm. By default, the stream is defined by an instance of `IloEnv` as `cout`.

`public void extract(const IloModel)`

This member function extracts the extractable objects from a model into the invoking algorithm if a member function exists to extract the objects from the model for the invoking algorithm. Not all extractable objects can be extracted by all algorithms; see the documentation of the algorithm class you are using for a list of extractable classes it supports.

When you use this member function to extract extractable objects from a model, it extracts all the elements of that model for which Concert Technology creates the representation of the extractable object suitable for the invoking algorithm.
The attempt to extract may fail. In case such a failure occurs, Concert Technology throws the exception `CannotExtractException` on platforms that support C++ exceptions when exceptions are enabled.

For example, a failure will occur if you attempt to extract more than one objective for an invoking algorithm that accepts only one objective, and Concert Technology will throw the exception `MultipleObjException` (documented in the ILOG CPLEX Reference Manual).

```java
public IloEnv getEnv()
```

This member function returns the environment of the invoking algorithm.

```java
public IloInt getIntValue(const IloIntVar)
```

This member function returns the integer value of an integer variable in the current solution of the invoking algorithm. For example, to access the variable, use the member function `getIntValue(var)` where `var` is an instance of the class `IloIntVar`.

If there is no value to return for `var`, this member function raises an error. This member function throws the exception `NotExtractedException` if there is no value to return (for example, if `var` was not extracted by the invoking algorithm).

```java
public void getIntValues(const IloIntVarArray, IloIntArray)
```

This member function accepts an array of variables `vars` and puts the corresponding values into the array `vals`; the corresponding values come from the current solution of the invoking algorithm. The array `vals` must be a clean, empty array when you pass it to this member function.

If there are no values to return for `vars`, this member function raises an error. On platforms that support C++ exceptions, when exceptions are enabled, this member function throws the exception `NotExtractedException` in such a case.

```java
public IloNum getObjValue()
```

This member function returns the numeric value of the objective function associated with the invoking algorithm.

```java
public IloAlgorithm::Status getStatus()
```

This member function returns a status indicating information about the current model and the solution. For explanations of the status, see the nested enumeration `IloAlgorithm::Status`.

```java
public IloNum getTime()
```

This member function returns the amount of time elapsed in seconds since the most recent reset of the invoking algorithm. (The member function `printTime` directs the output of `getTime` to the output channel of the invoking algorithm.)
The type of time returned is platform dependent. On Windows systems, the time returned is elapsed wall clock time. On UNIX systems, the time returned is CPU time.

```java
public IloNum getValue(const IloNumExprArg)
```

This member function returns the value of an expression in the current solution of the invoking algorithm. For example, to access the expression, use the member function `getValue(expr)` where `expr` is an instance of the class `IloNumExprArg`.

If there is no value to return for `expr`, this member function raises an error. This member function throws the exception `NotExtractedException` if there is no value to return (for example, if `expr` was not extracted by the invoking algorithm).

```java
public IloNum getValue(const IloObjective)
```

This member function returns the value of an objective in the current solution of the invoking algorithm. For example, to access the objective, use the member function `getValue(obj)` where `obj` is an instance of the class `IloObjective`.

If there is no value to return for `obj`, this member function raises an error. This member function throws the exception `NotExtractedException` if there is no value to return (for example, if `obj` was not extracted by the invoking algorithm).

```java
public IloNum getValue(const IloIntVar)
```

This member function returns the numerical value of an integer variable in the current solution of the invoking algorithm. For example, to access the variable, use the member function `getValue(var)` where `var` is an instance of the class `IloIntVar`.

If there is no value to return for `var`, this member function raises an error. This member function throws the exception `NotExtractedException` if there is no value to return (for example, if `var` was not extracted by the invoking algorithm).

```java
public IloNum getValue(const IloNumVar)
```

This member function returns the numeric variable in the current solution of the invoking algorithm. For example, to access the variable, use the member function `getValue(var)` where `var` is an instance of the class `IloNumVar`.

If there is no value to return for `var`, this member function raises an error. This member function throws the exception `NotExtractedException` if there is no value to return (for example, if `var` was not extracted by the invoking algorithm).

```java
public void getValues(const IloIntVarArray, IloNumArray)
```

This member function accepts an array of variables `vars` and puts the corresponding values into the array `vals`; the corresponding values come from the current solution of the invoking algorithm. The array `vals` must be a clean, empty array when you pass it to this member function.
If there are no values to return for `vars`, this member function raises an error. On platforms that support C++ exceptions, when exceptions are enabled, this member function throws the exception `NotExtractedException` in such a case.

```java
public void getValues(const IloNumVarArray, IloNumArray)
```

This member function accepts an array of variables `vars` and puts the corresponding values into the array `vals`; the corresponding values come from the current solution of the invoking algorithm. The array `vals` must be a clean, empty array when you pass it to this member function.

If there are no values to return for `vars`, this member function raises an error. On platforms that support C++ exceptions, when exceptions are enabled, this member function throws the exception `NotExtractedException` in such a case.

```java
public IloBool isExtracted(const IloExtractable)
```

This member function returns `IloTrue` if `extr` has been extracted for the invoking algorithm; otherwise, it returns `IloFalse`.

```java
public ostream & out()
```

This member function returns a reference to the stream currently used for warnings from the invoking algorithm. By default, the stream is defined by an instance of `IloEnv` as `cout`.

```java
public void printTime()
```

This member function directs the output of the member function `getTime` to an output channel of the invoking algorithm. (The member function `getTime` accesses the elapsed time in seconds since the most recent reset of the invoking algorithm.)

```java
public void resetTime()
```

This member function resets the timer on the invoking algorithm. The type of timer is platform dependent. On Windows systems, the time is elapsed wall clock time. On UNIX systems, the time is CPU time.

```java
public void setError(ostream &)
```

This member function sets the stream for errors generated by the invoking algorithm. By default, the stream is defined by an instance of `IloEnv` as `cout`.

```java
public void setOut(ostream &)
```

This member function redirects the `out()` stream with the stream given as a parameter. This member function can be used with `getNullStream` to suppress screen output by redirecting it to the null stream.

```java
public void setWarning(ostream &)
```
This member function sets the stream for warnings from the invoking algorithm. By
default, the stream is defined by an instance of IloEnv as cout.

    public IloBool solve()

This member function solves the current model in the invoking algorithm. In other
words, solve works with all extractable objects extracted from the model for the
algorithm. The member function returns IloTrue if it finds a solution (not necessarily
an optimal one). Here is an example of its use:

    if (algo.solve()) {
        algo.out() << "Status is " << algo.getStatus() << endl;
    }

If an objective of the model has been extracted into the invoking algorithm, this member
function solves the model to optimality. If there is currently no objective, this member
function searches for the first feasible solution. A feasible solution is not necessarily
optimal, though it satisfies all constraints.
**IloAlgorithm::CannotExtractException**

**Category**  
Inner Class

**InheritancePath**  
IloException  
IloAlgorithm::Exception  
IloAlgorithm::CannotExtractException

**Definition File**  
ilconcert/iloalg.h

**Summary**  
The class of exceptions thrown if an object cannot be extracted from a model.

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void end ()</td>
<td>This member function deletes the invoking exception. That is, it frees memory associated with the invoking exception.</td>
</tr>
<tr>
<td>public const IloAlgorithmI * getAlgorithm ()</td>
<td>The member function getAlgorithm returns the algorithm from which the exception was thrown.</td>
</tr>
<tr>
<td>public IloExtractableArray &amp; getExtractables ()</td>
<td></td>
</tr>
</tbody>
</table>

**Inherited methods from IloException**

IloException::end, IloException::getMessage

**Description**  
If an attempt to extract an object from a model fails, this exception is thrown.

**Methods**  

public void end ()

This member function deletes the invoking exception. That is, it frees memory associated with the invoking exception.

public const IloAlgorithmI * getAlgorithm ()

The member function getAlgorithm returns the algorithm from which the exception was thrown.

public IloExtractableArray & getExtractables ()
The member function `getExtractables` returns the extractable objects that triggered the exception.
IloAlgorithm::CannotRemoveException

Category: Inner Class

InheritancePath:

- IloException
- IloAlgorithm::CannotRemoveException

Definition File: ilconcert/iloalg.h

Summary: The class of exceptions thrown if an object cannot be removed from a model.

Method Summary:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void end()</td>
<td></td>
</tr>
<tr>
<td>public const IloAlgorithmI * getAlgorithm()</td>
<td></td>
</tr>
<tr>
<td>public IloExtractableArray &amp; getExtractables()</td>
<td></td>
</tr>
</tbody>
</table>

Inherited methods from IloException:
- IloException::end, IloException::getMessage

Description:
If an attempt to remove an extractable object from a model fails, this exception is thrown.

Methods:

public void end()

This member function deletes the invoking exception. That is, it frees memory associated with the invoking exception.

public const IloAlgorithmI * getAlgorithm()

The member function getAlgorithm returns the algorithm from which the exception was thrown.

public IloExtractableArray & getExtractables()
The member function `getExtractables` returns the extractable objects that triggered the exception.
**IloAlgorithm::Exception**

**Category**  Inner Class

**InheritancePath**

```
IloException
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>IloAlgorithm::Exception</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>IloAlgorithm::CannotExtractException</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>IloAlgorithm::CannotRemoveException</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>IloAlgorithm::NotExtractedException</td>
</tr>
</tbody>
</table>
```

**Definition File**  ilconcert/iloalg.h

**Summary**  The base class of exceptions thrown by classes derived from IloAlgorithm.

**Constructor Summary**

| public          | Exception(const char *) |

**Inherited methods from IloException**

- IloException::end, IloException::getMessage

**Description**

IloAlgorithm is the base class of algorithms in Concert Technology.

The class IloAlgorithm::Exception, derived from the class IloException, is the base class of exceptions thrown by classes derived from IloAlgorithm.

On platforms that support C++ exceptions, when exceptions are enabled, the member function IloAlgorithm::extract will throw an exception if you attempt to extract an unsuitable object from your model for an algorithm. An extractable object is unsuitable for an algorithm if there is no member function to extract the object from your model to that algorithm.
For example, an attempt to extract more than one objective into an algorithm that accepts only one objective will throw an exception.

Similarly, the member function getValue will throw an exception if you attempt to access the value of a variable that has not yet been bound to a value.

**See Also**

`IloAlgorithm`, `IloException`

**Constructors**

```cpp
public Exception(const char * str)
```

This constructor creates an exception thrown from a member of IloAlgorithm. The exception contains the message string str, which can be queried with the member function `IloException::getMessage`. 
IloAlgorithm::NotExtractedException

Category          Inner Class

InheritancePath   

Definition File   ilconcert/iloalg.h

Summary           The class of exceptions thrown if an extractable object has no value in the current solution of an algorithm.

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>NotExtractedException(const IloAlgorithm *,</td>
</tr>
<tr>
<td></td>
<td>const IloExtractable)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public const IloAlgorithm *</td>
<td>getAlgorithm()</td>
</tr>
<tr>
<td>public const IloExtractable &amp;</td>
<td>getExtractable()</td>
</tr>
</tbody>
</table>

Inherited methods from IloException

IloException::end, IloException::getMessage

Description

If an expression, numeric variable, objective, or array of extractable objects has no value in the current solution of an algorithm, this exception is thrown.

Constructors

public NotExtractedException(const IloAlgorithm *,   |
const IloExtractable)
The constructor `NotExtractedException` creates an exception thrown from the algorithm object `alg` for the extractable object `extr`.

**Methods**

```cpp
public const IloAlgorithmI * getAlgorithm()
```

The member function `getAlgorithm` returns the algorithm from which the exception was thrown.

```cpp
public const IloExtractable & getExtractable()
```

The member function `getExtractable` returns the extractable object that triggered the exception.
IloAlgorithm::Status

Category Inner Enumeration

Definition File ilconcert/iloalg.h

Synopsis

Status {
  Unknown,
  Feasible,
  Optimal,
  Infeasible,
  Unbounded,
  InfeasibleOrUnbounded,
  Error
};

Summary

An enumeration for the class IloAlgorithm.

Description

IloAlgorithm is the base class of algorithms in Concert Technology, and IloAlgorithm::Status is an enumeration limited in scope to the class IloAlgorithm. The member function getStatus returns a status indicating information about the current model and the solution.

Unknown indicates that the algorithm has no information about the solution of the model.

Feasible indicates that the algorithm found a feasible solution (that is, an assignment of values to variables that satisfies the constraints of the model, though it may not necessarily be optimal). The member functions getValue access this feasible solution.

Optimal indicates that the algorithm found an optimal solution (that is, an assignment of values to variables that satisfies all the constraints of the model and that is proved optimal with respect to the objective of the model). The member functions getValue access this optimal solution.

Infeasible indicates that the algorithm proved the model infeasible; that is, it is not possible to find an assignment of values to variables satisfying all the constraints in the model.

Unbounded indicates that the algorithm proved the model unbounded.

InfeasibleOrUnbounded indicates that the model is infeasible or unbounded.

Error indicates that an error occurred and, on platforms that support exceptions, that an exception has been thrown.
**See Also**

- \texttt{IloAlgorithm}, \texttt{operator}, \texttt{IloCplex::Status} documented in the ILOG CPLEX Reference Manual

**Fields**

- Unknown
- Feasible
- Optimal
- Infeasible
- Unbounded
- InfeasibleOrUnbounded
- Error
**IloAllDiff**

**Category**  
Class

**InheritancePath**

**Definition File**  
ilconcert/ilomodel.h

**Summary**  
A constraint that forces constrained integer variables to assume different values in a model.

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAllDiff()</td>
<td></td>
</tr>
<tr>
<td>public IloAllDiff(ImplClass *)</td>
<td></td>
</tr>
<tr>
<td>public IloAllDiff(const IloEnv, const IloIntVarArray, const char *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass * getImpl()</td>
<td></td>
</tr>
</tbody>
</table>

**Inherited methods from IloConstraint**

IloConstraint::getImpl
Description
An instance of this class is a constraint that forces constrained integer variables to assume different values from one another in a model. In other words, no two of those integer variables will have the same integer value when this constraint is satisfied.

In order for the constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept Assert and NDEBUG.

What Is Extracted
All the variables (that is, instances of `IloNumVar` or one of its subclasses) that have appeared as a parameter of a constructor of `IloAllDiff` and all variables that have been explicitly added to the instance of `IloAllDiff` will be extracted by an algorithm (such as an instance of `IloSolver`, documented in the ILOG Solver Reference Manual) that extracts that constraint.

See Also
`IloAdd`, `IloConstraint`, `IloDiff`

Constructors

public `IloAllDiff()`

This constructor creates an empty handle. You must initialize it before you use it.
public \texttt{IloAllDiff}(\texttt{ImplClass} * impl)

This constructor creates a handle object from a pointer to an implementation object.

public \texttt{IloAllDiff}(\texttt{const IloEnv} env,
const \texttt{IloIntVarArray} vars,
const char * name)

This constructor creates a constraint that forces all the integer variables in \texttt{vars} to assume different values from each other. If \texttt{vars} is empty, this constructor creates an empty instance of \texttt{IloAllDiff}, and then you must fill the constraint; that is, you must put variables into the array. You must add this constraint to a model and extract the model for an algorithm in order for it to be taken into account.

**Methods**

public \texttt{ImplClass} * \texttt{getImpl}()

This member function returns a pointer to the implementation object of the invoking handle.
IloAllMinDistance

Category: Class

InheritancePath

Definition File: ilconcert/ilomodel.h

Summary: A constraint on the minimum absolute distance between a pair of variables in an array.

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IloAllMinDistance()</td>
</tr>
<tr>
<td></td>
<td>IloAllMinDistance(ImplClass *)</td>
</tr>
<tr>
<td></td>
<td>IloAllMinDistance(const IloEnv,</td>
</tr>
<tr>
<td></td>
<td>const IloIntVarArray, IloInt,</td>
</tr>
<tr>
<td></td>
<td>const char *)</td>
</tr>
</tbody>
</table>

Method Summary

| Public ImplClass * | getImpl()                                    |

Inherited methods from IloConstraint

IloConstraint::getImpl
An instance of the class IloAllMinDistance is a constraint that makes sure that the absolute distance between any pair of variables in an array of constrained numeric variables will be greater than or equal to a given integer.

See Also

IloAllDiff

Constructors

public IloAllMinDistance()

This constructor creates an empty handle. You must initialize it before you use it.

public IloAllMinDistance(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloAllMinDistance(const IloEnv env,
const IloIntVarArray vars,
IloInt k,
const char * name)

This constructor returns a constraint that insures that the absolute distance between any pair of variables in the array vars will be greater than or equal to k. You must add this constraint to a model and extract the model for an algorithm in order for it to be taken into account.

Methods

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.
IloAllNullIntersect

**Category**  Global Function

**Definition File**  ilconcert/iloanyset.h

**Synopsis**

public IloConstraint IloAllNullIntersect(const IloEnv env,
const IloAnySetVarArray vars)

public IloConstraint IloAllNullIntersect(const IloEnv env,
const IloIntSetVarArray vars)

**Description**

This function creates and returns a constraint (an instance of IloConstraint) for use in a model. The constraint insures that for the sets in the array vars, the intersection of vars[i] with vars[j] will be empty for all i and j when this constraint is satisfied.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
### IloAnd

**Category**  
Class

**Inheritance Path**

```
IloExtractable
  |    
  v    
IloNumExprArg
  |    
  v    
IloIntExprArg
  |    
  v    
IloConstraint
  |    
  v    
IloAnd
```

**Definition File**  
ilconcert/ilomodel.h

**Summary**  
A conjunctive constraint that defines a logical conjunctive-AND among other constraints.

#### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAnd()</td>
<td>IloAnd()</td>
</tr>
<tr>
<td>public IloAnd(ImplClass *)</td>
<td>IloAnd(ImplClass *)</td>
</tr>
<tr>
<td>public IloAnd(const IloEnv, const char *)</td>
<td>IloAnd(const IloEnv, const char *)</td>
</tr>
</tbody>
</table>

#### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void add(const IloConstraintArray)</td>
<td>add(const IloConstraintArray)</td>
</tr>
<tr>
<td>public void add(const IloConstraint)</td>
<td>add(const IloConstraint)</td>
</tr>
<tr>
<td>public ImplClass * getImpl()</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public void remove(const IloConstraintArray)</td>
<td>remove(const IloConstraintArray)</td>
</tr>
<tr>
<td>public void remove(const IloConstraint)</td>
<td>remove(const IloConstraint)</td>
</tr>
</tbody>
</table>
Description

An instance of `IloAnd` represents a conjunctive constraint. In other words, it defines a logical conjunctive-AND among any number of constraints. It lets you represent a constraint on constraints in your model. Since an instance of `IloAnd` is a constraint itself, you can build up extensive conjunctions by adding constraints to an instance of `IloAnd` by means of the member function `add`. You can also remove constraints from an instance of `IloAnd` by means of the member function `remove`.

The elements of a conjunctive constraint must be in the same environment.

In order for the constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

Conjunction of Goals

---

Inherited methods from `IloConstraint`

- `IloConstraint::getImpl`
If you want to represent the conjunction of goals (rather than constraints) in your model, then you should consider the function \texttt{IloAndGoal} (documented in the ILOG Solver Reference Manual).

**What Is Extracted**

All the constraints (that is, instances of \texttt{IloConstraint} or one of its subclasses) that have been added to a conjunctive constraint (an instance of \texttt{IloAnd}) and that have not been removed from it will be extracted when an algorithm such as \texttt{IloSolver} (documented in the ILOG Solver Reference Manual) extracts the constraint.

**Example**

For example, you may write:

\begin{verbatim}
IloAnd and(env);
and.add(constraint1);
and.add(constraint2);
and.add(constraint3);
\end{verbatim}

Those lines are equivalent to:

\begin{verbatim}
IloAnd and = constraint1 && constraint2 && constraint3;
\end{verbatim}

**See Also**

\texttt{IloConstraint, IloOr, operator &&}

**Constructors**

- \texttt{public IloAnd()}
  
  This constructor creates an empty handle. You must initialize it before you use it.

- \texttt{public IloAnd(ImplClass * impl)}
  
  This constructor creates a handle object from a pointer to an implementation object.

- \texttt{public IloAnd(const IloEnv env,}
  \hspace{1em} \texttt{const char * name)}
  
  This constructor creates a conjunctive constraint for use in the environment \texttt{env}. In order for the constraint to take effect, you must add it to a model with the template \texttt{IloAdd} or the member function \texttt{IloModel::add} and extract the model for an algorithm with the member function \texttt{IloAlgorithm::extract}.

  The optional argument \texttt{name} is set to 0 by default.

**Methods**

- \texttt{public void add(const IloConstraintArray array)}

This member function makes all the elements in array elements of the invoking conjunctive constraint. In other words, it applies the invoking conjunctive constraint to all the elements of array.

**Note:** The member function *add* notifies Concert Technology algorithms about this change to the invoking object.

```java
public void add(const IloConstraint constraint)
```

This member function makes *constraint* one of the elements of the invoking conjunctive constraint. In other words, it applies the invoking conjunctive constraint to *constraint*.

**Note:** The member function *add* notifies Concert Technology algorithms about this change to the invoking object.

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```java
public void remove(const IloConstraintArray array)
```

This member function removes all the elements of array from the invoking conjunctive constraint so that the invoking conjunctive constraint no longer applies to any of those elements.

**Note:** The member function *remove* notifies Concert Technology algorithms about this change to the invoking object.

```java
public void remove(const IloConstraint constraint)
```

This member function removes *constraint* from the invoking conjunctive constraint so that the invoking conjunctive constraint no longer applies to *constraint*.

**Note:** The member function *remove* notifies Concert Technology algorithms about this change to the invoking object.
IloAny

Category: Type Definition

Definition File: ilconcert/ilosys.h

Synopsis: void * IloAny

Description: This type definition represents objects in a model handled by Concert Technology enumerated variables and by Concert Technology set variables. A pointer to any object, whether a predefined Concert Technology type or an instance of a C++ class, is implicitly converted to IloAny. By using this type, you can be sure that these components of your application will port without any source change across different hardware platforms.

See Also: IloModel
IloAnyArray

**Category**  Class

**InheritancePath**

**Definition File**  ilconcert/iloany.h

**Summary**  The array class of the enumerated type definition IloAny.

---

### Constructor Summary

| public | IloAnyArray(IloDefaultArrayI *) |
| public | IloAnyArray(const IloAnyArray &) |
| public | IloAnyArray(const IloEnv, IloInt) |
| public | IloAnyArray(const IloEnv, IloInt, const IloAny, const IloAny, ...) |

---

### Method Summary

| public void | add(const IloAny) |

---

**Description**  For each basic type, Concert Technology defines a corresponding array class. IloAnyArray is the array class of the basic enumerated type definition (IloAny) for a model.

Instances of IloAnyArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

If you would like to represent a set of enumerated values (that is, no repeated elements, no order among elements), consider an instance of IloAnySet.
Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

**See Also**

`IloAny`, `IloAnySet`, `operator>>`, `operator`

**Constructors**

public `IloAnyArray(IloDefaultArrayI * i)`

This constructor creates an empty array of elements. You cannot create instances of the undocumented class `IloDefaultArrayI`. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public `IloAnyArray(const IloAnyArray & copy)`

This copy constructor creates a handle to the array of pointers indicated by `copy`.

public `IloAnyArray(const IloEnv env, `IloInt n)`

This constructor creates an array of `n` elements, all of which are empty handles.

public `IloAnyArray(const IloEnv env, `IloInt n, `const IloAny p0, `const IloAny p1, `...)`

This constructor creates an array of `n` elements for use in a model.

**Methods**

public `void add(const IloAny p)`

This member function appends `p` to the invoking array.
IloAnyBinaryPredicate

Category   Class
InheritancePath

Definition File   ilconcert/ilotupleset.h
Summary   A class to define binary predicates on objects in a model.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAnyBinaryPredicate()</td>
<td></td>
</tr>
<tr>
<td>public IloAnyBinaryPredicate(IloAnyBinaryPredicateI *)</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAnyBinaryPredicateI * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public IloBool isTrue(const IloAny, const IloAny)</td>
<td></td>
</tr>
<tr>
<td>public void operator=(const IloAnyBinaryPredicateI &amp;)</td>
<td></td>
</tr>
</tbody>
</table>

Description

This class makes it possible for you to define binary predicates operating on arbitrary objects in a model. A predicate is an object with a member function (such as `isTrue`) that checks whether or not a property is satisfied by an ordered set of (pointers to) objects.

Defining a New Class of Predicates

Predicates, like other Concert Technology objects, depend on two classes: a handle class, `IloAnyBinaryPredicate`, and an implementation class, such as `IloAnyBinaryPredicateI`, where an object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of an instance of `IloAnyBinaryPredicateI` allocated in a Concert Technology
environment. As a Concert Technology user, you will be working primarily with handles.

If you define a new class of predicates yourself, you must define its implementation class together with the corresponding virtual member function isTrue, as well as a member function that returns an instance of the handle class IloAnyBinaryPredicate.

**Arity**

As a developer, you can use predicates in Concert Technology applications to define your own constraints that have not already been predefined in Concert Technology. In that case, the *arity* of the predicate (that is, the number of constrained variables involved in the predicate, and thus the size of the array that the member function isTrue must check) must be two.

**See Also**

IloTableConstraint

**Constructors**

```cpp
public IloAnyBinaryPredicate()
```

This constructor creates an empty binary predicate. In other words, the predicate is an empty handle with a null handle pointer. You must assign the elements of the predicate before you attempt to access it, just as you would any other pointer. Any attempt to access it before this assignment will throw an exception (an instance of IloSolver::SolverErrorException).

```cpp
public IloAnyBinaryPredicate(IloAnyBinaryPredicateI * impl)
```

This constructor creates a handle object (an instance of the class IloAnyBinaryPredicate) from a pointer to an implementation object (an instance of the implementation class IlcAnyPredicateI, documented in the ILOG Solver Reference Manual).

**Methods**

```cpp
public IloAnyBinaryPredicateI * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```cpp
public IloBool isTrue(const IloAny val1,
                     const IloAny val2)
```

This member function returns IloTrue if the values val1 and val2 make the invoking binary predicate valid. It returns IloFalse otherwise.

```cpp
public void operator=(const IloAnyBinaryPredicate & h)
```

This assignment operator copies h into the invoking predicate by assigning an address to the handle pointer of the invoking object. That address is the location of the implementation object of the argument h. After execution of this operator, both the invoking predicate and h point to the same implementation object.
IloAnySet

Category       Class

InheritancePath

Definition File ilconcert/iloanyset.h

Summary       A class whose instances represent a set of enumeration values.

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloAnySet(const IloEnv, const IloAnyArray)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySet(const IloEnv)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>add(IloAnySet)</td>
</tr>
<tr>
<td>public void</td>
<td>add(IloAny)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>contains(IloAnySet)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>contains(IloAny)</td>
</tr>
<tr>
<td>public IloAnySet</td>
<td>copy()</td>
</tr>
<tr>
<td>public void</td>
<td>empty()</td>
</tr>
<tr>
<td>public void</td>
<td>end()</td>
</tr>
<tr>
<td>public IloEnv</td>
<td>getEnv()</td>
</tr>
<tr>
<td>public IloInt</td>
<td>getSize()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>intersects(IloAnySet)</td>
</tr>
<tr>
<td>public void</td>
<td>remove(IloAnySet)</td>
</tr>
<tr>
<td>public void</td>
<td>remove(IloAny)</td>
</tr>
<tr>
<td>public void</td>
<td>setIntersection(IloAnySet)</td>
</tr>
<tr>
<td>public void</td>
<td>setIntersection(IloAny)</td>
</tr>
</tbody>
</table>
An instance of this class represents a set of enumerated values. The same enumerated value will not appear more than once in a set. The elements of a set are not ordered. The class `IloAnySet::Iterator` offers you a way to traverse the elements of such a set.

If you are considering modeling issues where you want to represent repeated elements or where you want to exploit an indexed order among the elements, then you might want to look at the class `IloAnyArray` instead of this class for sets.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

### See Also

`IloAny`, `IloAnyArray`, `IloAnySet::Iterator` 

### Constructors

```cpp
public IloAnySet(const IloEnv env, 
                  const IloAnyArray array)
```

This constructor creates a set of enumerated values for `env` from the elements in `array`.

```cpp
public IloAnySet(const IloEnv env)
```

This constructor creates an empty set (no elements) for `env`. You must use the member function `IloAnySet::add` to fill this set with elements.

### Methods

```cpp
public void add(IloAnySet set)
```

This member function adds `set` to the invoking set. By adds, we mean that the invoking set becomes the union of its former elements and the elements of `set`.

```cpp
public void add(IloAny elt)
```

This member function adds `elt` to the invoking set. By adds, we mean that the invoking set becomes the union of its former elements and the new `elt`.

```cpp
public IloBool contains(IloAnySet set)
```

This member function returns a Boolean value (zero or one) that indicates whether `set` intersects the invoking set. The value one indicates that the invoking set contains all the elements of `set`, and that the intersection of the invoking set with `set` is precisely `set`. 

---

**IloAnySet**

<table>
<thead>
<tr>
<th>Inner Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloAnySet::IloAnySet::Iterator</code></td>
</tr>
</tbody>
</table>
The value zero indicates that the intersection of the invoking set \( \text{set} \) is not precisely set.

```java
public IloBool contains(IloAny elt)
```

This member function returns a Boolean value (zero or one) that indicates whether \( \text{elt} \) is an element of the invoking set. The value one indicates that the invoking set contains \( \text{elt} \); the value zero indicates that the invoking set does not contain \( \text{elt} \).

```java
public IloAnySet copy()
```

This member function creates a clone of the array.

```java
public void empty()
```

This member function removes the elements from the invoking set. In other words, the invoking set becomes the empty set.

```java
public void end()
```

This member function reclaims memory associated with the invoking set.

```java
public IloEnv getEnv()
```

This member function returns the environment of the invoking set.

```java
public IloInt getSize()
```

This member function returns an integer indicating the size of the invoking set (that is, how many elements it contains).

```java
public IloBool intersects(IloAnySet set)
```

This member function returns a Boolean value (zero or one) that indicates whether \( \text{set} \) intersects the invoking set. The value one indicates that the intersection of \( \text{set} \) and the invoking set is not empty (at least one element in common); the value zero indicates that the intersection of \( \text{set} \) and the invoking set is empty (no elements in common).

```java
public void remove(IloAnySet set)
```

This member function removes all the elements of \( \text{set} \) from the invoking set.

```java
public void remove(IloAny elt)
```

This member function removes \( \text{elt} \) from the invoking set.

```java
public void setIntersection(IloAnySet set)
```

This member function changes the invoking set so that it includes only the elements of \( \text{set} \). In other words, the invoking set becomes the intersection of its former elements with the elements of \( \text{set} \).

```java
public void setIntersection(IloAny elt)
```
This member function changes the invoking set so that it includes only the element indicated by \texttt{elt}. In other words, the invoking set becomes the intersection of its former elements with \texttt{elt}.
IloAnySet::Iterator

Category: Inner Class

Inheritance Path

Definition File: ilconcert/iloanyset.h

Summary: An iterator to traverse the elements of IloAnySet.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool ok()</td>
<td>This member function returns IloTrue if there is a current element and invoking iterator points to it. Otherwise, it returns IloFalse.</td>
</tr>
<tr>
<td>public IloAny operator *()</td>
<td>This operator returns the current element, the one to which the invoking iterator points.</td>
</tr>
<tr>
<td>public void operator++()</td>
<td>This operator advances the iterator to point to the next value in the set.</td>
</tr>
</tbody>
</table>

Description: An instance of the nested class IloAnySet::Iterator is an iterator that traverses the elements of a finite set of pointers (an instance of IloAnySet).

See Also: IloAnySet

Methods

public IloBool ok()

This member function returns IloTrue if there is a current element and invoking iterator points to it. Otherwise, it returns IloFalse.

To traverse the elements of a finite set of pointers, use the following code:

```cpp
IloAny val;
for(IloAnySet::Iterator iter(set); iter.ok(); ++iter){
   val = *iter;
   // do something with val
}
```

public IloAny operator *()

This operator returns the current element, the one to which the invoking iterator points.

public void operator++()

This operator advances the iterator to point to the next value in the set.
IloAnySetVar

Category  Class

InheritancePath

Definition File  ilconcert/iloanyset.h

Summary  Class to represent a set of enumerated values as a constrained variable.

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloAnySetVar()</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySetVar(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySetVar(const IloEnv, const IloAnyArray, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySetVar(const IloEnv, const IloAnyArray, const IloAnyArray, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>addPossible(IloAny)</td>
</tr>
<tr>
<td>public void</td>
<td>addRequired(IloAny)</td>
</tr>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public void</td>
<td>getPossibleSet(IloAnySet)</td>
</tr>
<tr>
<td>public IloAnySet</td>
<td>getPossibleSet()</td>
</tr>
<tr>
<td>public IloAnySet::Iterator</td>
<td>getPossibleSetIterator()</td>
</tr>
<tr>
<td>public void</td>
<td>getRequiredSet(IloAnySet)</td>
</tr>
<tr>
<td>public IloAnySet</td>
<td>getRequiredSet()</td>
</tr>
<tr>
<td>public IloAnySet::Iterator</td>
<td>getRequiredSetIterator()</td>
</tr>
<tr>
<td>public void</td>
<td>removePossible(IloAny)</td>
</tr>
<tr>
<td>public void</td>
<td>removeRequired(IloAny)</td>
</tr>
</tbody>
</table>
An instance of this class offers a convenient way to represent a set of enumerated values as a constrained variable in Concert Technology.

A constrained variable representing a set of enumerated values (that is, an instance of `IloAnySetVar`) is defined in terms of two other sets: its required elements and its possible elements. Its required elements are those that must be in the set. Its possible elements are those that may be in the set. This class offers member functions for accessing the required and possible elements of a set of enumerated values.

The function `IloCard` offers you a way to constrain the number of elements in a set variable. That is, `IloCard` constrains the cardinality of a set variable.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

### See Also

- `IloAnySet`, `IloAnySetVarArray`, `IloCard`, `IloEqIntersection`, `IloEqUnion`, `IloExtractable`, `IloMember`, `IloModel`, `IloNotMember`, `IloNullIntersect`, `IloSubset`, `IloSubsetEq`

### Constructors

**public `IloAnySetVar()`**

This constructor creates an empty handle. You must initialize it before you use it.

**public `IloAnySetVar(ImplClass * impl)`**

This constructor creates a handle object from a pointer to an implementation object.

**public `IloAnySetVar(const IloEnv env, const IloAnyArray possible, const char * name)`**

This constructor creates a constrained set variable from the values in `possible` and makes the set variable part of the environment indicated by `env`, where the set consists of enumerated values. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.

**public `IloAnySetVar(const IloEnv env, const IloAnyArray possible, const IloAnyArray required, const char * name)`**
With this constructor, you can indicate both the required and the possible sets that characterize the instance of IloAnySetVar that it creates. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.

**Methods**

```java
public void addPossible(IloAny elt)
```

This member function adds `elt` to the set of possible elements of the invoking set variable.

**Note:** *The member function addPossible notifies Concert Technology algorithms about this change of this invoking object.*

```java
public void addRequired(IloAny elt)
```

This member function adds `elt` to the set of required elements of the invoking set variable.

**Note:** *The member function addRequired notifies Concert Technology algorithms about this change of this invoking object.*

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```java
public void getpossibleSet(IloAnySet set)
```

This member function accesses the possible elements of the invoking set variable and puts those elements into its parameter `set`.

```java
public IloAnySet getpossibleSet()
```

This member function returns the possible elements of the invoking set variable.

```java
public IloAnySet::Iterator getpossibleSetIterator()
```

This member function returns an `IloAnySet::Iterator` to traverse the possible elements of the invoking set variable.

```java
public void getRequiredSet(IloAnySet set)
```

This member function accesses the required elements of the invoking set variable and puts those elements into its parameter `set`.

```java
public IloAnySet getRequiredSet()
```

This member function returns the required elements of the invoking set variable.
public IloAnySet::Iterator getRequiredSetIterator()

This member function returns an IloAnySet::Iterator to traverse the required elements of the invoking set variable.

public void removePossible(IloAny elt)

This member function removes elt as a possible element of the invoking set variable.

Note: The member function removePossible notifies Concert Technology algorithms about this change of this invoking object.

public void removeRequired(IloAny elt)

This member function removes elt as a required element of the invoking set variable.

Note: The member function removeRequired notifies Concert Technology algorithms about this change of this invoking object.
**IloAnySetVarArray**

**Category**  
Class

**Inheritance Path**

**Definition File**  
ilconcert/iloanyset.h

**Summary**  
The array class of the set variable class IloAnySetVar.

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloAnySetVarArray(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySetVarArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnySetVarArray(const IloEnv, IloInt, const IloAnySetVar, const IloAnySetVar v1...)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>add(IloInt, const IloAnySetVar)</td>
</tr>
<tr>
<td>public void</td>
<td>add(const IloAnySetVar)</td>
</tr>
<tr>
<td>public void</td>
<td>add(const IloAnySetVarArray)</td>
</tr>
<tr>
<td>public IloAnySetVar</td>
<td>operator<a href="IloInt"></a></td>
</tr>
<tr>
<td>public IloAnySetVar &amp;</td>
<td>operator<a href="IloInt"></a></td>
</tr>
</tbody>
</table>

**Inherited methods from IloExtractableArray**

IloExtractableArray::add, IloExtractableArray::add,
IloExtractableArray::add, IloExtractableArray::endElements
Description
For each basic type, Concert Technology defines a corresponding array class. IloAnySetVarArray is the array class of the set variable class for enumerated values (IloAnySetVar) in a model.

Instances of IloAnySetVarArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added or removed from the array.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

For more information on arrays, see the concept Arrays.

See Also
IloAnySetVar, IloModel, operator

Constructors

public IloAnySetVarArray(IloDefaultArrayI * i)
This constructor creates an empty extensible array of set variables, where each set is a set of enumerated values. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloAnySetVarArray(const IloEnv env, IloInt n)
This constructor creates an extensible array of n set variables, where each set is a set of enumerated values. Initially, the n elements are empty handles.

public IloAnySetVarArray(const IloEnv env, IloInt n, const IloAnySetVar v0, const IloAnySetVar v1...)
This constructor creates an extensible array of n set variables, as indicated by v0, v1, etc.

Methods

public void add(IloInt more, const IloAnySetVar x)
This member function appends x to the invoking array multiple times. The parameter more indicates how many times.

public void add(const IloAnySetVar x)
This member function appends x to the invoking array.

public void add(const IloAnySetVarArray array)
This member function appends the elements in array to the invoking array.
public IloAnySetVar operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i. Concert Technology uses the const operator

IloAnySetVar operator[](IloInt i) const;

on const arrays.

public IloAnySetVar & operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.
IloAnyTernaryPredicate

Category: Class

InheritancePath

### Definition File
ilconcert/ilotupleset.h

### Summary
A class to define ternary predicates on objects in a model.

#### Constructor Summary
<table>
<thead>
<tr>
<th>Public</th>
<th>IloAnyTernaryPredicate()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloAnyTernaryPredicate(IloAnyTernaryPredicateI *)</td>
</tr>
</tbody>
</table>

#### Method Summary
<table>
<thead>
<tr>
<th>Public</th>
<th>IloAnyTernaryPredicateI * getImpl()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloBool isTrue(const IloAny, const IloAny, const IloAny)</td>
</tr>
<tr>
<td>Public</td>
<td>void operator=(const IloAnyTernaryPredicateI &amp;)</td>
</tr>
</tbody>
</table>

### Description
This class makes it possible for you to define ternary predicates operating on arbitrary objects in a model. A predicate is an object with a member function (such as `isTrue`) that checks whether or not a property is satisfied by an ordered set of (pointers to) objects. A ternary predicate checks an ordered set of three objects.

### Defining a New Class of Predicates
Predicates, like other Concert Technology objects, depend on two classes: a handle class, `IloAnyTernaryPredicate`, and an implementation class, such as `IloAnyTernaryPredicateI`, where an object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of an instance of `IloAnyTernaryPredicateI` allocated in a Concert Technology...
environment. As a Concert Technology user, you will be working primarily with handles.

If you define a new class of predicates yourself, you must define its implementation class together with the corresponding virtual member function isTrue, as well as a member function that returns an instance of the handle class IloAnyTernaryPredicate.

**Arity**

As a developer, you can use predicates in Concert Technology applications to define your own constraints that have not already been predefined in Concert Technology. In that case, the *arity* of the predicate (that is, the number of constrained variables involved in the predicate, and thus the size of the array that the member function isTrue must check) must be three.

**See Also**

IloTableConstraint

**Constructors**

public IloAnyTernaryPredicate()

This constructor creates an empty ternary predicate. In other words, the predicate is an empty handle with a null handle pointer. You must assign the elements of the predicate before you attempt to access it, just as you would any other pointer. Any attempt to access it before this assignment will throw an exception (an instance of IloSolver::SolverErrorException).

public IloAnyTernaryPredicate(IloAnyTernaryPredicateI * impl)

This constructor creates a handle object (an instance of the class IloAnyTernaryPredicate) from a pointer to an implementation object (an instance of the implementation class IloAnyTernaryPredicateI).

**Methods**

public IloAnyTernaryPredicateI * getImpl()

This member function returns a pointer to the implementation object of the invoking handle.

public IloBool isTrue(const IloAny val1, const IloAny val2, const IloAny val3)

This member function returns IloTrue if the values val1, val2, and val3 make the invoking ternary predicate valid. It returns IloFalse otherwise.

public void operator=(const IloAnyTernaryPredicate & h)

This assignment operator copies h into the invoking predicate by assigning an address to the handle pointer of the invoking object. That address is the location of the implementation object of the argument h. After execution of this operator, both the invoking predicate and h point to the same implementation object.
IloAnyTupleSet

Category    Class
InheritancePath

Definition File  ilconcert/ilotupleset.h
Summary        A class to represent set of enumerated tuples in a model.

Constructor Summary

<table>
<thead>
<tr>
<th>public</th>
<th>IloAnyTupleSet()</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloAnyTupleSet(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloAnyTupleSet(const IloEnv, const IloInt)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>public IloBool</th>
<th>add(const IloAnyArray)</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isIn(const IloAnyArray)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>remove(const IloAnyArray)</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractable

IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId, IloExtractable::getImpl, IloExtractable::getName, IloExtractable::getObject, IloExtractable::setName, IloExtractable::setObject

Description

A tuple is an ordered set of values represented by an array. A set of enumerated tuples in a model is represented by an instance of IloAnyTupleSet. That is, the elements of a
tuple set are tuples of enumerated values (such as pointers). The number of values in a tuple is known as the arity of the tuple, and the arity of the tuples in a set is called the arity of the set. (In contrast, the number of tuples in the set is known as the cardinality of the set.)

As a handle class, IloAnyTupleSet manages certain set operations efficiently. In particular, elements can be added to such a set. It is also possible to search a given set with the member function isIn to see whether or not the set contains a given element.

In addition, a set of tuples can represent a constraint defined on a constrained variable, either as the set of allowed combinations of values of the constrained variable on which the constraint is defined, or as the set of forbidden combinations of values.

There are a few conventions governing tuple sets:

- When you create the set, you must specify the arity of the tuple-elements it contains.
- You use the member function IloAnyTupleSet::add to add tuples to the set. You may add tuples to the set in a model; you cannot add tuples to an instance of this class during a search, nor inside a constraint, nor inside a goal.

Concert Technology will throw an exception (an instance of IloSolver::SolverErrorException) if you attempt:

- to add a tuple with a different number of variables from the arity of the set;
- to search for a tuple with an arity different from the set arity.

You do not have to worry about memory allocation. If you respect these conventions, Concert Technology manages allocation and de-allocation transparently for you.

See Also
IlcAnyTupleSet documented in the ILOG Solver Reference Manual, IloAnyTupleSetIterator, IloTableConstraint, IloExtractable

Constructors
public IloAnyTupleSet()
This constructor creates an empty handle. You must initialize it before you use it.

public IloAnyTupleSet(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloAnyTupleSet(const IloEnv env, const IloInt arity)
This constructor creates a set of tuples (an instance of the class IloAnyTupleSet) with the arity indicated by arity.

Methods
public IloBool add(const IloAnyArray tuple)
This member function adds a tuple represented by the array tuple to the invoking set. If you attempt to add an element that is already in the set, that element will not be added.
again. Added elements are not copied; that is, there is no memory duplication. Concert Technology will throw an exception if the size of the array is not equal to the arity of the invoking set. You may use this member function to add tuples to the invoking set in a model; you may not add tuples in this way during a search, inside a constraint, or inside a goal. For those purposes, see IlcAnyTupleSet, documented in the ILOG Solver Reference Manual.

public ImplClass * getImpl()

This member function returns a pointer to the implementation object of the invoking handle.

public IloBool isIn(const IloAnyArray tuple)

This member function returns IloTrue if tuple belongs to the invoking set. Otherwise, it returns IloFalse. Concert Technology will throw an exception if the size of the array is not equal to the arity of the invoking set.

public IloBool remove(const IloAnyArray tuple)

This member function removes tuple from the invoking set in a model. You may use this member function to remove tuples from the invoking set in a model; you may not remove tuples in this way during a search, inside a constraint, or inside a goal. For those purposes, see IlcAnyTupleSet documented in the ILOG Solver Reference Manual.
**IloAnyTupleSetIterator**

**Category**  Class

**InheritancePath**

- **Definition File**  ilconcert/ilotupleset.h

**Summary**  An iterator to traverse the elements of a finite set of tuples of enumerated values.

### Constructor Summary

| public | IloAnyTupleSetIterator(const IloEnv, IloAnyTupleSet) |

### Method Summary

| public IloBool | ok() |
| public IloAnyArray | operator *() |
| public void | operator++() |

**Description**  An instance of the class `IloAnyTupleSetIterator` is an iterator that traverses the elements of a finite set of tuples of enumerated values (instance of `IloAnyTupleSet`).

**See Also**  IlcAnyTupleSet documented in the ILOG Solver Reference Manual, IloAnyTupleSet

**Constructors**

- `public IloAnyTupleSetIterator(const IloEnv env, IloAnyTupleSet tSet)`

  This constructor creates an iterator associated with `tSet` to traverse its elements.

**Methods**

- `public IloBool ok()`

  This member function returns `IloTrue` if there is a current element and invoking iterator points to it. Otherwise, it returns `IloFalse`. 
To traverse the elements of a finite set of tuples, use the following code:

```cpp
IloAnyTupleSet val;
for(IloAnyTupleSetIterator iter(env, tset); iter.ok(); ++iter){
    val = *iter;
    // do something with val
}
```

**public IloAnyArray operator *()**

This operator returns the current element, the one to which the invoking iterator points.

**public void operator++()**

This operator advances the iterator to point to the next tuple in the set.
IloAnyVar

Category          Class

InheritancePath

Definition File    ilconcert/iloany.h

Summary           A class to represent an enumerated variable.

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
<tr>
<td>public IloAnyVar()</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>IloAnyVar(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>IloAnyVar(const IloEnv, const IloAnyArray, const char *)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass * getImpl()</td>
</tr>
<tr>
<td>public void     getPossibleValues(IloAnyArray)</td>
</tr>
<tr>
<td>public void     setPossibleValues(const IloAnyArray)</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractable

IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId, IloExtractable::getImpl, IloExtractable::getName, IloExtractable::getObject, IloExtractable::setName, IloExtractable::setObject

Description          An instance of this class offers a convenient way to represent an enumerated variable in Concert Technology.
Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

**See Also**

IloAny, IloAnyVarArray, IloExtractable, IloModel

**Constructors**

`public IloAnyVar()`  
This constructor creates an empty handle. You must initialize it before you use it.

`public IloAnyVar(ImplClass * impl)`  
This constructor creates a handle object from a pointer to an implementation object.

`public IloAnyVar(const IloEnv env,  
                  const IloAnyArray array,  
                  const char * name)`  
This constructor creates a constrained enumerated variable from the values in `array` and makes the variable part of the environment indicated by `env`. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.

**Methods**

`public ImplClass * getImpl()`  
This member function returns a pointer to the implementation object of the invoking handle.

`public void getPossibleValues(IloAnyArray values)`  
This member function accesses the possible values of the invoking enumerated variable and puts those values into its parameter `values`.

`public void setPossibleValues(const IloAnyArray values)`  
This member function sets `values` as the domain of the invoking enumerated variable.
IloAnyVarArray

Category  Class

InheritancePath

Definition File  ilconcert/iloany.h

Summary  A class to represent an array of enumerated variables.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAnyVarArray(IloDefaultArrayI *)</td>
<td></td>
</tr>
<tr>
<td>public IloAnyVarArray(const IloEnv, IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloAnyVarArray(const IloEnv, IloInt, const IloAnyVar,</td>
<td></td>
</tr>
<tr>
<td>const IloAnyVar,</td>
<td></td>
</tr>
<tr>
<td>const IloAnyVar v1...)</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void add(IloInt, const IloAnyVar)</td>
<td></td>
</tr>
<tr>
<td>public void add(const IloAnyVar)</td>
<td></td>
</tr>
<tr>
<td>public void add(const IloAnyVarArray)</td>
<td></td>
</tr>
<tr>
<td>public IloAnyVar operator<a href="IloInt"></a></td>
<td></td>
</tr>
<tr>
<td>public IloAnyVar &amp; operator<a href="IloInt"></a></td>
<td></td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractableArray

IloExtractableArray::add, IloExtractableArray::add,
IloExtractableArray::add, IloExtractableArray::endElements
IloAnyVarArray

Description
For each basic type, Concert Technology defines a corresponding array class.  
IloAnyVarArray is the array class of the enumerated variable class (IloAnyVar) 
for a model. The parent class for IloAnyVarArray is the class 
IloExtractableArray.

Instances of IloAnyVarArray are extensible. That is, you can add more elements to 
such an array. References to an array change whenever an element is added or removed 
from the array.

Most member functions in this class contain assert statements. For an explanation of 
the macro NDEBUG (a way to turn on or turn off these assert statements), see the 
concept Assert and NDEBUG.

See Also
IloAnyVar, IloModel, operator

Constructors
public IloAnyVarArray(IloDefaultArrayI * i)

This constructor creates an empty extensible array of enumerated variables. You cannot 
create instances of the undocumented class IloDefaultArrayI. As a parameter in 
this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter 
in functions and member functions that accept an array as a parameter.

public IloAnyVarArray(const IloEnv env, 
IloInt n)

This constructor creates an extensible array of n enumerated variables. Initially, the n 
elements are empty handles.

public IloAnyVarArray(const IloEnv env, 
IloInt n, 
const IloAnyVar v0, 
const IloAnyVar v1...)

This constructor creates an extensible array of n enumerated variables, as indicated by 
v0, v1, etc.

Methods
public void add(IloInt more, 
const IloAnyVar x)

This member function appends x to the invoking array multiple times. The parameter 
more indicates how many times.

public void add(const IloAnyVar x)

This member function appends x to the invoking array.

public void add(const IloAnyVarArray array)

This member function appends the elements in array to the invoking array.

public IloAnyVar operator[](IloInt i)
This operator returns a reference to the object located in the invoking array at the position indicated by the index i. Concert Technology uses the const operator

\[
\text{IloAnyVar operator[]} (\text{IloInt i}) \text{ const;}
\]

on const arrays.

public \text{IloAnyVar & \text{operator}[]} (\text{IloInt i})

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.
IloArcCos

Category: Global Function

Definition File: ilconcert/iloexpression.h

Synopsis:
- public IlonExprArg IloArcCos(const IlonExprArg arg)
- public Ilon IloCos(Ilon val)
- public Ilon IloSin(Ilon val)
- public Ilon IloTan(Ilon val)
- public Ilon IloArcCos(Ilon val)
- public Ilon IloArcSin(Ilon val)
- public Ilon IloArcTan(Ilon val)
- public IlonExprArg IloSin(const IlonExprArg arg)
- public IlonExprArg IloCos(const IlonExprArg arg)
- public IlonExprArg IloTan(const IlonExprArg arg)
- public IlonExprArg IloArcSin(const IlonExprArg arg)
- public IlonExprArg IloArcTan(const IlonExprArg arg)

Description:
Concert Technology offers predefined functions that return an expression from a trigonometric function on an expression. These predefined functions also return a numeric value from a trigonometric function on a numeric value as well.

Programming Hint:
If you want to manipulate constrained floating-point expressions in degrees, we strongly recommend that you call the trigonometric functions on variables expressed in radians and then convert the results to degrees (rather than declaring the constrained floating-point expressions in degrees and then converting them to radians to call the trigonometric functions).

The reason for that advice is that the method we recommend gives more accurate results in the context of the usual floating-point pitfalls.
IloArray

Category Class

InheritancePath

Definition File ilconcert/iloenv.h

Summary A template to create classes of arrays for elements of a given class.

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloArray(IloEnv, IloInt)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>add(IloArray&lt; X &gt;)</td>
</tr>
<tr>
<td>public void</td>
<td>add(IloInt, X)</td>
</tr>
<tr>
<td>public void</td>
<td>add(X)</td>
</tr>
<tr>
<td>public void</td>
<td>clear()</td>
</tr>
<tr>
<td>public void</td>
<td>end()</td>
</tr>
<tr>
<td>public IloEnv</td>
<td>getEnv()</td>
</tr>
<tr>
<td>public IloInt</td>
<td>getSize()</td>
</tr>
<tr>
<td>public X &amp;</td>
<td>operator<a href="IloInt"></a></td>
</tr>
<tr>
<td>public const X &amp;</td>
<td>operator<a href="IloInt"></a></td>
</tr>
<tr>
<td>public void</td>
<td>remove(IloInt, IloInt)</td>
</tr>
</tbody>
</table>

Description This C++ template creates a class of arrays for elements of a given class. In other words, you can use this template to create arrays of Concert Technology objects; you can also use this template to create arrays of arrays (that is, multidimensional arrays).

In its synopsis, X represents a class, x is an instance of the class X. This template creates the array class (IloArrayX) for any class in Concert Technology, including classes with names in the form IloXArray, such as IloExtractableArray. Concert
Technology predefines the array classes listed here as See Also. The member functions defined by this template are documented in each of those predefined classes.

The classes you create in this way consist of extensible arrays. That is, you can add elements to the array as needed.

**Deleting Arrays**

The member function `end` created by this template deletes only the array; the member function does not delete the elements of the array.

**Copying Arrays**

Like certain other Concert Technology classes, a class of arrays created by `IloArray` is a handle class corresponding to an implementation class. In other words, an instance of an `IloArray` class is a handle pointing to a corresponding implementation object.

More than one handle may point to the same implementation object. As long as you do not modify any of the handles, each of them will still point to the same implementation object.

However, if you modify an array, then Concert Technology considers whether the array is the sole handle pointing to its implementation object or whether the array is one of many handles pointing to the same implementation object.

If the array is the sole handle pointing to its implementation object, Concert Technology modifies the array.

If the array is one handle among many pointing to the same implementation object, then Concert Technology first creates a copy of the implementation object corresponding to the array (the handle you are modifying) and then modifies the copy. Consequently, the other handles continue to point to the original, unmodified implementation object.

In other words, when Concert Technology copies an array, it uses a lazy copy mechanism.

**Input and Output of Multidimensional Arrays**

The template `operator >>` makes it possible to read numerical values from a file in the format `[x, y, z, ...]` where `x, y, z` are the results of the `operator >>` for class `X`. Class `X` must provide a default constructor for `operator >>` to work. That is, the statement `X x;` must work for `X`. This input operator is limited to numerical values.

Likewise, the template `operator <<` makes it possible to write to a file in the format `[x, y, z, ...]` where `x, y, z` are the results of the `operator <<` for class `X`. (This output operator is not limited to numerical values, as the input operator is.)

These two operators make it possible to read and write multidimensional arrays of numerical values like this:
IloArray\langle IloArray\langle IloIntArray \rangle \rangle >

(Notice the space between \texttt{>} \texttt{>} at the end of that statement. It is necessary in C++.)

However, there is a practical limit of four on the number of dimensions supported by the input operator for reading multidimensional arrays. This limit is due to the inability of certain C++ compilers to support templates correctly. Specifically, you can read input by means of the input operator for multidimensional arrays of one, two, three, or four dimensions. There is no such limit on the number of dimensions with respect to the output operator for multidimensional arrays.

See Also

\begin{itemize}
\item IloAnyArray,
\item IloAnySetVarArray,
\item IloAnyVarArray,
\item IloBoolArray,
\item IloBoolVarArray,
\item IloConstraintArray,
\item IloExprArray,
\item IloExtractableArray,
\item IloFloatArray,
\item IloFloatVarArray,
\item IloIntArray,
\item IloIntVarArray,
\item IloNumColumnArray,
\item IloNumVarArray,
\item IloRangeArray,
\item IloSemiContVarArray,
\item IloSolutionArray,
\item IloSOS1Array,
\item IloSOS2Array
\end{itemize}

Constructors

\begin{verbatim}
public IloArray(IloEnv env, 
  IloInt max)
\end{verbatim}

This constructor creates an array of \texttt{max} elements, all of which are empty handles.

Methods

\begin{verbatim}
public void add(IloArray< X > ax)
\end{verbatim}

This member function appends the elements in \texttt{ax} to the invoking array.

\begin{verbatim}
public void add(IloInt more, 
  X x)
\end{verbatim}

This member function appends \texttt{x} to the invoking array multiple times. The parameter \texttt{more} indicates how many times.

\begin{verbatim}
public void add(X x)
\end{verbatim}

This member function appends \texttt{x} to the invoking array.

\begin{verbatim}
public void clear()
\end{verbatim}

This member function removes all the elements from the invoking array. In other words, it produces an empty array.

\begin{verbatim}
public void end()
\end{verbatim}

This member function first removes the invoking extractable object from all other extractable objects where it is used (such as a model, ranges, etc.) and then deletes the invoking extractable object. That is, it frees all the resources used by the invoking object. After a call to this member function, you cannot use the invoking extractable object again.
public IloEnv getEnv()

This member function returns the environment where the invoking array was created. The elements of the invoking array belong to the same environment.

public IloInt getSize()

This member function returns an integer indicating the size of the invoking array. An empty array has size 0 (zero).

public X & operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.

public const X & operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i. Concert Technology uses the const operator

IloArray operator[](IloInt i) const;

on const arrays.

public void remove(IloInt first,
IloInt nb)

This member function removes elements from the invoking array. It begins removing elements at the index indicated by first, and it removes nb elements (nb = 1 by default).
IloBarrier

Category       Class
InheritancePath

Definition File ilconcert/ilothread.h
Summary        A system class to synchronize threads at a specified number.

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int</td>
</tr>
</tbody>
</table>

Description
The class IloBarrier provides synchronization primitives adapted to Concert Technology. A barrier, an instance of this class, serves as a rendezvous for a specific number of threads. After you create a barrier for n threads, the first n-1 threads to reach that barrier will be blocked. The nth thread to arrive at the barrier completes the synchronization and wakes up the n-1 threads already waiting at that barrier. When the nth thread arrives, the barrier resets itself. Any other thread that arrives at this point is blocked and will participate in a new barrier of size n.

See ILOUSEMT for details about the compilation macro to use with instances of this class.

Note: The class IloBarrier has nothing to do with the ILOG CPLEX barrier optimizer.

System Class
IloBarrier is a system class.
Most Concert Technology classes are actually handle classes whose instances point to objects of a corresponding implementation class. For example, instances of the Concert Technology class `IloNumVar` are handles pointing to instances of the implementation class `IloNumVarI`. Their allocation and de-allocation in a Concert Technology environment are managed by an instance of `IloEnv`.

However, system classes, such as `IloBarrier`, differ from that Concert Technology pattern. `IloBarrier` is an ordinary C++ class. Its instances are allocated on the C++ heap.

Instances of `IloBarrier` are not automatically de-allocated by a call to `IloEnv::end`. You must explicitly destroy instances of `IloBarrier` by means of a call to the delete operator (which calls the appropriate destructor) when your application no longer needs instances of this class.

Furthermore, you should not allocate—neither directly nor indirectly—any instance of `IloBarrier` in a Concert Technology environment because the destructor for that instance of `IloBarrier` will never be called automatically by `IloEnv::end` when it cleans up other Concert Technology objects in that Concert Technology environment.

For example, it is not a good idea to make an instance of `IloBarrier` part of a conventional Concert Technology model allocated in a Concert Technology environment because that instance will not automatically be de-allocated from the Concert Technology environment along with the other Concert Technology objects.

### De-allocating Instances of IloBarrier

Instances of `IloBarrier` differ from the usual Concert Technology objects because they are not allocated in a Concert Technology environment, and their de-allocation is not managed automatically for you by `IloEnv::end`. Instead, you must explicitly destroy instances of `IloBarrier` by calling the delete operator when your application no longer needs those objects.

### See Also

- `IloCondition`
- `IloFastMutex`
- `ILOUSEMT`

### Constructors

```java
public IloBarrier(int count)
```

This constructor creates an instance of `IloBarrier` of size `count` and allocates it on the C++ heap (not in a Concert Technology environment).

### Methods

```java
public int wait()
```

The first `count-1` calls to this member function block the calling thread. The last call (that is, the call numbered `count`) wakes up all the `count-1` waiting threads. Once a thread has been woken up, it leaves the barrier. When a thread leaves the barrier (that is, when it returns from the `wait` call), it will return either 1 (one) or 0 (zero). If the thread returns 0, the barrier is not yet empty. If the thread returns 1, it was the last thread at the barrier.
A non-empty barrier contains blocked threads or exiting threads.
IloBaseEnvMutex

Category: Class

InheritancePath: 

Definition File: ilconcert/iloenv.h

Summary: A class to initialize multithreading in an application.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td>public virtual void lock()</td>
</tr>
<tr>
<td>unlock</td>
<td>public virtual void unlock()</td>
</tr>
</tbody>
</table>

Description: An instance of this base class in the function IloInitMT initializes multithreading in a Concert Technology application. For a general purpose mutex, see the class IloFastMutex.

See Also: IloFastMutex, IloInitMT

Methods

- public virtual void lock()
  
  This member function locks a mutex.

- public virtual void unlock()
  
  This member function unlocks a mutex.
IloBool

Category: Type Definition

Definition File: ilconcert/ilosys.h

Synopsis: IloInt IloBool

Description: This type definition represents Boolean values in Concert Technology. Those values are IloTrue and IloFalse. Booleans are, in fact, integers of type IloInt. IloFalse is 0 (zero), and IloTrue is 1 (one). This type anticipates the built-in bool type proposed for standard C++. By using this type, you can be sure that the Concert Technology components of your application will port in this respect without source changes across different hardware platforms.

See Also: IloBoolArray, IloInt, IloModel, IloNum
**IloBoolAbstraction**

**Category**  
Global Function

**Definition File**  
ilconcert/ilomodel.h

**Synopsis**  
public IloConstraint IloBoolAbstraction(const IloEnv env,  
const IloBoolVarArray y,  
const IloIntVarArray x,  
const IloIntArray values)

class IloConstraint IloBoolAbstraction(const IloEnv env,  
const IloBoolVarArray avars,  
const IloAnyVarArray vars,  
const IloAnyArray values)

**Summary**  
A function to create a constraint that abstracts an array of Boolean variables.

**Description**  
This function creates and returns a constraint that abstracts an array of constrained Boolean variables in a model. It differs from IloAbstraction in that its y-array is an array of Boolean variables (also known as 0-1 variables or binary variables). Like IloAbstraction, for each element \(x[i]\), there is a variable \(y[i]\) corresponding to the abstraction of \(x[i]\) with respect to an array of values. That is,

\[
\begin{align*}
x[i] = v \text{ with } v \text{ in values if and only if } y[i] = \text{IloTrue}; \\
x[i] = v \text{ with } v \text{ not in values if and only if } y[i] = \text{IloFalse}.
\end{align*}
\]

This constraint maintains a many-to-one mapping that makes it possible to define constraints that impinge only on a particular set of values from the domains of constrained variables.

**Example**  
For simplicity, assume that an array \(x\) consists of three elements with the domains \([3], [4], \text{and } [5]\). Assume that the values we are interested in are \([4, 12, 16]\). Then IloBoolAbstraction produces the elements of the array \(y\), like this:

<table>
<thead>
<tr>
<th>(x)</th>
<th>($)</th>
<th>Values</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>12</td>
<td>IloFalse</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>IloTrue</td>
<td>IloFalse</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>16</td>
<td>IloFalse</td>
</tr>
</tbody>
</table>
Adding a Constraint to a Model, Extracting a Model for an Algorithm

In order for the constraint returned by `IloBoolAbstraction` to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.

Exceptions

If the arrays x and y are not the same size, this function throws the exception `IloBoolAbstraction::InvalidArraysException`. 

IloBoolArray

**Category**
Class

**InheritancePath**

![Inheritance Path Diagram]

**Definition File**
ilconcert/iloenv.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBoolArray(IloArrayI *)</td>
</tr>
<tr>
<td>public IloBoolArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public IloBoolArray(const IloEnv, IloInt, const IloBool)</td>
</tr>
<tr>
<td>public IloBoolArray(const IloEnv, IloInt, const IloBool, const IloBool v1...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void add(IloInt, const IloBool)</td>
</tr>
<tr>
<td>public void add(const IloBool)</td>
</tr>
<tr>
<td>public void add(const IloBoolArray)</td>
</tr>
</tbody>
</table>

**Inherited methods from IloIntArray**

IloIntArray::toNumArray
**IloBoolArray**

**Description**

IloBoolArray is the array class of the basic Boolean class for a model. It is a handle class. The implementation class for IloBoolArray is the undocumented class IloBoolArrayI.

Instances of IloBoolArray are extensible. (They differ from instances of IloBoolArray in this respect.) References to an array change whenever an element is added to or removed from the array.

For each basic type, Concert Technology defines a corresponding array class. That array class is a handle class. In other words, an object of that class contains a pointer to another object allocated in a Concert Technology environment associated with a model. Exploiting handles in this way greatly simplifies the programming interface since the handle can then be an automatic object: as a developer using handles, you do not have to worry about memory allocation.

As handles, these objects should be passed by value, and they should be created as automatic objects, where “automatic” has the usual C++ meaning.

Member functions of a handle class correspond to member functions of the same name in the implementation class.

**Assert and NDEBUG**

Most member functions of the class IloBoolArray are inline functions that contain an assert statement. This statement checks that the handle pointer is non-null. These statements can be suppressed by the macro NDEBUG. This option usually reduces execution time. The price you pay for this choice is that attempts to access through null pointers are not trapped and usually result in memory faults.

**See Also**

IloBool

**Constructors**

public **IloBoolArray**(IloArrayI * i)

This constructor creates an array of Boolean values from an implementation object.

public **IloBoolArray**(const IloEnv env, IloInt n)

This constructor creates an array of n Boolean values for use in a model in the environment indicated by env. By default, its elements are empty handles.

public **IloBoolArray**(const IloEnv env, IloInt n, const IloBool v0)

This constructor creates an array of n Boolean values.

public **IloBoolArray**(const IloEnv env, IloInt n, const IloBool v0, const IloBool v1...)
This constructor creates an array of \( n \) Boolean values; the elements of the new array take the corresponding values: \( v_0, v_1, \ldots, v(n-1) \).

### Methods

**public void add(IloInt more, const IloBool x)**

This member function appends \( x \) to the invoking array of Boolean values; it appends \( x \) \( \text{more times} \).

**public void add(const IloBool x)**

This member function appends the value \( x \) to the invoking array.

**public void add(const IloBoolArray x)**

This member function appends the values in the array \( x \) to the invoking array.
IloBoolVar

**Category**  
Class

**InheritancePath**

**Definition File**  
ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloBoolVar(IloEnv, IloInt, const char *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloBoolVar(IloEnv, const char *)</td>
</tr>
<tr>
<td>Public</td>
<td>IloBoolVar(const IloAddNumVar &amp;, const char *)</td>
</tr>
</tbody>
</table>

**Inherited methods from IloIntVar**

- IloIntVar::getImpl
- IloIntVar::getLB
- IloIntVar::getMax
- IloIntVar::getMin
- IloIntVar::getUB
- IloIntVar::setBounds
- IloIntVar::setLB
- IloIntVar::setMax
- IloIntVar::setMin
- IloIntVar::setPossibleValues
- IloIntVar::setUB

**Inherited methods from IloIntExprArg**
Description

An instance of this class represents a constrained Boolean variable in a Concert Technology model. Boolean variables are also known as binary decision variables. They can assume the values 0 (zero) or 1 (one).

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

What Is Extracted


An instance of IloBoolVar is extracted by IloCplex (documented in the ILOG CPLEX Reference Manual) as a column representing a numeric variable of type Bool with bounds as indicated by IloBoolVar.

See Also

IloIntVar, IloNumVar

Constructors

public IloBoolVar(IloEnv env,
                   IloInt min,
                   IloInt max,
                   const char * name)

This constructor creates a Boolean variable and makes it part of the environment env. By default, the Boolean variable assumes a value of 0.0 (zero) or 1.0. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.
public IloBoolVar (IloEnv env,  
   const char * name)

This constructor creates a Boolean variable and makes it part of the environment env. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.

public IloBoolVar (const IloAddNumVar & column,  
   const char * name)

This constructor creates an instance of IloBoolVar like this:

IloNumVar (column, 0.0, 1.0, ILOBOOL, name);
**IloBoolVarArray**

**Category**  
Class

**InheritancePath**

```
  IloExtractableArray  
  IloIntExprArray  
  IloIntVarArray  
  IloBoolVarArray
```

**Definition File**  
`ilconcert/iloexpression.h`

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloBoolVarArray</td>
<td>(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>IloBoolVarArray</td>
<td>(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>IloBoolVarArray</td>
<td>(const IloEnv, const IloNumColumnArray)</td>
</tr>
<tr>
<td>IloBoolVarArray</td>
<td>(const IloEnv, IloInt, const IloBoolVar, const IloBoolVar, ...)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>add(IloInt, const IloBoolVar)</td>
</tr>
<tr>
<td>void</td>
<td>add(const IloBoolVar)</td>
</tr>
<tr>
<td>void</td>
<td>add(const IloBoolVarArray)</td>
</tr>
<tr>
<td>IloBoolVar</td>
<td>operator<a href="IloInt"></a></td>
</tr>
<tr>
<td>IloBoolVar &amp;</td>
<td>operator<a href="IloInt"></a></td>
</tr>
</tbody>
</table>
IloBoolVarArray

Description
For each basic type, Concert Technology defines a corresponding array class. IloBoolVarArray is the array class of the Boolean variable class for a model. It is a handle class.

Instances of IloBoolVarArray are extensible.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also
IloBoolVar

Constructors
public IloBoolVarArray(IloDefaultArrayI * i)
This constructor creates an empty extensible array of Boolean variables.

public IloBoolVarArray(const IloEnv env, IloInt n)
This constructor creates an extensible array of n Boolean variables.

public IloBoolVarArray(const IloEnv env, const IloNumColumnArray columnarray)
This constructor creates an extensible array of Boolean variables from a column array.

Inherited methods from IloIntVarArray
IloIntVarArray::add, IloIntVarArray::add, IloIntVarArray::add, IloIntVarArray::operator(), IloIntVarArray::operator[], IloIntVarArray::operator[], IloIntVarArray::toNumVarArray

Inherited methods from IloIntExprArray
IloIntExprArray::add, IloIntExprArray::add, IloIntExprArray::add, IloIntExprArray::operator(), IloIntExprArray::operator[], IloIntExprArray::operator[]

Inherited methods from IloExtractableArray
IloExtractableArray::add, IloExtractableArray::add, IloExtractableArray::add, IloExtractableArray::endElements
public **IloBoolVarArray** (const IloEnv env,
   IloInt n,
   const IloBoolVar v0,
   const IloBoolVar v1,
   ...)

This constructor creates an extensible array of \( n \) Boolean variables, the elements of the new array take the corresponding values: \( v_0, v_1, \ldots, v_{(n-1)} \).

**Methods**

public void **add** (IloInt more,
   const IloBoolVar x)

This member function appends \( x \) to the invoking array of Boolean variables; it appends \( x \) more times.

public void **add** (const IloBoolVar x)

This member function appends the value \( x \) to the invoking array.

public void **add** (const IloBoolVarArray x)

This member function appends the variables in the array \( x \) to the invoking array.

public IloBoolVar **operator[]** (IloInt i)

This operator returns a reference to the extractable object located in the invoking array at the position indicated by the index \( i \). Concert Technology uses the **const** operator IloBoolVar **operator[]** (IloInt i) const;

on const arrays.

public IloBoolVar & **operator[]** (IloInt i)

This operator returns a reference to the extractable object located in the invoking array at the position indicated by the index \( i \).
**IloBox**

**Category** Class

**InheritancePath**

**Definition File** ilconcert/ilobox.h

**Summary** A class of multidimensional boxes for multidimensional placement problems.

---

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBox()</td>
<td></td>
</tr>
<tr>
<td>public IloBox(ImplClass *)</td>
<td></td>
</tr>
<tr>
<td>public IloBox(const IloEnv, IloInt,</td>
<td></td>
</tr>
<tr>
<td>const IloIntVarArray, const IloIntArray)</td>
<td></td>
</tr>
</tbody>
</table>

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloConstraint contains(IloBox)</td>
<td></td>
</tr>
<tr>
<td>public IloInt getDimensions()</td>
<td></td>
</tr>
<tr>
<td>public ImplClass * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public IloConstraint getNotOverlapConstraint()</td>
<td></td>
</tr>
<tr>
<td>public IloIntVar getOrigin(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloInt getSize(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloConstraint notOverlapInDimension(IloBox, IloBox, IloInt)</td>
<td></td>
</tr>
</tbody>
</table>
Instances of the class `IloBox` are multidimensional boxes that appear in multidimensional placement problems. To solve packing or placement problems, you may need to be able to place boxes within a given container. In such a situation, both the boxes to place and the container to hold them are instances of the class `IloBox`.

To specify the containment relation that a given container holds a given box, use the member function `contains`.

**Description**

<table>
<thead>
<tr>
<th>public <code>IloConstraint</code></th>
<th><code>overlapInDimension(IloBox, IloBox, IloInt)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>public <code>IloConstraint</code></td>
<td><code>precedenceInDimension(IloBox, IloBox, IloInt)</code></td>
</tr>
</tbody>
</table>

**Inherited methods from `IloConstraint`**

- `IloConstraint::getImpl`

**Inherited methods from `IloIntExprArg`**

- `IloIntExprArg::getImpl`

**Inherited methods from `IloNumExprArg`**

- `IloNumExprArg::getImpl`

**Inherited methods from `IloExtractable`**

- `IloExtractable::end`, `IloExtractable::getEnv`, `IloExtractable::getId`, `IloExtractable::getImpl`, `IloExtractable::getName`, `IloExtractable::getObject`, `IloExtractable::setName`, `IloExtractable::setObject`
See Also

Constructors

public IloBox()
This constructor creates an empty handle. You must initialize it before you use it.

public IloBox(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloBox(const IloEnv env,
IloInt dimensions,
const IloIntVarArray origin,
const IloIntArray size)
This constructor creates a box according to the specifications passed in the parameters.
The parameter dimensions indicates the number of dimensions the box has. The arrays origin and size must contain the same number of elements as the number of dimensions of the box. In dimension i, the box extends from origin[i] to origin[i] + size[i]. For example, the statement

IloBox(env, 2, IloNumVarArray (env, 2, 3, 0), IloNumArray(env, 2, 8, 4));

creates a box as shown in this illustration.

Methods

public IloConstraint contains(IloBox box)
This member function creates a constraint that requires the invoking box to contain box. The parameter box should have the same number of dimensions as the invoking box.

public IloInt getDimensions()
This member function returns the number of dimensions of the invoking box.

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.

public IloConstraint getNotOverlapConstraint()
This member function returns a constraint that specifies that none of the boxes contained in the invoking box can overlap along any of their dimensions.

The filter level of this constraint can be set using IloSolver::setFilterLevel. Currently two filter levels are allowed, IlcBasic, the default, and IlcMedium, which is slower but propagates more. You can specify the default filter level of this constraint using the method IloSolver::setDefaultFilterLevel. Refer to the ILOG Solver Reference Manual for details about these methods.

public IloIntVar getOrigin(IloInt dimension)

This member function returns the origin of the invoking box along the dimension indicated by dimension.

public IloInt getSize(IloInt dimension)

This member function returns the length of the invoking box along the dimension indicated by dimension.

public IloConstraint notOverlapInDimension(IloBox box1,
                                         IloBox box2,
                                         IloInt dimension)

This member function returns a constraint that specifies that if box1 and box2 are contained in the invoking box, then they do not overlap along the dimension indicated by dimension.

public IloConstraint overlapInDimension(IloBox box1,
                                        IloBox box2,
                                        IloInt dimension)

This member function returns a constraint that specifies that if box1 and box2 are contained in the invoking box, then they must overlap along the dimension indicated by dimension.

public IloConstraint precedenceInDimension(IloBox box1,
                                           IloBox box2,
                                           IloInt dimension)

If box1 and box2 are contained in the invoking box, then this member function returns a constraint that specifies the precedence of box1 over box2 along the dimension indicated by dimension.
IloCard

Category: Global Function

Definition File: ilconcert/iloanyset.h

Synopsis: public IloIntVar IloCard(IloAnySetVar vars)

Description: This function creates and returns a constrained numeric variable that represents the number of elements in vars. In other words, it constrains the cardinality of a set variable.

For example, to constrain mySet to contain four or more elements, you can use IloCard in the following way:

```cpp
model.add (IloCard (mySet) >= 4);
```
IloCeil

Category  Global Function
Definition File  ilconcert/iloenv.h
Synopsis  public IloNum IloCeil(IloNum val)
Description  This function computes the least integer value not less than val.

Examples:

IloCeil(IloInfinity) is IloInfinity.
IloCeil(-IloInfinity) is -IloInfinity.
IloCeil(0) is 0.
IloCeil(0.4) is 1.
IloCeil(-0.4) is 0.
IloCeil(0.5) is 1.
IloCeil(-0.5) is 0.
IloCeil(0.6) is 1.
IloCeil(-0.6) is 0.
IloCondition

Category       Class
InheritancePath

Definition File    ilconcert/ilothread.h

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloCondition()</td>
<td>Constructor</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void broadcast()</td>
<td>broadcast</td>
</tr>
<tr>
<td>public void notify()</td>
<td>notify</td>
</tr>
<tr>
<td>public void wait(IloFastMutex *)</td>
<td>wait</td>
</tr>
</tbody>
</table>

Description

The class IloCondition provides synchronization primitives adapted to Concert Technology for use in a parallel application.

See ILOUSEMT for details about the compilation macro to use with instances of this class.

An instance of the class IloCondition allows several threads to synchronize on a specific event. In this context, inter-thread communication takes place through signals. A thread expecting a condition of the computation state (say, conditionC) to be true before it executes a treatmentT can wait until the condition is true. When computation reaches a state where conditionC holds, then another thread can signal this fact by notifying a single waiting thread or by broadcasting to all the waiting threads that conditionC has now been met.

The conventional template for waiting on conditionC looks like this:

```c
mutex.lock();
while (conditionC does not hold)
    condition.wait(&mutex);
doTreatmentT();
```
mutex.unlock();

That template has the following properties:

1. The whole fragment is a critical section so that the evaluation of \texttt{conditionC} is protected. (Indeed, it would be unsafe to evaluate \texttt{conditionC} while at the same time another thread modifies the computation state and affects the truth value of \texttt{conditionC}.) The pair of member functions \texttt{IloFastMutex::lock} and \texttt{IloFastMutex::unlock} delimit the critical section.

2. When a thread enters the \texttt{wait} call, the mutex is automatically unlocked by the system.

3. The loop that repeatedly checks \texttt{conditionC} is essential to the correctness of the code fragment. It protects against the following possibility: between the time that a thread modifies the computation state (so that \texttt{conditionC} holds) and notifies a waiting thread and the moment the waiting thread wakes up, the computation state might have been changed by another thread, and \texttt{conditionC} might very well be false.

4. Upon returning from the \texttt{wait} call, the mutex is locked. The operation of waking up and locking the mutex is atomic. In other words, nothing can happen between the waking and the locking.

**System Class**

\texttt{IloCondition} is a system class.

Most Concert Technology classes are actually handle classes whose instances point to objects of a corresponding implementation class. For example, instances of the Concert Technology class \texttt{IloNumVar} are handles pointing to instances of the implementation class \texttt{IloNumVarI}. Their allocation and de-allocation on the Concert Technology heap are managed by an instance of \texttt{IloEnv}.

However, system classes, such as \texttt{IloCondition}, differ from that Concert Technology pattern. \texttt{IloCondition} is an ordinary C++ class. Its instances are allocated on the C++ heap.

Instances of \texttt{IloCondition} are not automatically de-allocated by a call to \texttt{IloEnv::end}. You must explicitly destroy instances of \texttt{IloCondition} by means of a call to the delete operator (which calls the appropriate destructor) when your application no longer needs instances of this class.

Furthermore, you should not allocate—neither directly nor indirectly—any instance of \texttt{IloCondition} on the Concert Technology heap because the destructor for that instance of \texttt{IloCondition} will never be called automatically by \texttt{IloEnv::end} when it cleans up other Concert Technology objects on the Concert Technology heap.
For example, it is not a good idea to make an instance of `IloCondition` part of a conventional Concert Technology model allocated on the Concert Technology heap because that instance will not automatically be de-allocated from the Concert Technology heap along with the other Concert Technology objects.

**De-allocating Instances of IloCondition**

Instances of `IloCondition` differ from the usual Concert Technology objects because they are not allocated on the Concert Technology heap, and their de-allocation is not managed automatically for you by `IloEnv::end`. Instead, you must explicitly destroy instances of `IloCondition` by calling the delete operator when your application no longer needs those objects.

**See Also**

`IloFastMutex`, `ILOUSEMT`

**Constructors**

```cpp
public IloCondition()
```

This constructor creates an instance of `IloCondition` and allocates it on the C++ heap (not in a Concert Technology environment). The instance contains data structures specific to an operating system.

**Methods**

```cpp
public void broadcast()
```

This member function wakes all threads currently waiting on the invoking condition. If there are no threads waiting, this member function does nothing.

```cpp
public void notify()
```

This member function wakes one of the threads currently waiting on the invoking condition.

```cpp
public void wait(IloFastMutex * m)
```

This member function first puts the calling thread to sleep while it unlocks the mutex `m`. Then, when either of the member functions `broadcast` or `notify` wakes up that thread, this member function acquires the lock on `m` and returns.
**IloConstraint**

**Category**  
Class

**InheritancePath**

![Inheritance Diagram]

**Definition File**  
ilconcert/iloexpression.h
Description

An instance of this class is a constraint in a model. To create a constraint, you can:

◆ use a constructor from a subclass of IloConstraint, such as IloRange, IloAllDiff, etc. For example:

IloAllDiff allDiff(env, vars);

◆ use a logical operator between constraints to return a constraint. For example, you can use the logical operators on other constraints, like this:
IloOr or = myConstraint1 ||
myConstraint2;

◆ use an arithmetic operator between a numeric variable and an expression to return a
constraint. For example, you can use the arithmetic operators on numeric variables
or expressions, like this:

IloRange rng = ( x + 3*y <= 7 );

After you create a constraint, you must explicitly add it to the model in order for it to be
taken into account. To do so, use the member function IloModel::add or the
template IloAdd. Then extract the model for an algorithm with the member function
IloAlgorithm::extract.

Most member functions in this class contain assert statements. For an explanation of
the macro NDEBUG (a way to turn on or turn off these assert statements), see the
concept Assert and NDEBUG.

See Also
IloConstraintArray, IloModel, IloRange

Constructors

public IloConstraint()
This constructor creates an empty handle. You must initialize it before you use it.

public IloConstraint(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

Methods

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking
handle.
**IloConstraintArray**

**Category** Class

**InheritancePath**

```
IloExtractableArray
    IloConstraintArray
        IloRangeArray
    Ilo SOS1Array
    Ilo SOS2Array
```

**Definition File** ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloConstraintArray(IloDefaultArrayI *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloConstraintArray(const IloConstraintArray &amp;)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloConstraintArray(const IloEnv, IloInt)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloConstraintArray(const IloEnv, IloInt, const IloConstraint, const IloConstraint v1...)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public void</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(IloInt, const IloConstraint)</td>
<td>add</td>
<td></td>
</tr>
<tr>
<td>add(const IloConstraint)</td>
<td>add</td>
<td></td>
</tr>
<tr>
<td>add(const IloConstraintArray)</td>
<td>add</td>
<td></td>
</tr>
<tr>
<td>operator<a href="IloInt"></a></td>
<td>operator[]</td>
<td></td>
</tr>
<tr>
<td>operator<a href="IloInt"></a> &amp;</td>
<td>operator[] &amp;</td>
<td></td>
</tr>
</tbody>
</table>
Description

For each basic type, Concert Technology defines a corresponding array class. 
IloConstraintArray is the array class of constraints for a model.

Instances of IloConstraintArray are extensible. That is, you can add more 
elements to such an array. References to an array change whenever an element is added 
or removed from the array.

Most member functions in this class contain assert statements. For an explanation of 
the macro NDEBUG (a way to turn on or turn off these assert statements), see the 
concept Assert and NDEBUG.

Arrays

See Also

IloConstraint, operator>>, operator

Constructors

public IloConstraintArray(IloDefaultArrayI * i)

This constructor creates an empty array. You cannot create instances of the 
undocumented class IloDefaultArrayI. As a parameter in this default constructor, 
it allows you to pass 0 (zero) as a value to an optional parameter in functions and 
member functions that accept an array as a parameter.

public IloConstraintArray(const IloConstraintArray & copy)

This copy constructor makes a copy of the array indicated by copy.

public IloConstraintArray(const IloEnv env, 
IloInt n)

This constructor creates an array of n elements, each of which is an empty handle.

public IloConstraintArray(const IloEnv env, 
IloInt n, 
const IloConstraint v0, 
const IloConstraint v1...)

This constructor creates an array of n constraints for use in a model.

Methods

public void add(IloInt more, 
const IloConstraint x)

This member function appends constraint to the invoking array multiple times. The parameter 
more indicates how many times.
public void add(const IloConstraint x)
This member function appends constraint to the invoking array.

public void add(const IloConstraintArray x)
This member function appends the elements in array to the invoking array.

public IloConstraint operator[](IloInt i)
This operator returns a reference to the constraint located in the invoking array at the position indicated by the index i. Concert Technology uses the const operator

IloConstraint & operator[](IloInt i) const;

on const arrays.

public IloConstraint & operator[](IloInt i)
This operator returns a reference to the constraint located in the invoking array at the position indicated by the index i.
IloConversion

Category  Class

InheritancePath

Definition File  ilconcert/iloexpression.h

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloConversion()</td>
</tr>
<tr>
<td>public</td>
<td>IloConversion(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloConversion(const IloEnv, const IloNumVar, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloConversion(const IloEnv, const IloNumVarArray, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloConversion(const IloEnv, const IloIntVarArray, IloNumVar::Type, const char *)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>ImplClass * getImpl()</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractable
Description

An instance of this class offers you a means to change the type of a numeric variable. For example, in a model (an instance of IloModel) extracted for an algorithm (such as an instance of the class IloCplex), you may want to convert the type of a given numeric variable (an instance of IloNumVar) from ILOFLOAT to ILOINT or to ILOBOOL (or from IloNumVar::Float to IloNumVar::Int or to IloNumVar::Bool). Such a change is known as a conversion.

After you create a conversion, you must explicitly add it to the model in order for it to be taken into account. To do so, use the member function IloModel::add or the template IloAdd. Then extract the model for an algorithm (such as an instance of IloCplex) with the member function IloAlgorithm::extract.

Multiple Type Conversions of the Same Variable

You can convert the type of a numeric variable in a model. To do so, create an instance of IloConversion and add it to the model. You can also convert the type of a numeric variable after the model has been extracted for an algorithm (such as an instance of IloCplex, documented in the ILOG CPLEX Reference Manual).

An instance of IloCplex will not accept more than one type conversion of the same variable. That is, you can change the type once, but not twice, in a single instance of IloCplex. Attempts to convert the type of the same variable more than once will throw the exception IloCplex::MultipleConversionException, documented in the ILOG CPLEX Reference Manual.

In situations where you want to change the type of a numeric variable more than once (for example, from Boolean to integer to floating-point), there are these possibilities:

◆ You can remove a prior conversion of a given variable in a given model. To do so, use its member function IloExtractable::end to delete it and optionally add a new conversion.

◆ You can apply different conversions to a given variable in more than one model, like this:

```cpp
IloNumVar x(env, 0, 10, ILOBOOL);
IloRange rng = (x <= 10);
IloModel mdl1(env);
mdl1.add(rng);
mdl1.add(IloConversion(env, x, ILOINT));
IloCplex cplex1(mdl1);
IloModel mdl2(env);
mdl2.add(rng);
```
```
mdl2.add(IloConversion(env, x, ILOFLOAT));
IloCplex cplex2(md12);
```

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

**See Also**

**Constructors**

- **public `IloConversion()`**
  
  This constructor creates an empty handle. You must initialize it before you use it.

- **public `IloConversion(ImplClass * impl)`**
  
  This constructor creates a handle object from a pointer to an implementation object.

- **public `IloConversion(const IloEnv env, const IloNumVar var, IloNumVar::Type t, const char * name)`**
  
  This constructor accepts a numeric variable and a type; it creates a handle to a type conversion to change the type of the variable `var` to the type indicated by `t`. You may use the parameter `name` to name the type conversion so that you can refer to it by a string identifier.

- **public `IloConversion(const IloEnv env, const IloNumVarArray vars, IloNumVar::Type t, const char * name)`**
  
  This constructor accepts an array of numeric variables and a type; it creates a handle to a type conversion to change the type of each variable in the array `vars` to the type indicated by `t`. You may use the parameter `name` to name the type conversion so that you can refer to it by a string identifier.

- **public `IloConversion(const IloEnv env, const IloIntVarArray vars, IloNumVar::Type t, const char * name)`**
  
  This constructor accepts an array of integer variables and a type; it creates a handle to a type conversion to change the type of each variable in the array `vars` to the type indicated by `t`. You may use the parameter `name` to name the type conversion so that you can refer to it by a string identifier.

**Methods**

- **public ImplClass * getImpl()**
This member function returns a pointer to the implementation object of the invoking handle.
IloDeleterMode

Category  
Enumeration

Definition File  
ilconcert/iloenv.h

Synopsis  
IloDeleterMode
IloLinearDeleterMode,
IloSafeDeleterMode,
IloRecursiveDeleterMode,
IloSmartDeleterMode

};

Summary  
An enumeration to set the mode of an IloDeleter.

Description  
This enumeration allows you to set the IloDeleter mode. The modes
IloRecursiveDeleterMode and IloSafeDeleterMode are not documented
and should not be used.

You can set the mode using the member function IloEnv::setDeleter. For a
description of deletion in ILOG Concert Technology, refer to Deletion of Extractables.

Fields  
IloLinearDeleterMode
IloSafeDeleterMode
IloRecursiveDeleterMode
IloSmartDeleterMode
IloDiff

Category: Class

Inheritance Path:

- IloExtractable
- IloNumExprArg
- IloIntExprArg
- IloConstraint
- IloDiff

Definition File: ilconcert/ilomodel.h

Constructor Summary:

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloDiff()</td>
<td></td>
</tr>
<tr>
<td>public IloDiff(ImplClass *)</td>
<td></td>
</tr>
<tr>
<td>public IloDiff(const IloEnv,</td>
<td></td>
</tr>
<tr>
<td>const IloNumExprArg,</td>
<td></td>
</tr>
<tr>
<td>const IloNumExprArg,</td>
<td></td>
</tr>
<tr>
<td>const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloDiff(const IloEnv,</td>
<td></td>
</tr>
<tr>
<td>const IloNumExprArg,</td>
<td></td>
</tr>
<tr>
<td>const IloNum,</td>
<td></td>
</tr>
<tr>
<td>const char *)</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass * getImpl()</td>
<td></td>
</tr>
</tbody>
</table>

Inherited methods from IloConstraint:

IloConstraint::getImpl
Description

An instance of this class is a constraint that enforces inequality (that is, “not equal” as indicated by \(!=\) in Concert Technology.

To create a constraint, you can:

- use the inequality operator \(!=\) on constrained variables (instances of \(\text{IloNumVar}\) and its subclasses) or expressions (instances of \(\text{IloExpr}\) and its subclasses).
- use a constructor from this class.

In order for the constraint to take effect, you must add it to a model with the template \(\text{IloAdd}\) or the member function \(\text{IloModel::add}\) and extract the model for an algorithm with the member function \(\text{IloAlgorithm::extract}\).

Most member functions in this class contain assert statements. For an explanation of the macro \(\text{NDEBUG}\) (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also

\(\text{IloAllDiff, IloConstraint, IloExpr, IloNumVar}\)

Constructors

public \textbf{IloDiff}()

This constructor creates an empty handle. You must initialize it before you use it.

public \textbf{IloDiff}(\texttt{ImplClass * impl})
This constructor creates a handle object from a pointer to an implementation object.

```java
class IloDiff
{
public:
    IloDiff(const IloEnv env,
             const IloNumExprArg expr1,
             const IloNumExprArg expr2,
             const char * name);
}
```

This constructor creates a constraint that enforces inequality (!=) in a model between the two expressions that are passed as its arguments. You must use the template IloAdd or the member function IloModel::add to add this constraint to a model in order for it to be taken into account.

The optional argument name is set to 0 by default.

```java
public IloDiff(const IloEnv env,
               const IloNumExprArg expr1,
               IloNum val,
               const char * name);
```

This constructor creates a constraint that enforces inequality (!=) in a model between the expression expr1 and the floating-point value that are passed as its arguments. You must use the template IloAdd or the member function IloModel::add to add this constraint to a model in order for it to be taken into account.

The optional argument name is set to 0 by default.

### Methods

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.
IloDisableNANDetection

Category: Global Function

Definition File: ilconcert/ilosys.h

 Synopsis: public void IloDisableNANDetection()

Description: This function disables NaN (Not a number) detection.
IloDistribute

**Category** Class

**InheritancePath**

```
IloExtractable
  ↓
IloNumExprArg
    ↓
IloIntExprArg
      ↓
IloConstraint
        ↓
IloDistribute
```

**Definition File** `ilconcert/ilomodel.h`

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloDistribute()</code></td>
</tr>
<tr>
<td><code>IloDistribute(ImplClass *)</code></td>
</tr>
<tr>
<td><code>IloDistribute(const IloEnv, const IloIntVarArray, const IloIntArray, const IloIntVarArray, const char *)</code></td>
</tr>
<tr>
<td><code>IloDistribute(const IloEnv, const IloIntVarArray, const IloIntVarArray, const char *)</code></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ImplClass * getImpl()</code></td>
</tr>
</tbody>
</table>

---

**Inherited methods from IloConstraint**
Description

An instance of this class is a counting constraint in a model. You can use an instance of this class to count the number of occurrences of several values among the constrained variables in an array of constrained variables. You can also use an instance of this class to force the constrained variables of an array to assume values in such a way that only a limited number of the constrained variables assume each value.

For example, if we have five cars to paint in three available colors, then we might refer to the cars as c1, c2, c3, c4, c5, and the colors as p1, p2, p3. If we can allow no more than three cars to be painted p1, exactly three cars to be painted p2, and no more than one car to be painted p3, then we can represent our problem informally in terms of this constraint like this:

\[\text{cards} = \{[0,3], [3,3], [0,1]\}\]
\[\text{values} = \{p1, p2, p3\}\]
\[\text{vars} = \{c1, c2, c3, c4, c5\}\]

In more formal terms, the constrained variables in the array \text{cards} are equal to the number of occurrences in the array \text{vars} of the values in the array \text{values}. More precisely, for each i, \text{cards}[i] is equal to the number of occurrences of \text{values}[i]...
in the array vars. After propagation of this constraint, the minimum of cards[i] is at least equal to the number of variables contained in vars bound to the value at values[i]; and the maximum of cards[i] is at most equal to the number of variables contained in vars that contain the value at values[i] in their domain.

The arrays cards and values must be the same length; otherwise, Concert Technology throws an exception on platforms that support C++ exceptions when exceptions are enabled.

When an instance of this class is created by a constructor with only cards and vars as arguments (that is, there is no values parameter), then the array of values that are being counted must be an array of consecutive integers starting with 0 (zero). In that case, for each i, cards[i] is equal to the number of occurrences of i in the array vars. After propagation of this constraint, the minimum of cards[i] is at least equal to the number of variables contained in vars bound to the value i; and the maximum of cards[i] is at most equal to the number of variables contained in vars that contain i in their domain.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

**See Also**

- IloConstraint, IloSequence

**Constructors**

public IloDistribute()

This constructor creates an empty handle. You must initialize it before you use it.

public IloDistribute(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloDistribute(const IloEnv env,
const IloIntVarArray cards,
const IloIntArray values,
const IloIntVarArray vars,
const char * name)

This constructor creates a counting constraint in a model. You must use the template IloAdd or the member function IloModel::add to add this constraint to a model and then use IloAlgorithm::extract to extract the model for an algorithm in order for the constraint to be taken into account.

The arrays cards and values must be the same length; otherwise Concert Technology throws the exception InvalidArraysException.
public IloDistribute(const IloEnv env,
     const IloIntVarArray cards,
     const IloIntVarArray vars,
     const char * name)

This constructor creates a counting constraint in a model. You must use the template
IloAdd or the member function IloModel::add to add this constraint to a model
and then use IloAlgorithm::extract to extract the model for an algorithm in
order for the constraint to be taken into account.

Methods

public ImplClass * getImpl()

This member function returns a pointer to the implementation object of the invoking
handle.
**IloDiv**

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**

public IloIntExprArg IloDiv(const IloIntExprArg x, const IloIntExprArg y)

public IloIntExprArg IloDiv(const IloIntExprArg x, IloInt y)

public IloIntExprArg IloDiv(IloInt x, const IloIntExprArg y)

**Description**  
This function is available for integer division. For numerical division, use `operator/`. 
IloEmptyHandleException

Category  
Class

InheritancePath

IloException

IloEmptyHandleException

Definition File  
ilconcert/iloenv.h

Summary  
The class of exceptions thrown if an empty handle is passed.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloEmptyHandleException()</td>
<td></td>
</tr>
<tr>
<td>public IloEmptyHandleException(const char *)</td>
<td></td>
</tr>
</tbody>
</table>

Inherited methods from IloException

IloException::end, IloException::getMessage

Description  
The exception IloEmptyHandleException is thrown if an empty handle is passed as parameter to a method, function, or class constructor.

Constructors

public IloEmptyHandleException()

public IloEmptyHandleException(const char * message)

This constructor creates an exception containing the message string message.
**IloEnableNANDetection**

**Category**          Global Function  
**Definition File**   ilconcert/ilosys.h  
**Synopsis**          public void IloEnableNANDetection()  
**Description**       This function enables NaN (Not a number) detection.
IloEndMT

Category: Global Function

Definition File: ilconcert/iloenv.h

Synopsis: public void IloEndMT()

Description: This function ends multithreading in a Concert Technology application.
IloEnv

Category  Class
InheritancePath

Definition File  ilconcert/iloenv.h
Summary  The class of environments for models or algorithms in Concert Technology.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloEnv()</td>
<td></td>
</tr>
<tr>
<td>public IloEnv(IloEnvI *)</td>
<td></td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool</td>
<td>assumeStrictNumericalDivision()</td>
</tr>
<tr>
<td>public void</td>
<td>assumeStrictNumericalDivision(IloBool)</td>
</tr>
<tr>
<td>public void</td>
<td>IloEnv::end()</td>
</tr>
<tr>
<td>public IloExtractableI *</td>
<td>getExtractable(IloInt)</td>
</tr>
<tr>
<td>public IloEnvI *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloRandom</td>
<td>getRandom()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getTime()</td>
</tr>
<tr>
<td>public const char *</td>
<td>getVersion()</td>
</tr>
<tr>
<td>public void</td>
<td>printTime()</td>
</tr>
<tr>
<td>public void</td>
<td>IloEnv::setDeleter(IloDeleterMode)</td>
</tr>
<tr>
<td>public void</td>
<td>setNormalizer(IloBool)</td>
</tr>
<tr>
<td>public void</td>
<td>unsetDeleter()</td>
</tr>
</tbody>
</table>

Description  An instance of this class is an environment, managing memory and identifiers for modeling objects. Every Concert Technology object, such as an extractable object, a model, or an algorithm, must belong to an environment. In C++ terms, when you construct a model (an instance of IloModel) or an algorithm (an instance of IloCplex, documented in the ILOG CPLEX Reference Manual, or IloSolver,
documented in the ILOG Solver Reference Manual, for example), then you must pass one instance of IloEnv as a parameter of that constructor.

Environment and Memory Management

An environment (an instance of IloEnv) efficiently manages memory allocations for the objects constructed with that environment as a parameter. For example, when Concert Technology objects in your model are extracted by an algorithm, those extracted objects are handled as efficiently as possible with respect to memory management; there is no unnecessary copying that might cause memory explosions in your application on the part of Concert Technology.

When your application deletes an instance of IloEnv, Concert Technology will automatically delete all models and algorithms depending on that environment as well. You delete an environment by calling the member function env.end.

The memory allocated for Concert Technology arrays, expressions, sets, and columns is not freed until all references to these objects have terminated and the objects themselves have been deleted.

Certain classes documented in this manual, such as IloFastMutex, are known as system classes. They do not belong to a Concert Technology environment; in other words, an instance of IloEnv is not a parameter in their constructors. As a consequence, a Concert Technology environment does not attempt to manage their memory allocation and de-allocation; a call of IloEnv:end will not delete an instance of a system class. These system classes are clearly designated in this documentation, and the appropriate constructors and destructors for them are documented in this manual as well.

Environment and Initialization

An instance of IloEnv in your application initializes certain data structures and modeling facilities for Concert Technology. For example, IloEnv initializes the symbolic constant IloInfinity.

The environment also indicates the current assumptions about normalization or the reduction of terms in linear expressions. For an explanation of this concept, see the concept Normalization: Reducing Linear Terms.

Environment and Communication Streams

An instance of IloEnv in your application initializes the default output streams for general information, for error messages, and for warnings.

Environment and Extractable Objects

Every extractable object in your problem must belong to an instance of IloEnv. In C++ terms, in the constructor of certain extractable objects that you create, such as a constrained variable, you must pass an instance of IloEnv as a parameter to indicate which environment the extractable object belongs to. An extractable object (that is, an
instance of IloExtractable or one of its derived subclasses) is tied throughout its lifetime to the environment where it is created. It can be used only with extractable objects belonging to the same environment. It can be extracted only for an algorithm attached to the same environment.

Two different environments cannot share the same extractable object.

You can extract objects from only one environment into a given algorithm. In other words, algorithms do not extract objects from two or more different environments.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also

IloException, IloModel, operator new

Constructors

public IloEnv()

This constructor creates an environment to manage the extractable objects in Concert Technology.

public IloEnv(IloEnvI * impl)

This constructor creates an environment (a handle) from its implementation object.

Methods

public IloBool assumeStrictNumericalDivision()

This member function checks whether the arithmetic division operator / is restricted to numerical division only. If it returns IloTrue, operator / is restricted, and integer division is accessible only through the IloDiv function.

If it returns IloFalse, operator / is not restricted.

The default value is IloTrue.

public void assumeStrictNumericalDivision(IloBool val)

This member function specifies whether the arithmetic division operator / is restricted to numerical division only. If val is IloTrue, operator / is restricted, and integer division is accessible only through the IloDiv function.

If val is IloFalse, operator / is not restricted.

The default value is IloTrue.

public void end()

When you call this member function, it cleans up the invoking environment. In other words, it deletes all the extractable objects (instances of IloExtractable and its subclasses) created in that environment and frees the memory allocated for them. It also deletes all algorithms (instances of IloAlgorithm and its subclasses) created in that environment.
environment and frees memory allocated for them as well, including the representations of extractable objects extracted for those algorithms.

The memory allocated for Concert Technology arrays, expressions, sets, and columns is not freed until all references to these objects have terminated and the objects themselves have been deleted.

```java
public IloExtractableI * getExtractable(IloInt id)
```

This member function returns the extractable associated with the specified identifier id.

```java
public IloEnvI * getImpl()
```

This member function returns the implementation object of the invoking environment.

```java
public IloRandom getRandom()
```

Each instance of IloEnv contains a random number generator, an instance of the class IloRandom. This member function returns that IloRandom instance.

```java
public IloNum getTime()
```

This member function returns the amount of time elapsed in seconds since the construction of the invoking environment. (The member function printTime directs this information to the output stream of the invoking environment.)

```java
public const char * getVersion()
```

This member function returns a string indicating the version of ILOG Concert Technology.

```java
public void printTime()
```

This member function directs the output of the member function getTime to the output stream of the invoking environment. (The member function getTime accesses the elapsed time in seconds since the creation of the invoking environment.)

```java
public void setDeleter(IloDeleterMode mode)
```

This member function sets the mode for the deletion of extractables, as described in the concept Deletion of Extractables. The mode can be IloLinearDeleterMode or IloSafeDeleterMode.

```java
public void setNormalizer(IloBool val)
```

This member function turns on or off the facilities in Concert Technology for normalizing linear expressions. Normalizing linear expressions is also known as reducing the terms of a linear expression. In this context, a linear expression that does not contain multiple terms with the same variable is said to be normalized. The concept in this manual offers examples of this idea.

When val is IloTrue, (the default), then Concert Technology analyzes linear expressions to determine whether any variable appears more than once in a given linear expression. It then combines terms in the linear expression to eliminate any duplication
of variables. This mode may require more time during preliminary computation, but it avoids the possibility of an assertion failing in the case of duplicated variables in the terms of a linear expression.

When \texttt{val} is \texttt{IloFalse}, then Concert Technology assumes that all linear expressions in the invoking environment have already been processed to reduce them to their most efficient form. In other words, Concert Technology assumes that linear expressions have been normalized. This mode may save time during computation, but it entails the risk that a linear expression may contain one or more variables, each of which appears in one or more terms. This situation will cause certain \texttt{assert} statements in Concert Technology to fail if you do not compile with the flag \texttt{-DNDEBUG}.

\begin{verbatim}
public void unsetDeleter()

This member function unsets the mode for the deletion of extractables, as described in the concept Deletion of Extractables.
\end{verbatim}
IloEnvironmentMismatch

Category            Class

InheritancePath

Definition File     ilconcert/iloenv.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloEnvironmentMismatch()</td>
</tr>
<tr>
<td>public IloEnvironmentMismatch(const char *)</td>
</tr>
</tbody>
</table>

Inherited methods from **IloException**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloException::end, IloException::getMessage</td>
</tr>
</tbody>
</table>

Description

The `IloEnvironmentMismatch` exception is thrown if you try to build an object using objects from another environment.

Constructors

- public `IloEnvironmentMismatch()`
- public `IloEnvironmentMismatch(const char * message)`
IloEqIntersection

Category       Global Function
Definition File ilconcert/iloanyset.h
Synopsis
public IloConstraint IloEqIntersection(const IloEnv env,
                                     const IloAnySetVar var1,
                                     const IloAnySetVar var2,
                                     const IloAnySetVar var3)
public IloConstraint IloEqIntersection(const IloEnv env,
                                     const IloIntSetVar var1,
                                     const IloIntSetVar var2,
                                     const IloIntSetVar var3)

Description
This function creates and returns a constraint (an instance of IloConstraint) for use in Concert Technology. The constraint forces the intersection of the sets var2 and var3 to be precisely the elements of the set intersection.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
IloEqPartition

Category              Global Function

Definition File       ilconcert/iloanyset.h

Synopsis              
public IloConstraint IloEqPartition(const IloEnv env,
const IloAnySetVar var,
const IloAnySetVarArray vars)
public IloConstraint IloEqPartition(const IloEnv ,
const IloIntSetVar var,
const IloIntSetVarArray vars)

Description           These functions create and return a constraint. When that constraint is posted, it insures that the value of the variable var will be required by exactly one set variable of the array vars. In this context, a constraint will be posted after it has been added to a model and extracted by a solver (for example, an instance of IloSolver documented in the ILOG Solver Reference Manual).
IloEqUnion

**Category**  Global Function

**Definition File**  ilconcert/iloanyset.h

**Synopsis**

```cpp
public IloConstraint IloEqUnion(const IloEnv  env,
const IloAnySetVar  var1,
const IloAnySetVar  var2,
const IloAnySetVarVar  var3)
public IloConstraint IloEqUnion(const IloEnv  env,
const IloAnySetVarVar  var,
const IloAnySetVarArray  vars)
public IloConstraint IloEqUnion(const IloEnv  env,
const IloIntSetVar  var1,
const IloIntSetVar  var2,
const IloIntSetVarVar  var3)
public IloConstraint IloEqUnion(const IloEnv ,
const IloIntSetVar  var,
const IloIntSetVarArray  vars)
```

**Description**

This function creates and returns a constraint (an instance of `IloConstraint`) for use in a model. When its parameters are two sets of variables, such as `var2` and `var3`, the constraint forces the union of the sets `var2` and `var3` to be precisely the elements of the set `unionSet`. Likewise, when its parameters include an array of set variables, such as `vars`, the constraint forces the union of the `elements` of that array to be `unionSet`.

In order for the constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`. 
IloException

Category       Class
InheritancePath

Definition File ilconcert/ilosys.h

Constructor Summary

| protected | IloException(const char *, IloBool) |

Method Summary

| public virtual void | IloException::end() |
| public virtual const char * | IloException::getMessage() |

Description

This class is the base class for exceptions in Concert Technology. An instance of this class represents an exception on platforms that support exceptions when exceptions are enabled.

See Also

IloEnv, operator

Constructors

protected IloException(const char * message, IloBool deleteMessage)

This protected constructor creates an exception.

Methods

public virtual void end()
This member function deletes the invoking exception. That is, it frees memory associated with the invoking exception.

```csharp
public virtual const char * getMessage()
```

This member function returns the message (a character string) of the invoking exception.
IloExponent

Category          Global Function
Definition File   ilconcert/iloexpression.h
Synopsis          public IloNumExprArg IloExponent(const IloNumExprArg arg)
                  public IloNum IloExponent(IloNum val)
Description       Concert Technology offers predefined functions that return an expression from an algebraic function on expressions. These predefined functions also return a numeric value from an algebraic function on numeric values as well.

IloExponent returns the exponentiation of its argument. In order to conform to IEEE 754 standards for floating-point arithmetic, you should use this function in your Concert Technology applications, rather than the standard C++ exp.
IloExpr

Category    Class

InheritancePath

Definition File    ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloExpr()</td>
</tr>
<tr>
<td>public</td>
<td>IloExpr(IloNumExprI *)</td>
</tr>
<tr>
<td>public</td>
<td>IloExpr(const IloNumLinExprTerm)</td>
</tr>
<tr>
<td>public</td>
<td>IloExpr(const IloIntLinExprTerm)</td>
</tr>
<tr>
<td>public</td>
<td>IloExpr(IloNumExprArg)</td>
</tr>
<tr>
<td>public</td>
<td>IloExpr(const IloEnv, IloNum)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum</td>
<td>getConstant()</td>
</tr>
<tr>
<td>public IloNumLinTermI *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloExpr::LinearIterator</td>
<td>getLinearIterator()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isNormalized()</td>
</tr>
<tr>
<td>public IloInt</td>
<td>normalize()</td>
</tr>
<tr>
<td>public IloExpr &amp;</td>
<td>operator *(IloNum)</td>
</tr>
<tr>
<td>public IloExpr &amp;</td>
<td>operator++(const IloIntLinExprTerm)</td>
</tr>
<tr>
<td>public IloExpr &amp;</td>
<td>operator++(const IloNumLinExprTerm)</td>
</tr>
<tr>
<td>public IloExpr &amp;</td>
<td>operator++(const IloIntVar)</td>
</tr>
<tr>
<td>public IloExpr &amp;</td>
<td>operator++(const IloNumVar)</td>
</tr>
</tbody>
</table>
public IloExpr & operator+=(const IloNumExprArg)
public IloExpr & operator+=(IloNum)
public IloExpr & operator-=(const IloIntLinExprTerm)
public IloExpr & operator-=(const IloNumLinExprTerm)
public IloExpr & operator-=(const IloIntVar)
public IloExpr & operator-=(const IloNumVar)
public IloExpr & operator-=(const IloNumExprArg)
public IloExpr & operator-=(const IloNum)
public void remove(const IloNumVarArray)
public void setConstant(IloNum)
public void setLinearCoef(const IloNumVar, IloNum)
public void setLinearCoefs(const IloNumVarArray, IloNumArray)
public void setNumConstant(IloNum)

**Inherited methods from IloNumExpr**
- IloNumExpr::getImpl
- IloNumExpr::operator *=
- IloNumExpr::operator +=
- IloNumExpr::operator -=
- IloNumExpr::operator /=

**Inherited methods from IloNumExprArg**
- IloNumExprArg::getImpl

**Inherited methods from IloExtractable**
- IloExtractable::end
- IloExtractable::getEnv
- IloExtractable::getId
- IloExtractable::getImpl
- IloExtractable::getName
- IloExtractable::getObject
- IloExtractable::setName
- IloExtractable::setObject
Description

An instance of this class represents an expression in a model. An instance of \texttt{IloExpr} is a handle.

Expressions in Environments

The variables in an expression must all belong to the same environment as the expression itself. In other words, you must not mix variables from different environments within the same expression.

Most member functions in this class contain \texttt{assert} statements. For an explanation of the macro \texttt{NDEBUG} (a way to turn on or turn off these \texttt{assert} statements), see the concept \texttt{Assert and NDEBUG}.

Programming Hint: Creating Expressions

In addition to using a constructor of this class to create an expression, you may also initialize an instance of \texttt{IloExpr} as a C++ expression built from variables of a model. For example:

\begin{verbatim}
IloNumVar x;
IloNumVar y;
IloExpr expr = x + y;
\end{verbatim}

Programming Hint: Empty Handles and Null Expressions

This statement creates an empty handle:

\begin{verbatim}
IloExpr e1;
\end{verbatim}

You must initialize it before you use it. For example, if you attempt to use it in this way:

\begin{verbatim}
e1 += 10; // BAD IDEA
\end{verbatim}

Without the compiler option \texttt{-DNDEBUG}, that line will cause an \texttt{assert} statement to fail because you are attempting to use an empty handle.

In contrast, the following statement

\begin{verbatim}
IloExpr e2(env);
\end{verbatim}

creates a handle to a null expression. You can use this handle to build up an expression, for example, in this way:

\begin{verbatim}
IloNumVar x;
IloNumVar y;
IloExpr expr = x + y;
\end{verbatim}
e2 += 10; // OK

Normalizing Linear Expressions: Reducing the Terms

Normalizing is sometimes known as reducing the terms of a linear expression.

Linear expressions consist of terms made up of constants and variables related by arithmetic operations; for example, $x + 3y$ is a linear expression of two terms consisting of two variables. In some expressions, a given variable may appear in more than one term, for example, $x + 3y + 2x$. Concert Technology has more than one way of dealing with linear expressions in this respect, and you control which way Concert Technology treats expressions from your application.

In one mode, Concert Technology analyzes linear expressions that your application passes it and attempts to reduce them so that a given variable appears in only one term in the linear expression. This is the default mode. You set this mode with the member function `setNormalizer(IloTrue)`.

In the other mode, Concert Technology assumes that no variable appears in more than one term in any of the linear expressions that your application passes to Concert Technology. We call this mode assume normalized linear expressions. You set this mode with the member function `setNormalizer(IloFalse)`.

Certain constructors and member functions in this class check this setting in the environment and behave accordingly: they assume that no variable appears in more than one term in a linear expression. This mode may save time during computation, but it entails the risk that a linear expression may contain one or more variables, each of which appears in one or more terms. Such a case may cause certain assertions in member functions of this class to fail if you do not compile with the flag `-DNDEBUG`.

Certain constructors and member functions in this class check this setting in the environment and behave accordingly: they attempt to reduce expressions. This mode may require more time during preliminary computation, but it avoids the possibility of a failed assertion in case of duplicates.

See Also

IloExprArray, IloModel

Constructors

public IloExpr()

This constructor creates an empty handle. You must initialize it before you use it.

public IloExpr(IloNumExprI* expr)

This constructor creates an expression from a pointer to the implementation class of numeric expressions IloNumExprI*.

public IloExpr(const IloNumLinExprTerm term)

This constructor creates an integer expression with linear terms using the undocumented class IloNumLinExprTerm.
public IloExpr(const IloIntLinExprTerm term)

This constructor creates an integer expression with linear terms using the undocumented class IloIntLinExprTerm.

public IloExpr(IloNumExprArg)

This constructor creates an expression using the undocumented class IloNumExprArg.

public IloExpr(const IloEnv env, IloNum)

This constructor creates an expression in the environment indicated by env. It may be used to build other expressions from variables belonging to env. You must not mix variables of different environments within an expression.

Methods

public IloNum getConstant()

This member function returns the constant term in the invoking expression.

public IloNumLinTermI * getImpl()

This member function returns the implementation object of the invoking enumerated variable.

public IloExpr::LinearIterator getLinearIterator()

This method returns a linear iterator on the invoking expression.

public IloBool isNormalized()

This member function returns IloTrue if the invoking expression has been normalized using normalize.

public IloInt normalize()

This member function normalizes the invoking linear expression. Normalizing is sometimes known as reducing the terms of a linear expression. That is, if there is more than one linear term using the same variable in the invoking linear expression, then this member function merges those linear terms into a single term expressed in that variable. The return value indicates the number of merged terms.

For example, 1*x + 17*y - 3*x becomes 17*y - 2*x, and the member function returns 1 (one).

If you attempt to use this member function on a nonlinear expression, it throws an exception.

public IloExpr & operator *(IloNum val)

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than x = x * ...
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \)

```java
public IloExpr & operator+=(const IloNumLinExprTerm term)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \)

```java
public IloExpr & operator+=(const IloIntVar var)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \)

```java
public IloExpr & operator+=(const IloNumVar var)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \)

```java
public IloExpr & operator+=(const IloNumExprArg expr)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \)

```java
public IloExpr & operator-=(const IloIntLinExprTerm term)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \)

```java
public IloExpr & operator-=(const IloIntVar var)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \)

```java
public IloExpr & operator-=(const IloNumVar var)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \)

```java
public IloExpr & operator-=(const IloNumExprArg expr)
```

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \)

```java
public IloExpr & operator-=(IloNum val)
```
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \)

public IloExpr & operator/=(IloNum val)

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x / \ldots \)

public void remove(const IloNumVarArray vars)

This member function removes all occurrences of all variables listed in the array \( vars \) from the invoking expression. For linear expressions, the effect of this member function is equivalent to setting the coefficient for all the variables listed in \( vars \) to 0 (zero).

public void setConstant(IloNum cst)

This member function assigns \( cst \) as the constant term in the invoking expression.

public void setLinearCoef(const IloNumVar var, IloNum value)

This member function assigns \( value \) as the coefficient of \( var \) in the invoking expression if the invoking expression is linear. This member function applies only to linear expressions. In other words, you cannot use this member function to change the coefficient of a non-linear expression. An attempt to do so will cause Concert Technology to throw an exception.

public void setLinearCoefs(const IloNumVarArray vars, IloNumArray values)

For each of the variables in \( vars \), this member function assigns the corresponding value of \( values \) as its linear coefficient if the invoking expression is linear. This member function applies only to linear expressions. In other words, you cannot use this member function to change the coefficient of a non-linear expression. An attempt to do so will cause Concert Technology to throw an exception.

public void setNumConstant(IloNum constant)

This member function assigns \( constant \) as the constant term in the invoking expression.
IloExprArray

**Category**  
Class

**Inheritance Path**

```
IloExpressionArray  
IloNumExprArray  
IloExprArray
```

**Definition File**  
ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public IloExprArray(IloDefaultArrayI *)</code></td>
<td></td>
</tr>
<tr>
<td><code>public IloExprArray(const IloEnv, IloInt)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public IloNumExprArg operator()(IloNumExprArg)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Inherited methods from IloNumExprArray

- IloNumExprArray::add, IloNumExprArray::add, IloNumExprArray::add, IloNumExprArray::operator()

### Inherited methods from IloExtractableArray

- IloExtractableArray::add, IloExtractableArray::add, IloExtractableArray::add, IloExtractableArray::endElements
IloExprArray

Description
For each basic type, Concert Technology defines a corresponding array class.
IloExprArray is the array class of the expressions class (IloExpr) for a model.
Instances of IloExprArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also
IloExpr

Constructors
public IloExprArray(IloDefaultArrayI * i)
This constructor creates an empty array of expressions for use in a model. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloExprArray(const IloEnv env,
                    IloInt n)
This constructor creates an array of n elements. Initially, the n elements are empty handles.

Methods
public IloNumExprArg operator()(IloNumExprArg anIntegerVariable)
This subscripting operator returns an expression argument for use in a constraint or expression. For clarity, let's call A the invoking array. When anIntegerVariable is bound to the value i, the domain of the expression is the domain of A[i]. More generally, the domain of the expression is the union of the domains of the expressions A[i] where the i are in the domain of anIntegerVariable.
IloExpr::LinearIterator

Category          Inner Class

InheritancePath

Definition File   ilconcert/iloexpression.h

Summary          An iterator over the linear part of an expression.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum getCoef()</td>
<td>Returns the coefficient of the current term.</td>
</tr>
<tr>
<td>public IloNumVar getVar()</td>
<td>Returns the variable of the current term.</td>
</tr>
<tr>
<td>public IloBool ok()</td>
<td>Returns IloTrue if there is a current element and the iterator points to it. Otherwise, returns IloFalse.</td>
</tr>
<tr>
<td>public void operator++()</td>
<td>Advances the iterator to point to the next term of the linear part of the expression.</td>
</tr>
</tbody>
</table>

Description          An instance of the nested class IloExpr::LinearIterator is an iterator that traverses the linear part of an expression.

Methods              

This member function returns the coefficient of the current term.

This member function returns the variable of the current term.

This member function returns IloTrue if there is a current element and the iterator points to it. Otherwise, it returns IloFalse.

This operator advances the iterator to point to the next term of the linear part of the expression.
IloExtractable

Category          Class

InheritancePath

Definition File  ilconcert/iloextractable.h

Constructor Summary

| public | IloExtractable(IloExtractableI *) |

Method Summary

| public void | IloExtractable::end() |
| public IloEnv | IloExtractable::getEnv() |
| public IloInt | IloExtractable::getId() |
| public IloExtractableI * | IloExtractable::getImpl() |
| public const char * | IloExtractable::getName() |
| public IloAny | IloExtractable::getObject() |
This class is the base class of all extractable objects (that is, instances of such classes as `IloConstraint`, `IloNumVar`, and so forth).

Instances of subclasses of this class represent objects (such as constraints, constrained variables, objectives, and so forth) that can be extracted by Concert Technology from your model for use by your application in Concert Technology algorithms.

Not every algorithm can extract every extractable object of a model. For example, a model may include more than one objective, but you can extract only one objective for an instance of `IloCplex`, documented in the ILOG CPLEX Reference Manual.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

Adding Extractable Objects

Generally, for an extractable object to be taken into account by one of the algorithms in Concert Technology, you must add the extractable object to a model with the member function `IloModel::add` and extract the model for the algorithm with the member function `IloAlgorithm::extract`.

Environment and Extractable Objects

Every extractable object in your model must belong to one instance of `IloEnv`. An extractable object (that is, an instance of `IloExtractable` or one of its derived subclasses) is tied throughout its lifetime to the environment where it is created. It can be used only with extractable objects belonging to the same environment. It can be extracted only for an algorithm attached to the same environment.

Notification

When you change an extractable object, for example by removing it from a model, Concert Technology notifies algorithms that have extracted the model containing this extractable object about the change. Member functions that carry out such notification are noted in this documentation.

See Also

`IloEnv`, `IloGetClone`, `IloModel`

Constructors

```cpp
public IloExtractable(IloExtractableI * obj)
```

This constructor creates a handle to the implementation object.

Methods

```cpp
public void end()
```
This member function first removes the invoking extractable object from all other extractable objects where it is used (such as a model, ranges, etc.) and then deletes the invoking extractable object. That is, it frees all the resources used by the invoking object. After a call to this member function, you cannot use the invoking extractable object again.

**Note:** The member function ends notifies Concert Technology algorithms about the destruction of this invoking object.

```java
public IloEnv getEnv()
```

This member function returns the environment to which the invoking extractable object belongs. An extractable object belongs to exactly one environment; different environments cannot share the same extractable object.

```java
public IloInt getId()
```

This member function returns the ID of the invoking extractable object.

```java
public IloExtractableI * getImpl()
```

This member function returns a pointer to the implementation object of the invoking extractable object. This member function is useful when you need to be sure that you are using the same copy of the invoking extractable object in more than one situation.

```java
public const char * getName()
```

This member function returns a character string indicating the name of the invoking object (if there is one).

```java
public IloAny getObject()
```

This member function returns the object associated with the invoking object (if there is one). Normally, an associated object contains user data pertinent to the invoking object.

```java
public void setName(const char * name)
```

This member function assigns name to the invoking object.

```java
public void setObject(IloAny object)
```

This member function associates object with the invoking object. The member function getObject accesses this associated object afterwards. Normally, object contains user data pertinent to the invoking object.
IloExtractableArray

**Category**  
Class

**Inheritance Path**

- IloExtractableArray
  - IloAnySetVarArray
  - IloAnyVarArray
  - IloConstraintArray
  - IloIntExprArray
  - IloIntSetVarArray
  - IloNumExprArray

**Definition File**  
ilconcert/iloextractable.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloExtractableArray(IloDefaultArrayI *)</code></td>
<td></td>
</tr>
<tr>
<td><code>IloExtractableArray(const IloExtractableArray &amp;)</code></td>
<td></td>
</tr>
<tr>
<td><code>IloExtractableArray(const IloEnv, IloInt)</code></td>
<td></td>
</tr>
<tr>
<td><code>IloExtractableArray(const IloEnv, IloInt, const IloExtractable, const IloExtractable v1...)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloExtractableArray::add(IloInt, const IloExtractable)</code></td>
<td></td>
</tr>
<tr>
<td><code>IloExtractableArray::add(const IloExtractable)</code></td>
<td></td>
</tr>
</tbody>
</table>
An instance of this class is an array of extractable objects (instances of the class `IloExtractable` or its subclasses).

Instances of `IloExtractableArray` are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

For information on arrays, see the concept `Arrays`.

### See Also
- `IloExtractable`, `operator`

### Constructors

**Public `IloExtractableArray(IloDefaultArrayI * i)`**

This constructor creates an empty array of elements. You cannot create instances of the undocumented class `IloDefaultArrayI`. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

**Public `IloExtractableArray(const IloExtractableArray & r)`**

This copy constructor creates a handle to the array of extractable objects indicated by `copy`.

**Public `IloExtractableArray(const IloEnv env, IloInt n)`**

This constructor creates an array of `n` elements, each of which is an empty handle.

**Public `IloExtractableArray(const IloEnv env, IloInt n, const IloExtractable v0, const IloExtractable v1...)`**

This constructor creates an array of `n` extractable objects for use in a model.

### Methods

**Public void `add(IloInt more, const IloExtractable x)`**

This member function appends `extr` to the invoking array multiple times. The parameter `more` indicates how many times.

**Public void `add(const IloExtractable x)`**

**Public void `IloExtractableArray::add(const IloExtractableArray)`**

**Public void `IloExtractableArray::endElements()`**
This member function appends \texttt{extr} to the invoking array.

\texttt{public void add(const IloExtractableArray x)}

This member function appends the elements in \texttt{array} to the invoking \texttt{array}.

\texttt{public void endElements()}\)

This member function calls \texttt{IloExtractable::end} for each of the elements in the invoking \texttt{array} and then calls \texttt{IloExtractableArray::clear} for the array itself. This member function is the recommended way to delete the elements of an array.
IloFastMutex

Category Class

InheritancePath

Definition File ilconcert/ilothread.h

Constructor Summary

| public | IloFastMutex() |

Method Summary

| public int | isOpened() |
| public void | IloFastMutex::lock() |
| public void | IloFastMutex::unlock() |

Description

The class IloFastMutex provides synchronization primitives adapted to the needs of Concert Technology. In particular, an instance of the class IloFastMutex is a non-recursive mutex that implements mutual exclusion from critical sections of code in multithreaded applications. The purpose of a mutex is to guarantee that concurrent calls to a critical section of code in a multithreaded application are serialized. If a critical section of code is protected by a mutex, then two (or more) threads cannot execute the critical section simultaneously. That is, an instance of this class makes it possible for you to serialize potentially concurrent calls.

Concert Technology implements a mutex by using a single resource that you lock when your application enters the critical section and that you unlock when you leave. Only one thread can own that resource at a given time.

See ILOUSEMT for details about the compilation macro to use with instances of this class.

Protection by a Mutex
A critical section of code in a multithreaded application is protected by a mutex when that section of code is encapsulated by a pair of calls to the member functions `IloFastMutex::lock` and `IloFastMutex::unlock`.

In fact, we say that a pair of calls to the member functions lock and unlock defines a critical section. The conventional way of defining a critical section looks like this:

```cpp
mutex.lock();
while (conditionC does not hold)
    condition.wait(&mutex);
doTreatmentT();
mutex.unlock();
```

The class `IloCondition` provides synchronization primitives to express conditions in critical sections of code.

**State of a Mutex**

A mutex (an instance of `IloFastMutex`) has a state; the state may be locked or unlocked. You can inquire about the state of a mutex to determine whether it is locked or unlocked by using the member function `isLocked`. When a thread enters a critical section of code in a multithreaded application and then locks the mutex defining that critical section, we say that the thread owns that lock and that lock belongs to the thread until the thread unlocks the mutex.

**Exceptions**

The member functions `IloFastMutex::lock` and `IloFastMutex::unlock` can throw C++ exceptions when exceptions are enabled on platforms that support them. These are the possible exceptions:

- **IloMutexDeadlock**: Instances of `IloFastMutex` are not recursive. Consequently, if a thread locks a mutex and then attempts to lock that mutex again, the member function `lock` throws the exception `MutexDeadlock`. On platforms that do not support exceptions, it causes the application to exit.

- **IloMutexNotOwner**: The thread that releases a given lock (that is, the thread that unlocks a mutex) must be the same thread that locked the mutex in the first place. For example, if a thread A takes lock L and thread B attempts to unlock L, then the member function `unlock` throws the exception `MutexNotOwner`. On platforms that do not support exceptions, it causes the application to exit.

- **IloMutexNotOwner**: The member function `unlock` throws this exception whenever a thread attempts to unlock an instance of `IloFastMutex` that is not already locked. On platforms that do not support exceptions, it causes the application to exit.

**System Class: Memory Management**
IloFastMutex is a system class.

Most Concert Technology classes are actually handle classes whose instances point to objects of a corresponding implementation class. For example, instances of the Concert Technology class IloNumVar are handles pointing to instances of the implementation class IloNumVarI. Their allocation and de-allocation in internal data structures of Concert Technology are managed by an instance of IloEnv.

However, system classes, such as IloFastMutex, differ from that pattern. IloFastMutex is an ordinary C++ class. Its instances are allocated on the C++ heap. Instances of IloFastMutex are not automatically de-allocated by a call to IloEnv::end. You must explicitly destroy instances of IloFastMutex by means of a call to the delete operator (which calls the appropriate destructor) when your application no longer needs instances of this class.

Furthermore, you should not allocate—neither directly nor indirectly—any instance of IloFastMutex in the Concert Technology environment because the destructor for that instance of IloFastMutex will never be called automatically by IloEnv::end when it cleans up other Concert Technology objects in the Concert Technology environment. In other words, allocation of any instance of IloFastMutex in the Concert Technology environment will produce memory leaks.

For example, it is not a good idea to make an instance of IloFastMutex part of a conventional Concert Technology model allocated in the Concert Technology environment because that instance will not automatically be de-allocated from the Concert Technology environment along with the other Concert Technology objects.

De-allocating Instances of IloFastMutex

Instances of IloFastMutex differ from the usual Concert Technology objects because they are not allocated in the Concert Technology environment, and their de-allocation is not managed automatically for you by IloEnv::end. Instead, you must explicitly destroy instances of IloFastMutex by calling the delete operator when your application no longer needs those objects.

See Also

IloBarrier, IloCondition, ILOUSEMT

Constructors

```cpp
public IloFastMutex()
```

This constructor creates an instance of IloFastMutex and allocates it on the C++ heap (not in the Concert Technology environment). This mutex contains operating system-specific resources to represent a lock. You may use this mutex for purposes that are private to a process. Its behavior is undefined for inter-process locking.

Methods

```cpp
public int isLocked()
```
This member function returns a Boolean value that indicates the state of the invoking mutex. That is, it tells you whether the mutex is locked by the calling thread (0) or unlocked (1) or locked by a thread other than the calling thread (also 1).

```java
public void lock()
```

This member function acquires a lock for the invoking mutex on behalf of the calling thread. That lock belongs to the calling thread until the member function `unlock` is called.

If you call this member function and the invoking mutex has already been locked, then the calling thread is suspended until the first lock is released.

```java
public void unlock()
```

This member function releases the lock on the invoking mutex, if there is such a lock.

If you call this member function on a mutex that has not been locked, then this member function throws an exception if C++ exceptions have been enabled on a platform that supports exceptions. Otherwise, it causes the application to exit.
IloFloatArray

Category  Macro

Synopsis  IloFloatArray()

Description  IloFloatArray is the array class of the basic floating-point class for a model. It is a handle class. The implementation class for IloFloatArray is the undocumented class IloFloatArrayI.

Instances of IloFloatArray are extensible. (They differ from instances of IlcFloatArray in this respect. IlcFloatArray is documented in the ILOG Solver Reference Manual.)

For each basic type, Concert Technology defines a corresponding array class. That array class is a handle class. In other words, an object of that class contains a pointer to another object allocated on the Concert Technology heap associated with a model. Exploiting handles in this way greatly simplifies the programming interface since the handle can then be an automatic object: as a developer using handles, you do not have to worry about memory allocation.

As handles, these objects should be passed by value, and they should be created as automatic objects, where “automatic” has the usual C++ meaning.

Member functions of a handle class correspond to member functions of the same name in the implementation class.

Assert and NDEBUG

Most member functions of the class IloFloatArray are inline functions that contain an assert statement. This statement checks that the handle pointer is non-null. These statements can be suppressed by the macro NDEBUG. This option usually reduces execution time. The price you pay for this choice is that attempts to access through null pointers are not trapped and usually result in memory faults.

See Also  IloNum
**IloFloatVar**

**Category**  
Macro

**Synopsis**  
`IloFloatVar()`

**Description**  
An instance of this class represents a constrained floating-point variable in Concert Technology.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept Assert and NDEBUG.

**See Also**  
`IloFloatVarArray`, `IloNumVar`
IloFloatVarArray

**Category**  
Macro

**Synopsis**  
*IloFloatVarArray*()

**Summary**  
The array class of *IloFloatVar*.

**Description**  
For each basic type, Concert Technology defines a corresponding array class.  
*IloFloatVarArray* is the array class of the floating-point variable class for a model.  
It is a handle class.

Instances of *IloFloatVarArray* are extensible.

Most member functions in this class contain *assert* statements. For an explanation of the macro *NDEBUG* (a way to turn on or turn off these *assert* statements), see the concept *Assert* and *NDEBUG*.

**See Also**  
*IloFloatVar*
IloFloor

Category: Global Function

Definition File: ilconcert/iloenv.h

Synopsis: public IloNum IloFloor(IloNum val)

Description: This function computes the largest integer value not greater than val.

Examples:

IloFloor(IloInfinity) is IloInfinity.
IloFloor(-IloInfinity) is -IloInfinity.
IloFloor(0) is 0.
IloFloor(0.4) is 0.
IloFloor(-0.4) is -1.
IloFloor(0.5) is 0.
IloFloor(-0.5) is -1.
IloFloor(0.6) is 0.
IloFloor(-0.6) is -1.
IloFunction

Category Class

InheritancePath

Description
Concert Technology offers you the means to define classes of functions that map instances of one class $X$ to instances of another class $Y$.

This C++ template creates a class of handles to the implementation class built by the template IloFunctionI.

Normally, you subclass the class IloFunctionI<$X,Y>$ and in doing so, you define its pure virtual member function:

```c++
virtual $Y$ getValue($X$);
```

Then you use this template to define a handle to that class of type IloFunction<$X,Y>$.

Definition file: <ilconcert/iloset.h>
**IloGetClone**

**Category**  
Global Function

**Definition File**  
ilconcert/iloextractable.h

**Synopsis**  
public X IloGetClone(const IloEnv env,  
const X x)

**Description**  
This C++ template creates a clone (that is, an exact copy) of an instance of the class X.
## IloHalfPi

### Category
Global Variable

### Definition File
`ilconcert/ilosys.h`

Concert Technology predefines conventional trigonometric constants to conform to IEEE 754 standards for quarter pi, half pi, pi, three-halves pi, and two pi.

```c
extern const IloNum IloHalfPi;  // = 1.57079632679489661923
```
IloIfThen

Category  Class

InheritancePath

Definition File  ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Access</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIfThen()</td>
</tr>
<tr>
<td>public</td>
<td>IloIfThen(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIfThen(const IloEnv, const IloConstraint, const IloConstraint, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Access</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
</tbody>
</table>

Inherited methods from IloConstraint

IloConstraint::getImpl
Description

An instance of \texttt{IloIfThen} represents a condition constraint. Generally, a condition constraint is composed of an if part (the conditional statement or left side) and a then part (the consequence or right side).

In order for a constraint to take effect, you must add it to a model with the template \texttt{IloAdd} or the member function \texttt{IloModel::add} and extract the model for an algorithm with the member function \texttt{IloAlgorithm::extract}.

Most member functions in this class contain \texttt{assert} statements. For an explanation of the macro \texttt{NDEBUG} (a way to turn on or turn off these \texttt{assert} statements), see the concept \texttt{Assert and NDEBUG}.

See Also

\texttt{IloConstraint}

Constructors

public \texttt{IloIfThen}()

This constructor creates an empty handle. You must initialize it before you use it.

public \texttt{IloIfThen}(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public \texttt{IloIfThen}(const \texttt{IloEnv} env,
const \texttt{IloConstraint} left,
const \texttt{IloConstraint} right,
const char * name)
This constructor creates a condition constraint in the environment indicated by \texttt{env}. The parameter \texttt{left} indicates the if part of the condition. The parameter \texttt{right} indicates the then part of the condition. The string \texttt{name} indicates the name of the constraint; it is set to 0 by default. For the constraint to take effect, you must add it to a model and extract the model for an algorithm.

### Methods

**public ImplClass * getImpl()**

This member function returns a pointer to the implementation object of the invoking handle.
IloInfinity

Category: Global Variable

Definition File: ilconcert/ilosys.h

This symbolic constant represents the largest double-precision floating-point number on a given platform. It is initialized when you create an instance of IloEnv. In practice, when you use this symbolic constant as an upper bound of a variable in your model, you are effectively stating that the variable is unbounded.

See the ILOG Solver Reference Manual and User's Manual for details about how ILOG Solver treats floating-point calculations in instances of IloSolver in conformity with IEEE 754. In particular, ILOG Solver offers other symbolic constants, such as IlcIntMax or IlcFloatMax that may be more appropriate for your application if you do not intend to state that your variables are effectively unbounded.


See Also:
**IloInitMT**

**Category**  
Global Function

**Definition File**  
ilconcert/iloenv.h

**Synopsis**  
public void IloInitMT()
public void IloInitMT(IloBaseEnvMutex *)

**Description**  
This function initializes multithreading in a Concert Technology application.
IloInt

Category: Type Definition

Definition File: ilconcert/ilosys.h

Synopsis: long IloInt

Description: This type definition represents signed integers in Concert Technology.

See Also: IloBool, IloModel, IloNum
IloIntArray

Category Class

InheritancePath

Description IloIntArray is the array class of the basic integer class for a model. It is a handle class. The implementation class for IloIntArray is the undocumented class IloIntArrayI.

Instances of IloIntArray are extensible. (They differ from instances of IlcIntArray in this respect.) References to an array change whenever an element is added to or removed from the array.

For each basic type, Concert Technology defines a corresponding array class. That array class is a handle class. In other words, an object of that class contains a pointer to another object allocated in a Concert Technology environment associated with a model. Exploiting handles in this way greatly simplifies the programming interface since the handle can then be an automatic object: as a developer using handles, you do not have to worry about memory allocation.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntArray(IloArrayI *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntArray(const IloEnv, IloInt, IloInt, IloInt v1...)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloNumArray</td>
</tr>
<tr>
<td></td>
<td>IloIntArray::toNumArray()</td>
</tr>
</tbody>
</table>
As handles, these objects should be passed by value, and they should be created as automatic objects, where “automatic” has the usual C++ meaning.

Member functions of a handle class correspond to member functions of the same name in the implementation class.

**Assert and NDEBUG**

Most member functions of the class `IloIntArray` are inline functions that contain an `assert` statement. This statement checks that the handle pointer is non-null. These statements can be suppressed by the macro `NDEBUG`. This option usually reduces execution time. The price you pay for this choice is that attempts to access through null pointers are not trapped and usually result in memory faults.

### See Also

`IloInt`

### Constructors

public `IloIntArray(IloArrayI * i)`

This constructor creates an array of integers from an implementation object.

public `IloIntArray(const IloEnv env, IloInt n)`

This constructor creates an array of `n` integers for use in a model in the environment indicated by `env`. By default, its elements are empty handles.

public `IloIntArray(const IloEnv env, IloInt n, IloInt v0, IloInt v1...)`

This constructor creates an array of `n` integers; the elements of the new array take the corresponding values: `v0`, `v1`, ..., `v(n-1)`.

### Methods

public `IloNumArray toNumArray()`

This constructor creates an array of integers from an array of numeric values.
**IloIntBinaryPredicate**

**Category**  
Class

**Inheritance Path**

![IloIntBinaryPredicate]

**Definition File**  
ilconcert/ilotupleset.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloIntBinaryPredicate()</td>
<td></td>
</tr>
<tr>
<td>public IloIntBinaryPredicate(IloIntBinaryPredicateI *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloIntBinaryPredicateI *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isTrue(const IloInt, const IloInt)</td>
</tr>
<tr>
<td>public void</td>
<td>operator=(const IloIntBinaryPredicateI &amp;)</td>
</tr>
</tbody>
</table>

**Description**

This class makes it possible for you to define binary predicates operating on arbitrary objects in a model. A predicate is an object with a member function (such as `isTrue`) that checks whether or not a property is satisfied by an ordered set of (pointers to) objects.

**Defining a New Class of Predicates**

Predicates, like other Concert Technology objects, depend on two classes: a handle class, `IloIntBinaryPredicate`, and an implementation class, such as `IloIntBinaryPredicateI`, where an object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of an instance of `IloIntBinaryPredicateI` allocated in a Concert Technology environment. As a Concert Technology user, you will be working primarily with handles.
If you define a new class of predicates yourself, you must define its implementation class together with the corresponding virtual member function `isTrue`, as well as a member function that returns an instance of the handle class `IloIntBinaryPredicate`.

**Arity**

As a developer, you can use predicates in Concert Technology applications to define your own constraints that have not already been predefined in Concert Technology. In that case, the *arity* of the predicate (that is, the number of constrained variables involved in the predicate, and thus the size of the array that the member function `isTrue` must check) must be two.

**See Also**  
`IloTableConstraint`

**Constructors**

```cpp
public IloIntBinaryPredicate()
```

This constructor creates an empty binary predicate. In other words, the predicate is an empty handle with a null handle pointer. You must assign the elements of the predicate before you attempt to access it, just as you would any other pointer. Int attempt to access it before this assignment will throw an exception (an instance of `IloSolver::SolverErrorException`).

```cpp
public IloIntBinaryPredicate(IloIntBinaryPredicateI * impl)
```

This constructor creates a handle object (an instance of the class `IloIntBinaryPredicate`) from a pointer to an implementation object (an instance of the implementation class `IlcIntPredicateI`, documented in the *ILOG Solver Reference Manual*).

**Methods**

```cpp
public IloIntBinaryPredicateI * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```cpp
public IloBool isTrue(const IloInt val1,  
                     const IloInt val2)
```

This member function returns `IloTrue` if the values `val1` and `val2` make the invoking binary predicate valid. It returns `IloFalse` otherwise.

```cpp
public void operator=(const IloIntBinaryPredicate & h)
```

This assignment operator copies `h` into the invoking predicate by assigning an address to the handle pointer of the invoking object. That address is the location of the implementation object of the argument `h`. After execution of this operator, both the invoking predicate and `h` point to the same implementation object.
IloIntExpr

Category          Class

InheritancePath

Definition File     ilconcert/iloexpression.h

Summary            The class of integer expressions in Concert Technology 2.0.

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntExpr()</td>
</tr>
<tr>
<td>public</td>
<td>IloIntExpr(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntExpr(const IloIntExprArg)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntExpr(const IloIntLinExprTerm)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntExpr(const IloEnv)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloIntExpr &amp;</td>
<td>operator *(IloInt)</td>
</tr>
<tr>
<td>public IloIntExpr &amp;</td>
<td>operator**(const IloIntExprArg)</td>
</tr>
<tr>
<td>public IloIntExpr &amp;</td>
<td>operator**(IloInt)</td>
</tr>
<tr>
<td>public IloIntExpr &amp;</td>
<td>operator--(const IloIntExprArg)</td>
</tr>
<tr>
<td>public IloIntExpr &amp;</td>
<td>operator--(IloInt)</td>
</tr>
</tbody>
</table>
Description
Integer expressions in Concert Technology 2.0 are represented using objects of type IloIntExpr.

Constructors
- **IloIntExpr()**
  This constructor creates an empty handle. You must initialize it before you use it.
- **IloIntExpr(ImplClass * impl)**
  This constructor creates a handle object from a pointer to an implementation object.
- **IloIntExpr(const IloIntExprArg arg)**
  This constructor creates an integer expression using the undocumented class IloIntExprArg.
- **IloIntExpr(const IloIntLinExprTerm term)**
  This constructor creates an integer expression with linear terms using the undocumented class IloIntLinExprTerm.
- **IloIntExpr(const IloEnv env)**
  This constructor creates an integer expression in the environment env.

Methods
- **ImplClass * getImpl()**
This member function returns a pointer to the implementation object of the invoking handle.

public IloIntExpr & operator *(IloInt val)
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x \times \ldots \).

public IloIntExpr & operator++(const IloIntExprArg expr)
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \).

public IloIntExpr & operator++(IloInt val)
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x + \ldots \).

public IloIntExpr & operator--(const IloIntExprArg expr)
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \).

public IloIntExpr & operator--(IloInt val)
This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than \( x = x - \ldots \).
**IloIntExprArg**

***Category***  
Class

***Inheritance Path***

![Inheritance Diagram]

***Definition File***  
ilconcert/iloexpression.h

***Summary***  
A class used internally in Concert Technology.

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntExprArg()</td>
</tr>
<tr>
<td>public</td>
<td>IloIntExprArg(ImplClass *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public ImplClass *</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntExprArg::getImpl()</td>
</tr>
</tbody>
</table>

**Inherited methods from**  
*IloNumExprArg*

IloNumExprArg::getImpl
Description
Concert Technology uses instances of these classes internally as temporary objects when it is parsing a C++ expression in order to build an instance of IloIntExpr. As a Concert Technology user, you will not need this class yourself; in fact, you should not use them directly. They are documented here because the return value of certain functions, such as IloSum or IloScalProd, can be an instance of this class.

Constructors
public IloIntExprArg()
This constructor creates an empty handle. You must initialize it before you use it.

public IloIntExprArg(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

Methods
public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.
IloIntExprArray

Category Class

InheritancePath

Definition File ilconcert/iloexpression.h

Summary The array class of IloIntExpr.

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloIntExprArray(IloDefaultArray1 *)</td>
<td>IloIntExprArray</td>
</tr>
<tr>
<td>public IloIntExprArray(const IloEnv, IloInt)</td>
<td>IloIntExprArray</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void IloIntExprArray::add(IloInt, const IloIntExpr)</td>
<td>Add integer expression to the array</td>
</tr>
<tr>
<td>public void IloIntExprArray::add(const IloIntExpr)</td>
<td>Add expression to the array</td>
</tr>
<tr>
<td>public void IloIntExprArray::add(const IloIntExprArray)</td>
<td>Add array to the array</td>
</tr>
<tr>
<td>public IloIntExprArg IloIntExprArray::operator<a href="IloIntExprArg"></a></td>
<td>Get expression at index</td>
</tr>
<tr>
<td>public IloIntExpr IloIntExprArray::operator<a href="IloInt"></a></td>
<td>Get expression at index</td>
</tr>
<tr>
<td>public IloIntExpr &amp; IloIntExprArray::operator<a href="IloInt"></a></td>
<td>Get reference to expression at index</td>
</tr>
</tbody>
</table>

### Inherited methods from IloExtractableArray

- add, add, add, endElements
IloIntExprArray

Description
For each basic type, Concert Technology defines a corresponding array class.
IloNumExprArray is the array class of the integer expressions class
(IloNumExpr) for a model.

Instances of IloIntExprArray are extensible. That is, you can add more elements
to such an array. References to an array change whenever an element is added to or
removed from the array.

Constructors
public IloIntExprArray(IloDefaultArrayI * i)
This constructor creates an empty array of elements. You cannot create instances of the
undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and
member functions that accept an array as a parameter.

public IloIntExprArray(const IloEnv env,
IloInt n)
This copy constructor creates a handle to the array of extractable objects indicated by copy.

Methods
public void add(IloInt more,
const IloIntExpr x)
This member function appends x to the invoking array multiple times. The parameter
more indicates how many times.

public void add(const IloIntExpr x)
This member function appends x to the invoking array.

public void add(const IloIntExprArray array)
This member function appends the elements in array to the invoking array.

public IloIntExprArg operator[](IloIntExprArg anIntegerVariable)
This subscripting operator returns an expression argument for use in a constraint or
expression. For clarity, let’s call A the invoking array. When anIntegerVariable is
bound to the value i, the domain of the expression is the domain of A[i]. More
generally, the domain of the expression is the union of the domains of the expressions
A[i] where the i are in the domain of anIntegerVariable.

public IloIntExpr operator[](IloInt i)
This operator returns a reference to the extractable object located in the invoking array at
the position indicated by the index i. Concert Technology uses the const operator

IloIntExpr operator[](IloInt i) const;

on const arrays.
public IloIntExpr & operator[](IloInt i)

This operator returns a reference to the extractable object located in the invoking array at the position indicated by the index i.
## IloIntSet

**Category**  
Class

**Inheritance Path**  
IloIntSet

**Definition File**  
ilconcert/iloset.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloIntSet (const IloEnv, const IloIntArray)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSet (const IloEnv, const IloNumArray)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSet (const IloEnv)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSet (IloIntSetI *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void add (IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public void add (IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloBool contains (IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public IloBool contains (IloInt)</td>
<td></td>
</tr>
<tr>
<td>public void empty ()</td>
<td></td>
</tr>
<tr>
<td>public void end ()</td>
<td></td>
</tr>
<tr>
<td>public IloEnv getEnv ()</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetI * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getSize ()</td>
<td></td>
</tr>
<tr>
<td>public IloBool intersects (IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public IloBool intersects (IloInt)</td>
<td></td>
</tr>
<tr>
<td>public void remove (IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public void remove (IloInt)</td>
<td></td>
</tr>
<tr>
<td>public void setIntersection (IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public void setIntersection (IloInt)</td>
<td></td>
</tr>
</tbody>
</table>
An instance of this class offers a convenient way to represent a set of integer values as a constrained variable in Concert Technology.

A constrained variable representing a set of integer values (that is, an instance of IloIntSetVar) is defined in terms of two other sets: its required elements and its possible elements. Its required elements are those that must be in the set. Its possible elements are those that may be in the set. This class offers member functions for accessing the required and possible elements of a set of integer values.

The function IloCard offers you a way to constrain the number of elements in a set variable. That is, IloCard constrains the cardinality of a set variable.

**See Also**
- IloCard, IloEqIntersection, IloEqUnion, IloExtractable, IloMember, IloModel, IloNotMember, IloNullIntersect, IloIntSetVarArray, IloSubset, IloSubsetEq

**Constructors**

public IloIntSet(const IloEnv &env, const IloIntArray &array)

This constructor creates a set of integer values in the environment `env` from the elements in `array`.

public IloIntSet(const IloEnv &env, const IloNumArray &array)

This constructor creates a set of numeric values in the environment `env` from the elements in `array`.

public IloIntSet(const IloEnv &env)

This constructor creates an empty set (no elements) in the environment `env`. You must use the member function `IloIntSet::add` to fill this set with elements.

public IloIntSet(IloIntSetI * impl)

This constructor creates a handle to a set of numeric values from its implementation object.

**Methods**

public void add(IloIntSet set)

This member function adds `set` to the invoking set. Here, "adds" means that the invoking set becomes the union of its former elements and the elements of `set`. 

**Inner Class**

Inner Class: IloIntSet::IloIntSet::Iterat or
To calculate the arithmetic sum of values in an array, use the function `IloSum`.

```
public void add(IloInt elt)
```

This member function adds `elt` to the invoking set. Here, “adds” means that the invoking set becomes the union of its former elements and the new `elt`.

```
public IloBool contains(IloIntSet set)
```

This member function returns a Boolean value (zero or one) that indicates whether `set` contains the invoking set. The value one indicates that the invoking set contains all the elements of `set`, and that the intersection of the invoking set with `set` is precisely `set`. The value zero indicates that the intersection of the invoking set and `set` is not precisely `set`.

```
public IloBool contains(IloInt elt)
```

This member function returns a Boolean value (zero or one) that indicates whether `elt` is an element of the invoking set. The value one indicates that the invoking set contains `elt`; the value zero indicates that the invoking set does not contain `elt`.

```
public void empty()
```

This member function removes the elements from the invoking set. In other words, the invoking set becomes the empty set.

```
public void end()
```

This member function deletes an `IloIntSet` object. This function should be called when the `IloIntSet` object is no longer needed.

```
public IloEnv getEnv()
```

This member function returns the environment to which the invoking set belongs.

```
public IloIntSetI * getImpl()
```

This member function returns a pointer to the implementation object of the invoking set.

```
public IloInt getSize()
```

This member function returns an integer indicating the size of the invoking set (that is, how many elements it contains).

```
public IloBool intersects(IloIntSet set)
```

This member function returns a Boolean value (zero or one) that indicates whether `set` intersects the invoking set. The value one indicates that the intersection of `set` and the invoking set is not empty (at least one element in common); the value zero indicates that the intersection of `set` and the invoking set is empty (no elements in common).

```
public void remove(IloIntSet set)
```

This member function removes all the elements of `set` from the invoking set.

```
public void remove(IloInt elt)
```
This member function removes elt from the invoking set.

public void setIntersection(IloIntSet set)

This member function changes the invoking set so that it includes only the elements of set. In other words, the invoking set becomes the intersection of its former elements with the elements of set.

public void setIntersection(IloInt elt)

This member function changes the invoking set so that it includes only the element indicated by elt. In other words, the invoking set becomes the intersection of its former elements with elt.
IloIntSet::Iterator

Category       Inner Class

InheritancePath

Definition File ilconcert/iloset.h

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IloBool</td>
<td>ok()</td>
</tr>
<tr>
<td>IloInt</td>
<td>operator *()</td>
</tr>
<tr>
<td>void</td>
<td>operator++()</td>
</tr>
</tbody>
</table>

### Description

An instance of the nested class IloIntSet::Iterator is an iterator that traverses the elements of a finite set of numeric values (an instance of IloIntSet).

### See Also

IloIntSet

### Methods

**public IloBool ok()**

This member function returns IloTrue if there is a current element and the invoking iterator points to it. Otherwise, it returns IloFalse.

To traverse the elements of a finite set of pointers, use the following code:

```c++
IloInt val;
for(IloIntSet::Iterator iter(set); iter.ok(); ++iter){
    val = *iter;
    // do something with val
}
```

**public IloInt operator *()**

This operator returns the current element, the one to which the invoking iterator points.

**public void operator++()**

This operator advances the iterator to point to the next value in the set.
IloIntSetVar

Category Class
InheritancePath

Definition File ilconcert/iloset.h

Summary The class IloIntSetVar.

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloIntSetVar()</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetVar(ImplClass *)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetVar(const IloEnv, const IloIntArray, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetVar(const IloEnv, const IloIntArray, const IloIntArray, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetVar(const IloEnv, const IloNumArray, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSetVar(const IloEnv, const IloNumArray, const IloNumArray, const char *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void addPossible(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public void addRequired(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public ImplClass * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public void getPossibleSet(IloIntSet)</td>
<td></td>
</tr>
<tr>
<td>public IloIntSet getPossibleSet()</td>
<td></td>
</tr>
</tbody>
</table>
Description

An instance of this class represents a set of integer values. The same integer value will not appear more than once in a set. The elements of a set are not ordered.

A constrained variable representing a set of integer values (that is, an instance of IloIntSetVar) is defined in terms of two other sets: its required elements and its possible elements. Its required elements are those that must be in the set. Its possible elements are those that may be in the set. This class offers member functions for accessing the required and possible elements of a set of integer values.

The function IloCard offers you a way to constrain the number of elements in a set variable. That is, IloCard constrains the cardinality of a set variable.

Constructors

public IloIntSetVar()

This constructor creates an empty handle. You must initialize it before you use it.

public IloIntSetVar(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloIntSetVar(const IloEnv env,
                     const IloIntArray array,
                     const char * name)

This constructor creates a constrained set variable and makes it part of the environment env, where the set consists of integer values. By default, its name is indicated by the empty string, but you can indicate a name of your choice.

public IloIntSetVar(const IloEnv env,
                     const IloIntArray possible,
                     const IloIntArray required,
                     const char * name)
This constructor creates a constrained set variable and makes it part of the environment `env`, where the set consists of integer values. The array `possible` indicates the set of possible elements of the set variable; the array `required` specifies the set of required elements of the set variable. By default, its name is indicated by the empty string, but you can indicate a name of your choice.

```cpp
public IloIntSetVar(const IloEnv  env,
                     const IloNumArray  array,
                     const char * name)
```

This constructor creates a constrained set variable and makes it part of the environment `env`, where the set consists of integer values. By default, its name is indicated by the empty string, but you can indicate a name of your choice.

```cpp
public IloIntSetVar(const IloEnv  env,
                     const IloNumArray  possible,
                     const IloNumArray  required,
                     const char * name)
```

This constructor creates a constrained set variable and makes it part of the environment `env`, where the set consists of integer values. The numeric array `possible` indicates the set of possible elements of the set variable; the numeric array `required` specifies the set of required elements of the set variable. By default, its name is indicated by the empty string, but you can indicate a name of your choice.

**Methods**

```cpp
public void addPossible(IloInt elt)
```

This member function adds `elt` to the set of possible elements of the invoking set variable.

**Note:** The member function `addPossible` notifies Concert Technology algorithms about this change of this invoking object.

```cpp
public void addRequired(IloInt elt)
```

This member function adds `elt` to the set of required elements of the invoking set variable.

**Note:** The member function `addRequired` notifies Concert Technology algorithms about this change of this invoking object.
This member function returns a pointer to the implementation object of the invoking handle.

public void get PossibleSet(IloIntSet set)

This member function accesses the possible elements of the invoking set variable and puts those elements into its parameter set.

public IloIntSet getPossibleSet()

This member function returns the possible elements of the invoking set variable.

public void getRequiredSet(IloIntSet set)

This member function accesses the possible elements of the invoking set variable and puts those elements into its parameter set.

public IloIntSet getRequiredSet()

This member function returns the required elements of the invoking set variable.

public void removePossible(IloInt elt)

This member function removes elt as a possible element of the invoking set variable.

Note: The member function removePossible notifies Concert Technology algorithms about this change of this invoking object.

public void removeRequired(IloInt elt)

This member function removes elt as a required element of the invoking set variable.

Note: The member function removeRequired notifies Concert Technology algorithms about this change of this invoking object.
IloIntSetVarArray

Category
Class

InheritancePath

Definition File
ilconcert/iloset.h

Summary
The array class of the set variable class for integer values.

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntSetVarArray(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntSetVarArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntSetVarArray(const IloEnv, IloInt, const IloIntSetVar, const IloIntSetVar v1...)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public, void</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(IloInt, const IloIntSetVar)</td>
<td></td>
</tr>
<tr>
<td>add(const IloIntSetVar)</td>
<td></td>
</tr>
<tr>
<td>add(const IloIntSetVarArray)</td>
<td></td>
</tr>
<tr>
<td>operator<a href="IloInt"></a></td>
<td></td>
</tr>
<tr>
<td>operator<a href="IloInt"></a></td>
<td></td>
</tr>
</tbody>
</table>

### Inherited methods from IloExtractableArray

IloExtractableArray::add, IloExtractableArray::add,
IloExtractableArray::add, IloExtractableArray::endElements
Description

For each basic type, Concert Technology defines a corresponding array class. `IloIntSetVarArray` is the array class of the set variable class for integer values (`IloIntSetVar`) in a model.

Instances of `IloIntSetVarArray` are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added or removed from the array.

 Constructors

```java
public IloIntSetVarArray(IloDefaultArrayI * i)

This constructor creates an empty extensible array of set variables. You cannot create instances of the undocumented class `IloDefaultArrayI`. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloIntSetVarArray(const IloEnv env, IloInt n)

This constructor creates an extensible array of n set variables, where each set is a set of integer values.

public IloIntSetVarArray(const IloEnv env, const IloInt n, const IloIntSetVar v0, const IloIntSetVar v1...)

This constructor creates an extensible array of n set variables, as indicated by `v0`, `v1`, etc.

 Methods

```java
public void add(IloInt more, const IloIntSetVar x)

This member function appends `x` to the invoking array multiple times. The parameter `more` indicates how many times.

public void add(const IloIntSetVar x)

This member function appends `x` to the invoking array.

public void add(const IloIntSetVarArray array)

This member function appends the elements in `array` to the invoking array.

public IloIntSetVar operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index `i`. Concert Technology uses the `const` operator

```java
IloIntSetVar operator[](IloInt i) const;

```java
```

on `const` arrays.
public IloIntSetVar & operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.
IloIntTernaryPredicate

Category          Class
InheritancePath

Definition File  ilconcert/ilotupleset.h

Constructor Summary

| public            | IloIntTernaryPredicate()          |
| public            | IloIntTernaryPredicate(IloIntTernaryPredicate I *) |

Method Summary

| public IloIntTernaryPredicateI * | getImpl()                        |
| public IloBool                  | isTrue(const IloInt,              |
|                                |   const IloInt,                   |
|                                |   const IloInt)                   |
| public void                    | operator=(const IloIntTernaryPredicate I) |

Description

This class makes it possible for you to define ternary predicates operating on arbitrary objects in a model. A predicate is an object with a member function (such as `isTrue`) that checks whether or not a property is satisfied by an ordered set of (pointers to) objects. A ternary predicate checks an ordered set of three objects.

Defining a New Class of Predicates

Predicates, like other Concert Technology objects, depend on two classes: a handle class, IloIntTernaryPredicate, and an implementation class, such as IloIntTernaryPredicateI, where an object of the handle class contains a data member (the handle pointer) that points to an object (its implementation object) of an instance of IloIntTernaryPredicateI allocated in a Concert Technology environment. As a Concert Technology user, you will be working primarily with handles.
If you define a new class of predicates yourself, you must define its implementation class together with the corresponding virtual member function isTrue, as well as a member function that returns an instance of the handle class IloIntTernaryPredicate.

Arity

As a developer, you can use predicates in Concert Technology applications to define your own constraints that have not already been predefined in Concert Technology. In that case, the arity of the predicate (that is, the number of constrained variables involved in the predicate, and thus the size of the array that the member function isTrue must check) must be three.

See Also

IloTableConstraint

Constructors

public IloIntTernaryPredicate()

This constructor creates an empty ternary predicate. In other words, the predicate is an empty handle with a null handle pointer. You must assign the elements of the predicate before you attempt to access it, just as you would any other pointer. An attempt to access it before this assignment will throw an exception (an instance of IloSolver::SolverErrorException).

public IloIntTernaryPredicate(IloIntTernaryPredicateI * impl)

This constructor creates a handle object (an instance of the class IloIntTernaryPredicate) from a pointer to an implementation object (an instance of the implementation class IloIntTernaryPredicateI).

Methods

public IloIntTernaryPredicateI * getImpl()

This member function returns a pointer to the implementation object of the invoking handle.

public IloBool isTrue(const IloInt val1, const IloInt val2, const IloInt val3)

This member function returns IloTrue if the values val1, val2, and val3 make the invoking ternary predicate valid. It returns IloFalse otherwise.

public void operator=(const IloIntTernaryPredicate & h)

This assignment operator copies h into the invoking predicate by assigning an address to the handle pointer of the invoking object. That address is the location of the implementation object of the argument h. After execution of this operator, both the invoking predicate and h point to the same implementation object.
IloIntTupleSet

Category     Class
InheritancePath

Definition File ilconcert/ilotupleset.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntTupleSet()</td>
</tr>
<tr>
<td>public</td>
<td>IloIntTupleSet(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntTupleSet(const IloEnv, const IloInt)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool</td>
<td>add(const IloIntArray)</td>
</tr>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isIn(const IloIntArray)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>remove(const IloIntArray)</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractable

IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId,
IloExtractable::getImpl, IloExtractable::getName,
IloExtractable::getObject, IloExtractable::setName,
IloExtractable::setObject

Description

A tuple is an ordered set of values represented by an array. A set of enumerated tuples in a model is represented by an instance of IloIntTupleSet. That is, the elements of a tuple set are tuples of enumerated values (such as pointers). The number of values in a tuple is known as the arity of the tuple, and the arity of the tuples in a set is called the
arity of the set. (In contrast, the number of tuples in the set is known as the cardinality of
the set.)

As a handle class, IloIntTupleSet manages certain set operations efficiently. In particular, elements can be added to such a set. It is also possible to search a given set with the member function \texttt{isIn} to see whether or not the set contains a given element.

In addition, a set of tuples can represent a constraint defined on a constrained variable, either as the set of allowed combinations of values of the constrained variable on which the constraint is defined, or as the set of forbidden combinations of values.

There are a few conventions governing tuple sets:

◆ When you create the set, you must specify the arity of the tuple-elements it contains.

◆ You use the member function \texttt{IloIntTupleSet::add} to add tuples to the set.

You can add tuples to the set in a model; you cannot add tuples to an instance of this class during a search, nor inside a constraint, nor inside a goal.

Concert Technology will throw an exception (an instance of \texttt{IloSolver::SolverErrorException}) if you attempt:

◆ to add a tuple with a different number of variables from the arity of the set;

◆ to search for a tuple with an arity different from the set arity.

You do not have to worry about memory allocation. If you respect these conventions, Concert Technology manages allocation and de-allocation transparently for you.

See Also

IIClntTupleSet documented in the ILOG Solver Reference Manual, IloIntTupleSetIterator, IloTableConstraint, IloExtractable

Constructors

\begin{verbatim}
public IloIntTupleSet()
This constructor creates an empty handle. You must initialize it before you use it.

public IloIntTupleSet(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloIntTupleSet(const IloEnv env,
                     const IloInt arity)
This constructor creates a set of tuples (an instance of the class IloIntTupleSet) with the arity indicated by arity.
\end{verbatim}

Methods

\begin{verbatim}
public IloBool add(const IloIntArray tuple)
This member function adds a tuple represented by the array \texttt{tuple} to the invoking set. If you attempt to add an element that is already in the set, that element will not be added again. Added elements are not copied; that is, there is no memory duplication. Concert Technology will throw an exception if the size of the array is not equal to the arity of the
\end{verbatim}
invoking set. You may use this member function to add tuples to the invoking set in a model; you may not add tuples in this way during a search, inside a constraint, or inside a goal. For those purposes, see IlcIntTupleSet, documented in the ILOG Solver Reference Manual.

public ImplClass * getImpl()

This member function returns a pointer to the implementation object of the invoking handle.

public IloBool isIn(const IloIntArray tuple)

This member function returns IloTrue if tuple belongs to the invoking set. Otherwise, it returns IloFalse. Concert Technology will throw an exception if the size of the array is not equal to the arity of the invoking set.

public IloBool remove(const IloIntArray tuple)

This member function removes tuple from the invoking set in a model. You may use this member function to remove tuples from the invoking set in a model; you may not remove tuples in this way during a search, inside a constraint, or inside a goal. For those purposes, see IlcIntTupleSet documented in the ILOG Solver Reference Manual.
**IloIntTupleSetIterator**

**Category**  
Class

**Inheritance Path**

**Definition File**  
ilconcert/ilotupleset.h

### Constructor Summary

| Public | IloIntTupleSetIterator(const IloEnv, IloIntTupleSet) |

### Method Summary

| Public IloBool | ok() |
| Public IloIntArray | operator *() |
| Public void | operator++() |

**Description**  
An instance of the class IloIntTupleSetIterator is an iterator that traverses the elements of a finite set of tuples of enumerated values (instance of IloIntTupleSet).

ILOG Solver Reference Manual

**See Also**  
IlcIntTupleSet documented in the IloIntTupleSet

**Constructors**

public **IloIntTupleSetIterator**(const IloEnv env, IloIntTupleSet tSet)

This constructor creates an iterator associated with tSet to traverse its elements.

**Methods**

public **IloBool** ok()

This member function returns IloTrue if there is a current element and invoking iterator points to it. Otherwise, it returns IloFalse.

To traverse the elements of a finite set of tuples, use the following code:

---

**ILOG Concert Technology 2.0 Reference Manual**  
226
IloIntTupleSet val;
for(IloIntTupleSetIterator iter(env, tset); iter.ok(); ++iter){
    val = *iter;
    // do something with val
}

public IloIntArray operator *()
This operator returns the current element, the one to which the invoking iterator points.

public void operator++()
This operator advances the iterator to point to the next tuple in the set.
IloIntVar

Category  Class

InheritancePath

Definition File  ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntVar()</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(ImplClass *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(IloEnv, IloInt, IloInt, const char *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(const IloAddNumVar &amp;, IloInt, IloInt, const char *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(const IloEnv, const IloIntArray, const char *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(const IloAddNumVar &amp;, const IloIntArray, const char *)</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>IloIntVar(const IloNumVar)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
<td>IloIntVar::getImpl()</td>
<td></td>
</tr>
</tbody>
</table>
An instance of this class represents a constrained integer variable in a Concert Technology model. If you are looking for a class of numeric variables that may assume integer values and may be relaxed to assume floating-point values, then consider the class `IloNumVar`. If you are looking for a class of binary decision variables (that is, variables that assume only the values 0 (zero) or 1 (one), then consider the class `IloBoolVar`.

**Description**

The lower and upper bound of an instance of this class is an integer.
What Is Extracted


An instance of IloIntVar is extracted by IloCplex (documented in the ILOG CPLEX Reference Manual) as a column representing a numeric variable of type Int with bounds as indicated by IloIntVar.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

Note: When numerical bounds are given to an integer variable (an IloIntVar or an IloNumVar with Type = Int) in the constructors of via a modifier (setUB, setLB, setBounds), they are inward rounded to an integer value. LB is rounded down and UB is rounded up.

See Also

IloBoolVar, IloNumVar

Constructors

public IloIntVar()

This constructor creates an empty handle. You must initialize it before you use it.

public IloIntVar(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloIntVar(IloEnv env,
    IloInt vmin,
    IloInt vmax,
    const char * name)

This constructor creates an instance of IloIntVar like this:

IloNumVar(env, vmin, vmax, ILOINT, name);

public IloIntVar(const IloAddNumVar & var,
    IloInt lowerBound,
    IloInt upperBound,
    const char * name)

This constructor creates an instance of IloIntVar like this:

IloNumVar(column, lowerBound, upperBound, ILOINT, name);
public **IloIntVar** (const **IloEnv** env,
            const **IloIntArray** values,
            const char * name)

This constructor calls upon its corresponding **IloNumVar** constructor.

public **IloIntVar** (const **IloAddNumVar** & var,
            const **IloIntArray** values,
            const char * name)

This constructor calls upon its corresponding **IloNumVar** constructor.

public **IloIntVar** (const **IloNumVar** var)

This constructor creates a new handle on **var** if it is of type **ILOINT**. Otherwise, an exception is thrown.

### Methods

**public ImplClass** * **getImpl**()

This member function returns a pointer to the implementation object of the invoking handle.

**public IloNum** **getLB**()

This member function returns the lower bound of the invoking variable.

**public IloInt** **getMax**()

This member function returns the maximal value of the invoking variable.

**public IloInt** **getMin**()

This member function returns the minimal value of the invoking variable.

**public IloNum** **getUB**()

This member function returns the upper bound of the invoking variable.

**public void** **setBounds** (**IloInt** lb,
            **IloInt** ub)

This member function sets **lb** as the lower bound and **ub** as the upper bound of the invoking numeric variable.

**Note:** The member function **setBounds** notifies Concert Technology algorithms about the change of bounds in this numeric variable.

**public void** **setLB** (**IloNum** min)

This member function sets **min** as the lower bound of the invoking variable.
public void setMax(IloInt max)
This member function returns the minimal value of the invoking variable to max.

Note: The member function setMax notifies Concert Technology algorithms about the change of bounds in this numeric variable.

public void setMin(IloInt min)
This member function returns the minimal value of the invoking variable to min.

Note: The member function setMin notifies Concert Technology algorithms about the change of bounds in this numeric variable.

public void setPossibleValues(const IloIntArray values)
This member function sets values as the domain of the invoking integer variable.

Note: The member function setPossibleValues notifies Concert Technology algorithms about the change of bounds in this numeric variable.

public void setUB(IloNum max)
This member function sets max as the upper bound of the invoking variable.

Note: The member function setUB notifies Concert Technology algorithms about the change of bounds in this numeric variable.
IloIntVarArray

Category
Class

InheritancePath

Definition File
ilconcert/iloexpression.h

Summary
The array class of IloIntVar.

Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Constructor Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloIntVarArray(IloDefaultArrayI *)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, IloInt)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, const IloIntArray, const IloIntArray)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, IloInt, const IloIntArray)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, IloInt, const IloIntArray)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, const IloNumColumnArray)</td>
<td></td>
</tr>
<tr>
<td>IloIntVarArray(const IloEnv, const IloNumColumnArray, const IloNumArray, const IloNumArray)</td>
<td></td>
</tr>
</tbody>
</table>
Description

For each basic type, Concert Technology defines a corresponding array class.
IloIntVarArray is the array class of the integer variable class for a model. It is a handle class.

Instances of IloIntVarArray are extensible.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.
See Also

**IloIntVar**

Constructors

public **IloIntVarArray**(IloDefaultArrayI * i)

This constructor creates an empty extensible array of integer variables.

public **IloIntVarArray**(const IloEnv env,
                           IloInt n)

This constructor creates an extensible array of \( n \) integer variables.

public **IloIntVarArray**(const IloEnv env,
                           const IloIntArray lb,
                           const IloIntArray ub)

This constructor creates an extensible array of integer variables with lower and upper bounds as indicated.

public **IloIntVarArray**(const IloEnv env,
                           IloInt lb,
                           const IloIntArray ub)

This constructor creates an extensible array of integer variables with a lower bound and an array of upper bounds as indicated.

public **IloIntVarArray**(const IloEnv env,
                           const IloIntArray lb,
                           IloInt ub)

This constructor creates an extensible array of integer variables with an array of lower bounds and an upper bound as indicated.

public **IloIntVarArray**(const IloEnv env,
                           IloInt n,
                           IloInt lb,
                           IloInt ub)

This constructor creates an extensible array of \( n \) integer variables, with a lower and an upper bound as indicated.

public **IloIntVarArray**(const IloEnv env,
                           const IloNumColumnArray columnarray)

This constructor creates an extensible array of integer variables from a column array.

public **IloIntVarArray**(const IloEnv env,
                           const IloNumColumnArray columnarray,
                           const IloNumArray lb,
                           const IloNumArray ub)

This constructor creates an extensible array of integer variables with lower and upper bounds as indicated from a column array.

public **IloIntVarArray**(const IloEnv env,
                           IloInt n,
                           const IloIntVar v0,
IloIntVarArray

```cpp
const IloIntVar v1,
...
```

This constructor creates an extensible array of \( n \) integer variables; the elements of the new array take the corresponding values, \( v_0, v_1, \ldots, v_{(n-1)} \).

**Methods**

```cpp
public void add(IloInt more,
    const IloIntVar x)
```

This member function appends \( x \) to the invoking array of integer variables; it appends \( x \) more times.

```cpp
public void add(const IloIntVar x)
```

This member function appends the value \( x \) to the invoking array.

```cpp
public void add(const IloIntVarArray x)
```

This member function appends the variables in the array \( x \) to the invoking array.

```cpp
public IloIntExprArg operator[](IloIntExprArg anIntegerVariable)
```

This subscripting operator returns an expression argument for use in a constraint or expression. For clarity, let’s call \( A \) the invoking array. When \( anIntegerVariable \) is bound to the value \( i \), the domain of the expression is the domain of \( A[i] \). More generally, the domain of the expression is the union of the domains of the expressions \( A[i] \) where the \( i \) are in the domain of \( anIntegerVariable \).

```cpp
public IloIntVar operator[](IloInt i)
```

This operator returns a reference to the object located in the invoking array at the position indicated by the index \( i \). Concert Technology uses the `const` operator

```cpp
IloIntVar operator[](IloInt i) const;
```

```cpp
on const arrays.
```

```cpp
public IloIntVar & operator[](IloInt i)
```

This operator returns a reference to the extractable object located in the invoking array at the position indicated by the index \( i \).

```cpp
public IloNumVarArray toNumVarArray()
```

This member function copies the invoking array into a new IloNumVarArray.
**IloInverse**

**Category**  
Class

**InheritancePath**

**Definition File**  
ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloInverse()</td>
</tr>
<tr>
<td>public</td>
<td>IloInverse(ImplClass *)</td>
</tr>
</tbody>
</table>
| public | IloInverse(const IloEnv,  
| | const IloIntVarArray,  
| | const IloIntVarArray,  
| | const char *) |

### Method Summary

| Public ImplClass * | getImpl() |

### Inherited methods from *IloConstraint*

IloConstraint::getImpl
Description

An instance of `IloInverse` represents an inverse constraint. Informally, we say that an inverse constraint works on two arrays, say, `f` and `invf`, so that an element of `f` composed with the corresponding element of `invf` produces the index of that element.

In formal terms, if the length of the array `f` is `n`, and the length of the array `invf` is `m`, then the inverse constraint insures that:

- for all `i` in the interval `[0, n-1]`, if `f[i]` is in `[0, m-1]` then `invf[f[i]] == i`;
- for all `j` in the interval `[0, m-1]`, if `invf[j]` is in `[0, n-1]` then `f[invf[j]] == j`.

In order for a constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert` and `NDEBUG`.

See Also

`IloConstraint`

Constructors

```
public IloInverse()
```

This constructor creates an empty handle. You must initialize it before you use it.
public **IloInverse**(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public **IloInverse**(const IloEnv  env,
       const IloIntVarArray  f,
       const IloIntVarArray  invf,
       const char * name)

This constructor creates an inverse constraint that if the length of the array \( f \) is \( n \), and the length of the array \( invf \) is \( m \), then this function returns a constraint that insures that:

- for all \( i \) in the interval \([0, n-1]\), if \( f[i] \) is in \([0, m-1]\) then \( invf[f[i]] == i \);
- for all \( j \) in the interval \([0, m-1]\), if \( invf[j] \) is in \([0, n-1]\) then \( f[invf[j]] == j \).

**Methods**

public ImplClass * **getImpl**()

This member function returns a pointer to the implementation object of the invoking handle.
IloIsNAN

**Category**  Global Function

**Definition File**  ilconcert/ilosys.h

**Synopsis**  

```c
public int IloIsNAN(double )
```

**Description**  
This function tests if a double value is a NaN (Not a number).
**IloIterator**

**Category**
Class

**InheritancePath**

![Inheritance Path](ilconcert/iloiterator.h)

**Definition File**
ilconcert/iloiterator.h

**Summary**
A template to create iterators for a class of extractable objects.

### Constructor Summary

| public | IloIterator(const IloEnv, IloBool) |

### Method Summary

| public IloBool | ok() |
| public void    | operator++() |

**Description**

This template creates iterators for a given class of extractable objects (denoted by E in the template) within an instance of IloEnv.

By default, an iterator created in this way will traverse instances of E and of its subclasses. You can prevent the iterator from traversing instances of subclasses of E (that is, you can limit its effect) by setting the parameter withSubClasses to IloFalse in the constructor of the iterator.

While an iterator created in this way is working, you must not create nor destroy any extractable objects in the instance of IloEnv where it is working. In other words, an iterator created in this way works only in a stable environment.

An iterator created with this template differs from an instance of IloModel::Iterator. An instance of IloModel::Iterator works only on extractable objects (instances of IloExtractable or its subclasses) that have explicitly been added to a model (an instance of IloModel). In contrast, an iterator created with this template will work on all extractable objects within a given environment, whether or not they have been explicitly added to a model.
See Also

- IloEnv
- IloExtractable
- IloModel
- IloModel::Iterator

Constructors

```cpp
public IloIterator(const IloEnv env,
                    IloBool withSubClasses)
```

This template constructor creates an iterator for instances of the class `E`. When the parameter `withSubClasses` is `IloTrue` (its default value), the iterator will also work on instances of the subclasses of `E`. When `withSubClasses` is `IloFalse`, the iterator works only on instances of `E`.

Methods

```cpp
public IloBool ok()
```

This member function returns `IloTrue` if there is a current element and the iterator points to it. Otherwise, it returns `IloFalse`.

```cpp
public void operator++()
```

This operator advances the iterator to point to the next value in the iteration.
**IloLog**

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**  
public IloNumExprArg IloLog(const IloNumExprArg arg)  
public IloNum IloLog(IloNum val)

**Description**  
Concert Technology offers predefined functions that return an expression from an algebraic function on expressions. These predefined functions also return a numeric value from an algebraic function on numeric values as well.

_IloLog returns the natural logarithm of its argument. In order to conform to IEEE 754 standards for floating-point arithmetic, you should use this function in your Concert Technology applications, rather than the standard C++ log._
IloMax

Category      Global Function

Definition File  ilconcert/iloexpression.h

Synopsis

public IloNum IloMax(const IloNumArray vals)
public IloNum IloMax(IloNum val1,
                     IloNum val2)
public IloInt IloMax(const IloIntArray vals)
public IloIntExprArg IloMax(const IloIntExprArray exprs)
public IloNumExprArg IloMax(const IloNumExprArray x,
                             const IloNumExprArg y)
public IloNumExprArg IloMax(const IloNumExprArg x,
                             IloNum y)
public IloNumExprArg IloMax(IloNum x,
                             const IloNumExprArg y)
public IloIntExprArg IloMax(const IloIntExprArg x,
                             const IloIntExprArg y)
public IloNumExprArg IloMax(const IloIntExprArg x,
                             IloInt y)
public IloNumExprArg IloMax(const IloIntExprArg x,
                             IloNum y)
public IloIntExprArg IloMax(const IloIntExprArg x,
                             const IloIntExprArg y)
public IloNumExprArg IloMax(IloInt x,
                             const IloIntExprArg y)
public IloNumExprArg IloMax(IloNum x,
                             const IloIntExprArg y)
public IloIntExprArg IloMax(int x,
                             const IloIntExprArg y)

Description

These functions compare their arguments and return the greatest value.
IloMaximize

**Category**
Global Function

**Definition File**
ilconcert/ilolinear.h

**Synopsis**

```c
public IloObjective IloMaximize(const IloEnv env,
                                 const IloNum constant,
                                 const char * name)
public IloObjective IloMaximize(const IloEnv env,
                                 const IloNumExprArg expr,
                                 const char * name)
```

**Description**

This function defines a maximization objective in a model. In other words, it simply offers a convenient way to create an instance of `IloObjective` with its sense defined as Maximize. However, an instance of `IloObjective` created by `IloMaximize` may not necessarily maintain its sense throughout the lifetime of the instance. The optional argument `name` is set to 0 by default.

You may define more than one objective in a model. However, algorithms conventionally take into account only one objective at a time.
IloMember

Category
Global Function

Definition File
ilconcert/iloanyset.h

Synopsis
public IloConstraint IloMember(const IloEnv env,
    const IloAnyVar var1,
    const IloAnySetVar var2)
public IloConstraint IloMember(const IloEnv env,
    IloAny val,
    const IloAnySetVar var2)
public IloConstraint IloMember(const IloEnv ,
    const IloNumExprArg expr,
    const IloNumArray elements)
public IloConstraint IloMember(const IloEnv ,
    const IloIntExprArg expr,
    const IloIntArray elements)
public IloConstraint IloMember(const IloEnv ,
    const IloIntVar var1,
    const IloIntSetVar var2)
public IloConstraint IloMember(const IloEnv ,
    IloInt var1,
    const IloIntSetVar var2)

Description
This function creates and returns a constraint (an instance of IloConstraint) for use in a model. The constraint forces element to be a member of setVar.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
**IloMin**

**Category**
Global Function

**Definition File**
ilconcert/iloexpression.h

**Synopsis**

```cpp
public IloNum IloMin(const IloNumArray vals)
public IloNum IloMin(IloNum val1,
                     IloNum val2)
public IloInt IloMin(const IloIntArray vals)
public IloIntExprArg IloMin(const IloIntExprArray exprs)
public IloNumExprArg IloMin(const IloNumExprArray exprs, const IloNumExprArg y)
public IloNumExprArg IloMin(const IloNumExprArg x,
                            const IloNumExprArg y)
public IloNumExprArg IloMin(const IloNum x, const IloNumExprArg y)
public IloIntExprArg IloMin(const IloIntExprArg x,
                            const IloIntExprArg y)
public IloNumExprArg IloMin(const IloIntExprArg x, IloInt y)
public IloNumExprArg IloMin(const IloIntExprArg x, IloNum y)
public IloIntExprArg IloMin(const IloIntExprArg x, int y)
public IloIntExprArg IloMin(IloInt x, const IloIntExprArg y)
public IloNumExprArg IloMin(IloNum x, const IloIntExprArg y)
public IloIntExprArg IloMin(int x, const IloIntExprArg y)
```

**Description**
These functions compare their arguments and return the least value. When its argument is an array, the function compares the elements of that array and returns the least value.
IloMinimize

Category: Global Function

Definition File: ilconcert/ilolinear.h

Synopsis:

```c
public IloObjective IloMinimize(const IloEnv  env,
                                  IloNum  constant,
                                  const char * name)

public IloObjective IloMinimize(const IloEnv  env,
                                  const IloNumExprArg  expr,
                                  const char * name)
```

Description:

This function defines a minimization objective in a model. In other words, it simply offers a convenient way to create an instance of `IloObjective` with its sense defined as Minimize. However, an instance of `IloObjective` created by `IloMinimize` may not necessarily maintain its sense throughout the lifetime of the instance. The optional argument `name` is set to 0 by default.

You may define more than one objective in a model. However, algorithms conventionally take into account only one objective at a time.
IloModel

**Category**  
Class

**Inheritance Path**

![Inheritance Diagram]

**Definition File**  
ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloModel()</td>
</tr>
<tr>
<td>public</td>
<td>IloModel(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloModel(const IloEnv, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public const</td>
<td>IloExtractableArray &amp; add(const IloExtractableArray &amp;)</td>
</tr>
<tr>
<td>public</td>
<td>IloExtractable add(const IloExtractable)</td>
</tr>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public void</td>
<td>remove(const IloExtractableArray)</td>
</tr>
<tr>
<td>public void</td>
<td>remove(const IloExtractable)</td>
</tr>
</tbody>
</table>

### Inherited methods from IloExtractable

- IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId,
- IloExtractable::getImpl, IloExtractable::getName,
- IloExtractable::getObject, IloExtractable::setName,
- IloExtractable::setObject
An instance of this class represents a model. A model consists of the extractable objects such as constraints, constrained variables, objectives, and possibly other modeling objects, that represent a problem. Concert Technology extracts information from a model and passes the information in an appropriate form to algorithms that solve the problem. (For information about extracting objects into algorithms, see the member function `IloAlgorithm::extract` and the template `IloAdd`.)

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

**Models and Submodels**

With Concert Technology, you may create more than one model in a given environment (an instance of `IloEnv`). In fact, you can create submodels. That is, you can add one model to another model within the same environment.

**What Is Extracted from a Model**

All the extractable objects (that is, instances of `IloExtractable` or one of its subclasses) that have been added to a model (an instance of `IloModel`) and that have not been removed from it will be extracted when an algorithm extracts the model. An instance of the nested class `IloModel::Iterator` accesses those extractable objects.

**See Also**

`IloEnv, IloExtractable, IloModel::Iterator`

**Constructors**

- **public `IloModel()`**
  
  This constructor creates an empty handle. You must initialize it before you use it.

- **public `IloModel(ImplClass * impl)`**
  
  This constructor creates a handle object from a pointer to an implementation object.

- **public `IloModel(const IloEnv env, const char * name)`**
  
  This constructor creates a model. By default, the name of the model is the empty string, but you can attribute a name to the model at its creation.

**Methods**

- **public const `IloExtractableArray & add(const IloExtractableArray & x)`**
  
  This member function adds the array of extractable objects to the invoking model.
public IloExtractable add(const IloExtractable x)
This member function adds the extractable object to the invoking model.

Note: The member function add notifies Concert Technology algorithms about this addition to the model.

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.

public void remove(const IloExtractableArray x)
This member function removes the array of extractable objects from the invoking model.

Note: The member function remove notifies Concert Technology algorithms about this removal from the model.

public void remove(const IloExtractable x)
This member function removes the extractable object from the invoking model.

Note: The member function remove notifies Concert Technology algorithms about this removal from the model.
IloModel::Iterator

Category Inner Class

InheritancePath

Definition File ilconcert/ilomodel.h

Constructor Summary

| Public | Iterator(const IloModel) |

Method Summary

| Public IloBool | ok() |
| Public IloExtractable | operator *() |
| Public void | operator++() |

Description

An instance of this nested class is an iterator capable of traversing the extractable objects in a model.

An iterator of this class differs from one created by the template IloIterator. Instances of IloIterator traverse all the extractable objects of a given class (indicated by E in the template) within a given environment (an instance of IloEnv), whether or not those extractable objects have been explicitly added to a model. Instances of IloModel::Iterator traverse only those extractable objects that have explicitly been added to a given model (an instance of IloModel).

See Also

IloIterator, IloModel

Constructors

public Iterator(const IloModel model)

This constructor creates an iterator to traverse the extractable objects in the model indicated by model.

Methods

public IloBool ok()

This member function returns IloTrue if there is a current element and the iterator points to it. Otherwise, it returns IloFalse.

public IloExtractable operator *()
This operator returns the current extractable object, the one to which the invoking iterator points.

```java
public void operator++()
```

This operator advances the iterator to point to the next extractable object in the model.
IloMonotonicDecreasingNumExpr

Category Global Function

Definition File ilconcert/iloexpression.h

Synopsis

public IloNumExprArg IloMonotonicDecreasingNumExpr(IloNumExprArg node,
IloNumFunction f,
IloNumFunction invf)

Description

This function creates a new constrained expression equal to \( f(x) \). The arguments \( f \) and \( invf \) must be pointers to functions of type IloNumFunction. Those two functions must be inverses of one another, that is,

\[
invf(f(x)) = x \text{ and } f(inv(x)) = x \text{ for all } x.
\]

Those two functions must also be monotonically decreasing.

IloMonotonicDecreasingNumExpr does not verify whether \( f \) and \( invf \) are inverses of one another. It does not verify whether they are monotonically decreasing either.

The effects of this function are reversible.
**IloMonotonicIncreasingNumExpr**

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**  
public IloNumExprArg IloMonotonicIncreasingNumExpr(IloNumExprArg node,  
IloNumFunction f,  
IloNumFunction invf)

**Description**  
This function creates a new constrained expression equal to \( f(x) \). The arguments \( f \) and \( \text{invf} \) must be pointers to functions of type IlcFloatFunction. Those two functions must be inverses of one another, that is,

\[
\text{invf}(f(x)) = x \quad \text{and} \quad f(\text{invf}(x)) = x \quad \text{for all} \quad x.
\]

IloMonotonicIncreasingNumExpr does not verify whether \( f \) and \( \text{invf} \) are inverses of one another. It does not verify whether they are monotonically increasing either.
IloMutexDeadlock

Category: Class

InheritancePath

Definition File: ilconcert/ilothread.h

Summary: The class of exceptions thrown due to mutex deadlock.

Description: This is the class of exceptions thrown if two or more threads become deadlocked waiting for a mutex owned by the other(s).
IloMutexNotOwner

Category  Class

InheritancePath

Definition File  ilconcert/ilothread.h

Summary  The class of exceptions thrown.

Description  The class of exceptions thrown if a thread attempts to unlock a mutex that it does not own.
IloMutexProblem

Category  Class
InheritancePath

Definition File  ilconcert/ilothread.h

Constructor Summary
| public | IloMutexProblem(const char *) |

Description
The class IloMutexProblem is part of the hierarchy of classes representing exceptions in Concert Technology. Concert Technology uses instances of this class when an error occurs with respect to a mutex, an instance of IloFastMutex.

An exception is thrown; it is not allocated in a Concert Technology environment; it is not allocated on the C++ heap. It is not necessary for you as a programmer to delete an exception explicitly. Instead, the system calls the constructor of the exception to create it, and the system calls the destructor of the exception to delete it.

When exceptions are enabled on a platform that supports C++ exceptions, an instance of IloMutexProblem makes it possible for Concert Technology to throw an exception in case of error. On platforms that do not support C++ exceptions, an instance of this class makes it possible for Concert Technology to exit in case of error.

Throwing and Catching Exceptions
Exceptions are thrown by value. They are not allocated on the C++ heap, nor in a Concert Technology environment. The correct way to catch an exception is to catch a reference to the error (indicated by the ampersand &), like this:

```
catch(IloMutexProblem& error);
```

See Also  IloException, IloFastMutex
Constructors

public IloMutexProblem(const char * msg)

This constructor creates an instance of IloMutexProblem to represent an exception in case of an error involving a mutex. This instance is not allocated on C++ heap; it is not allocated in a Concert Technology environment either.
IloNot

**Category**  
Class

**Inheritance Path**

**Definition File**  
ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloNot()</td>
</tr>
<tr>
<td>public</td>
<td>IloNot(ImplClass *)</td>
</tr>
</tbody>
</table>

### Method Summary

| Public ImplClass * | getImpl() |

### Inherited methods from **IloConstraint**

IloConstraint::getImpl

### Inherited methods from **IloIntExprArg**

IloIntExprArg::getImpl
The class `IloNot` represents a constraint that is the negation of its argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.

**See Also**
- `operator!`

**Constructors**

```cpp
public IloNot()
```

This constructor creates an empty handle. You must initialize it before you use it.

```cpp
public IloNot(ImplClass * impl)
```

This constructor creates a handle object from a pointer to an implementation object.

**Methods**

```cpp
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.
IloNotMember

Category: Global Function

Definition File: ilconcert/iloonyset.h

Synopsis:

public IloConstraint IloNotMember(const IloEnv env,
    const IloAnyVar var1,
    const IloAnySetVar var2)

public IloConstraint IloNotMember(const IloEnv env,
    IloAny val,
    const IloAnySetVar var2)

public IloConstraint IloNotMember(const IloEnv ,
    const IloIntVar var1,
    const IloIntSetVar var2)

public IloConstraint IloNotMember(const IloEnv ,
    IloInt var1,
    const IloIntSetVar var2)

Description:

This function creates and returns a constraint (an instance of IloConstraint) for use in a model. The constraint forces element not to be a member of setVar.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
IloNullIntersect

Category: Global Function
Definition File: ilconcert/iloanyset.h

Synopsis:

```java
public IloConstraint IloNullIntersect(const IloEnv env,
const IloAnySetVar var1,
const IloAnySetVar var2)
public IloConstraint IloNullIntersect(const IloEnv env,
const IloAnySet var1,
const IloAnySetVar var2)
public IloConstraint IloNullIntersect(const IloEnv env,
const IloAnySetVar var1,
const IloAnySet var2)
public IloConstraint IloNullIntersect(const IloEnv ,
const IloIntSetVar var1,
const IloIntSetVar var2)
public IloConstraint IloNullIntersect(const IloEnv ,
const IloIntSet var1,
const IloIntSetVar var2)
public IloConstraint IloNullIntersect(const IloEnv ,
const IloIntSetVar var1,
const IloIntSet var2)
```

Description:

This function creates and returns a constraint (an instance of `IloConstraint`) for use in a model. The constraint forces the set `var1` to have no elements in common with the set `var2`. In other words, the intersection of `var1` with `var2` will be empty when this constraint is satisfied.

In order for the constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.
IloNum

Category       Type Definition
Definition File ilconcert/ilosys.h
Synopsis       double IloNum
Description    This type definition represents numeric values as floating-point numbers in Concert Technology.
See Also       IloModel, IloInt
**IloNumArray**

**Category**  
Class

**Inheritance Path**

**Definition File**  
ilconcert/iloenv.h

**Constructor Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public IloNumArray(IloArrayI *)</code></td>
<td></td>
</tr>
<tr>
<td><code>public IloNumArray(const IloNumArray &amp;)</code></td>
<td></td>
</tr>
<tr>
<td><code>public IloNumArray(const IloEnv, IloInt)</code></td>
<td></td>
</tr>
<tr>
<td><code>public IloNumArray(const IloEnv, IloInt, IloNum, IloNum, ...)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public IloNumExprArg operator()(IloNumExprArg)</code></td>
<td></td>
</tr>
<tr>
<td><code>public IloIntArray toIntArray()</code></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

For each basic type, Concert Technology defines a corresponding array class.  
IloNumArray is the array class of the basic floating-point class (IloNum) for a model.

Instances of IloNumArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

**See Also**

IloNum, operator>>, operator
Constructors

public IloNumArray(IloArrayI * i)
This constructor creates an empty array of floating-point numbers for use in a model. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloNumArray(const IloNumArray & cpy)
This copy constructor creates a handle to the array of floating-point objects indicated by copy.

public IloNumArray(const IloEnv env, IloInt n)
This constructor creates an array of \( n \) elements. Initially, the \( n \) elements are empty handles.

public IloNumArray(const IloEnv env, IloInt n, IloNum f0, IloNum f1, ...)
This constructor creates an array of \( n \) floating-point objects for use in a model.

Methods

public IloNumExprArg operator()(IloNumExprArg)
This subscripting operator returns an expression node for use in a constraint or expression. For clarity, let's call \( A \) the invoking array. When an \( \text{anIntegerVariable} \) is bound to the value \( i \), then the domain of the expression is the domain of \( A[i] \). More generally, the domain of the expression is the union of the domains of the expressions \( A[i] \) where the \( i \) are in the domain of an \( \text{anIntegerVariable} \).

This operator is also known as an element constraint.

public IloIntArray toIntArray()
This member function copies the invoking array to a new IloIntVarArray, checking the type of the variables during the copy.
IloNumColumn

Category: Class
InheritancePath:

Definition File: ilconcert/iloexpression.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumColumn (const IloEnv)</td>
</tr>
<tr>
<td>public IloNumColumn (const IloAddNumVar &amp;)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void clear()</td>
</tr>
<tr>
<td>public IloNumColumn &amp;</td>
</tr>
<tr>
<td>operator const IloAddNumVar &amp;()</td>
</tr>
<tr>
<td>public IloNumColumn &amp;</td>
</tr>
<tr>
<td>operator=(const IloAddValueToRange &amp;)</td>
</tr>
<tr>
<td>public IloNumColumn &amp;</td>
</tr>
<tr>
<td>operator+=(const IloAddNumVar &amp;)</td>
</tr>
<tr>
<td>public IloNumColumn &amp;</td>
</tr>
<tr>
<td>operator+=(const IloNumColumn &amp;)</td>
</tr>
</tbody>
</table>

Description

An instance of this class helps you design a model through column representation. In other words, you can create a model by defining each of its columns as an instance of this class. In particular, an instance of IloNumColumn enables you to build a column for a numeric variable (an instance of IloNumVar) with information about the extractable objects (such as objectives, constraints, etc.) where that numeric variable may eventually appear, even if the numeric variable has not yet been created.

Usually you populate a column (an instance of this class) with objects returned by the operator() of the class (such as IloObjective::operator()) where you want to install the newly created variable, as in the examples below.

An instance of IloNumColumn keeps a list of those objects returned by operator(). In other words, an instance of IloNumColumn knows the extractable objects where a numeric variable will be added when it is created.
When you create a new instance of `IloNumVar` with an instance of `IloNumColumn` as a parameter, then Concert Technology adds the newly created numeric variable to all the extractable objects (such as constraints, ranges, objectives, etc.) for which an instance of `IloAddNumVar` will be added to this instance of `IloNumColumn`.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

For information on columnwise modeling, see the concept `Column-Wise Modeling`.

**See Also**  
`IloNumVar, IloObjective, IloRange`

**Constructors**

```cpp
public IloNumColumn(const IloEnv env)
```

This constructor creates an empty column in the environment `env`.

```cpp
public IloNumColumn(const IloAddNumVar & var)
```

This constructor creates a column and adds `var` to it.

**Methods**

```cpp
public void clear()
```

This member function removes (from the invoking column) its list of extractable objects.

```cpp
public operator const IloAddNumVar &()
```

This casting operator allows you to use instances of `IloNumColumn` in column expressions. It accepts an extractable object, such as an objective (an instance of `IloObjective`) or a constraint (an instance of `IloConstraint`). It returns the object derived from `IloAddNumVar` and needed to represent the extractable object in column format.

```cpp
public IloNumColumn & operator+= (const IloAddValueToRange & rhs)
```

This operator adds the appropriate instances of `IloAddValueToRange` for the right-hand side `rhs` to the invoking column.

**Examples:**

To use an instance of this class to create a column with a coefficient of 2 in the objective, with 10 in `range1`, and with 3 in `range2`, set:

```cpp
IloNumColumn col = obj(2) + range1(10) + range2(3);
```

To use an instance of this class to create a numeric variable corresponding to the column with lower bound 0 (zero) and upper bound 10:
IloNumVar var(env, col, 0, 10);

Another example:

IloNumColumn col1(env);
IloNumColumn col2 = rng7(3.1415);
coll += obj(1.0);
coll += rng(-12.0);
coll += rng2(13.7) + rng3(14.7);
coll += col1;

public IloNumColumn & operator+=(const IloAddNumVar & rhs)
This operator adds the appropriate instances of IloAddNumVar for the right-hand side rhs to the invoking column.

public IloNumColumn & operator+=(const IloNumColumn & rhs)
This operator assigns the right-hand side rhs to the invoking column.
IloNumColumnArray

**Category** Class

**Inheritance Path**

**Definition File** ilconcert/iloexpression.h

---

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumColumnArray(IloDefaultArrayI *)</td>
<td></td>
</tr>
<tr>
<td>public IloNumColumnArray(const IloEnv, IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloNumColumnArray(const IloNumColumnArray &amp;IloNumColumnArray)</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

For each basic type, Concert Technology defines a corresponding array class. IloNumColumnArray is the array class of the column representation class for a model. The implementation class for IloNumColumnArray is the undocumented class IloNumColumnArrayI.

Instances of IloNumColumnArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added or removed from the array.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

**See Also** IloModel, IloNumColumn

**Constructors**

public IloNumColumnArray(IloDefaultArrayI *i)

This constructor creates an empty extensible array of columns. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloNumColumnArray(const IloEnv env, IloInt n)
This constructor creates an array of \( n \) elements. Initially, the \( n \) elements are empty handles.

```java
public IloNumColumnArray(const IloNumColumnArray & h)
```

This copy constructor creates a handle to the array of column objects indicated by `copy`.
IloNumExpr

**Category**: Class

**Inheritance Path**

```
  IloExtractable
    IloNumExprArg
      IloNumExpr
        IloExpr
```

**Definition File**: ilconcert/iloexpression.h

**Summary**: The class of numeric expressions in a Concert model.

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloNumExpr()</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(const IloNumExprArg)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(const IloEnv, IloNum)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(const IloNumLinExprTerm)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(const IloIntLinExprTerm)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr(const IloExpr &amp;)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloNumExpr::getImpl()</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator *=(IloNum)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator++(const IloNumExprArg)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator++=(IloNum)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator--(const IloNumExprArg)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator--=(IloNum)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumExpr::operator/=(IloNum)</td>
</tr>
</tbody>
</table>
Description

Numeric expressions in Concert Technology 2.0 are represented using the class IloNumExpr.

Constructors

public **IloNumExpr**()

This constructor creates an empty handle. You must initialize it before you use it.

public **IloNumExpr**(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public **IloNumExpr**(const IloNumExprArg expr)

This constructor creates a numeric expression using the undocumented class IloNumExprArg.

public **IloNumExpr**(const IloEnv env, IloNum)

This constructor creates a numeric expression in the environment indicated by env. It may be used to build other expressions from variables belonging to env.

public **IloNumExpr**(const IloNumLinExprTerm term)

This constructor creates a numeric expression using the undocumented class IloNumLinExprTerm.
public **IloNumExpr** (const IloIntLinExprTerm term)

This constructor creates a numeric expression using the undocumented class IloIntLinExprTerm.

public **IloNumExpr** (const IloExpr & expr)

This is the copy constructor for this class.

---

**Methods**

public ImplClass * **getImpl**()

This member function returns a pointer to the implementation object of the invoking handle.

public IloNumExpr & **operator *=**(IloNum val)

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x * ...`

public IloNumExpr & **operator**++**(const IloNumExprArg expr)**

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x + ...`

public IloNumExpr & **operator**++**(IloNum val)**

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x + ...`

public IloNumExpr & **operator**--**(const IloNumExprArg expr)**

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x - ...`

public IloNumExpr & **operator**--**(IloNum val)**

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x - ...`

public IloNumExpr & **operator**/**(IloNum val)**

This operator is recommended for building a Concert Technology expression in a loop. It is more efficient than `x = x / ...`
IloNumExprArg

Category Class

InheritancePath

Definition File ilconcert/iloexpression.h

Summary A class used internally in Concert Technology.

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumExprArg()</td>
</tr>
<tr>
<td>public IloNumExprArg(ImpIClass *)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImpIClass * IloNumExprArg::getImpl()</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractable

- IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId,
- IloExtractable::getImp, IloExtractable::getName,
- IloExtractable::getObject, IloExtractable::setName,
- IloExtractable::setObject
IloNumExprArg

Description
Concert Technology uses instances of this class internally as temporary objects when it is parsing a C++ expression in order to build an instance of IloNumExpr. As a Concert Technology user, you will not need this class yourself; in fact, you should not use them directly. They are documented here because the return value of certain functions, such as IloSum or IloScalProd, can be an instance of this class.

Constructors
public IloNumExprArg()
This constructor creates an empty handle. You must initialize it before you use it.

public IloNumExprArg(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

Methods
public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.
IloNumExprArray

Category Class

InheritancePath

Definition File ilconcert/iloexpression.h

Summary The class IloNumExprArray.

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloNumExprArray(IloDefaultArray1 *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloNumExprArray(const IloEnv, IloInt)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloNumExprArray::add(IloInt, const IloNumExpr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloNumExprArray::add(const IloNumExpr)</td>
</tr>
<tr>
<td>Public</td>
<td>IloNumExprArray::add(const IloNumExprArray)</td>
</tr>
<tr>
<td>Public</td>
<td>IloNumExprArg</td>
</tr>
</tbody>
</table>

Inherited methods from IloExtractableArray

IloExtractableArray::add, IloExtractableArray::add,
For each basic type, Concert Technology defines a corresponding array class. **IloNumExprArray** is the array class of the numeric expressions class (**IloNumExpr**) for a model.

Instances of **IloNumExprArray** are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

**Constructors**

```java
public IloNumExprArray(IloDefaultArrayI * i)
```

This constructor creates an empty array of numeric expressions for use in a model. You cannot create instances of the undocumented class **IloDefaultArrayI**. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

```java
public IloNumExprArray(const IloEnv env,
                        IloInt n)
```

This constructor creates an array of `n` elements. Initially, the `n` elements are empty handles.

**Methods**

```java
public void add(IloInt more,
                const IloNumExpr x)
```

This member function appends `x` to the invoking array. The parameter `more` indicates how many times.

```java
public void add(const IloNumExpr x)
```

This member function appends `x` to the invoking array.

```java
public void add(const IloNumExprArray array)
```

This member function appends the elements in `array` to the invoking array.

```java
public IloNumExprArg operator[](IloNumExprArg anIntegerVariable)
```

This subscripting operator returns an expression argument for use in a constraint or expression. For clarity, let's call `A` the invoking array. When `anIntegerVariable` is bound to the value `i`, the domain of the expression is the domain of `A[i]`. More generally, the domain of the expression is the union of the domains of the expressions `A[i]` where the `i` are in the domain of `anIntegerVariable`. 
IloNumExpr::NonLinearExpression

Category Inner Class

InheritancePath

Definition File ilconcert/iloexpression.h

Summary The class of exceptions thrown if a numerical constant of a non-linear expression is set or queried.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public const IloNumExprArg getExpression()</td>
</tr>
</tbody>
</table>

Inherited methods from IloException

IloException::end, IloException::getMessage

DescriptionMethods

public const IloNumExprArg getExpression()

The member function getExprUsed returns the expression involved in the exception.
IloNumFunction

**Category**  
Type Definition

**Definition File**  
ilconcert/ilosys.h

**Synopsis**  
IloNum(* IloNumFunction)(IloNum)

**Description**  
This C++ type represents a pointer to a function that takes an argument of type IloNum and returns an object of type IloNum. This type is the type of the function passed as a parameter to the function IloMonotonicDecreasingNumExpr.

**See Also**  
IloMonotonicDecreasingNumExpr, IloMonotonicIncreasingNumExpr
IloNumVar

Category    Class

InheritancePath

Definition File    ilconcert/iloexpression.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumVar()</td>
</tr>
<tr>
<td>public IloNumVar(ImplClass *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloEnv, IloNum, IloNum, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloEnv, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloAddNumVar &amp;, IloNum, IloNum, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloEnv, const IloNumArray, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloAddNumVar &amp;, const IloNumArray, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public IloNumVar(const IloConstraint)</td>
</tr>
</tbody>
</table>
An instance of this class represents a numeric variable in a model. A numeric variable may be either an integer variable or a floating-point variable; that is, a numeric variable has a type, a value of the nested enumeration `IloNumVar::Type`. By default, its type is `Float`. It also has a lower and upper bound. A numeric variable cannot assume values less than its lower bound, nor greater than its upper bound.

If you are looking for a class of variables that can assume only constrained integer values, consider the class `IloIntVar`. If you are looking for a class of binary decision
variables that can assume only the values 0 (zero) or 1 (one), then consider the class 
IloBoolVar.

Most member functions in this class contain assert statements. For an explanation of 
the macro NDEBUG (a way to turn on or turn off these assert statements), see the 
concept Assert and NDEBUG.

Programming Hint

For each enumerated value in the nested enumeration IloNumVar::Type, Concert 
Technology offers an equivalent predefined C++ #define to make programming 
easier. For example, in your applications, you may write either statement:

```cpp
IloNumVar x(env, 0, 17, IloNumVar::Int); // using the enumeration
IloNumVar x(env, 0, 17, ILOINT);         // using the #define
```

See Also

IloBoolVar, IloIntVar, IloModel, IloNumVarArray, 
IloNumVar::Type

Constructors

public IloNumVar()

This constructor creates an empty handle. You must initialize it before you use it.

public IloNumVar(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloNumVar(const IloEnv env, 
IloNum lb, 
IloNum ub, 
IloNumVar::Type type, 
const char * name)

This constructor creates a constrained numeric variable and makes it part of the 
environment env. By default, the numeric variable ranges from 0.0 (zero) to the 
symbolic constant IloInfinity, but you can indicate other upper and lower bounds 
yourself. By default, the numeric variable assumes floating-point values. However, you 
can constrain it to be an integer by setting its type = Int. By default, its name is 
indicated by the empty string, but you can indicate a name of your own choice.

public IloNumVar(const IloEnv env,
This constructor creates a constrained numeric variable and makes it part of the environment env. The bounds of the variable are set by lowerBound and upperBound. By default, its name is indicated by the empty string, but you can indicate a name of your own choice.

public IloNumVar(const IloAddNumVar & var,
                 IloNum lowerBound,
                 IloNum upperBound,
                 IloNumVar::Type type,
                 const char * name)

This constructor creates a constrained numeric variable in column format. For more information on adding columns to a model, refer to the concept Column-Wise Modeling.

public IloNumVar(const IloEnv env,
                 const IloNumArray values,
                 IloNumVar::Type type,
                 const char * name)

This constructor creates a constrained discrete numeric variable and makes it part of the environment env. The new discrete variable will be limited to values in the set indicated by the array values. By default, its name is indicated by the empty string, but you can indicate a name of your own choice. You can use the member function setPossibleValues with instances created by this constructor.

public IloNumVar(const IloAddNumVar & var,
                 const IloNumArray values,
                 IloNumVar::Type type,
                 const char * name)

This constructor creates a constrained discrete numeric variable from var by limiting its domain to the values indicated in the array values. You may use the member function setPossibleValues with instances created by this constructor.

public IloNumVar(const IloConstraint constraint)

This constructor creates a constrained numeric variable which is equal to the truth value of constraint. The truth value of constraint is either 0 for IloFalse or 1 for IloTrue. You can use this constructor to cast a constraint to a constrained numeric variable. That constrained numeric variable can then be used like any other constrained numeric variable. It is thus possible to express sums of constraints, for example. The following line expresses the idea that all three variables cannot be equal:

```cpp
model.add((x != y) + (y != z) + (z != x) >= 2);
```

### Methods

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.

public IloNum getLB()

This member function returns the lower bound of the invoking numeric variable.

public void getPossibleValues(IloNumArray values)

This member function accesses the possible values of the invoking numeric variable and puts them in the array values.

public IloNumVar::Type getType()

This member function returns the type of the invoking numeric variable, indicating whether it is integer (Int) or floating-point (Float).

public IloNum getUB()

This member function returns the upper bound of the invoking numeric variable.

public void setBounds(IloNum lb, IloNum ub)

This member function sets lb as the lower bound and ub as the upper bound of the invoking numeric variable.

**Note:** The member function `setBounds` notifies Concert Technology algorithms about this change of bounds in this numeric variable.

public void setLB(IloNum num)

This member function sets num as the lower bound of the invoking numeric variable.

**Note:** The member function `setLB` notifies Concert Technology algorithms about this change of bound in this numeric variable.

public void setPossibleValues(const IloNumArray values)

This member function sets values as the domain of the invoking discrete numeric variable. This member function can be called only on instances of IloNumVar that have been created with one of the two discrete constructors; that is, instances of IloNumVar which have been defined by an explicit array of discrete values.
public void setUB(IloNum num)

This member function sets num as the upper bound of the invoking numeric variable.

**Note:** The member function `setPossibleValues` notifies Concert Technology algorithms about this change of domain in this discrete numeric variable.

**Note:** The member function `setUB` notifies Concert Technology algorithms about this change of bound in this numeric variable.
IloNumVarArray

**Category**  Class

**InheritancePath**

![Class Inheritance Diagram]

**Definition File**  ilconcert/iloexpression.h

**Summary**  The array class of IloNumVar.

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumVarArray(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, const IloNumArray, const IloNumArray, IloNumVar::Type)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, IloNum, const IloNumArray, IloNumVar::Type)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, const IloNumArray, IloNum, IloNumVar::Type)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, IloInt, IloNum, IloNumVar::Type)</td>
</tr>
<tr>
<td>public IloNumVarArray(const IloEnv, const IloNumColumnArray, IloNumVar::Type)</td>
</tr>
</tbody>
</table>
IloNumVarArray

**Description**

For each basic type, Concert Technology defines a corresponding array class. `IloNumVarArray` is the array class of the numeric variable class for a model.

Instances of `IloNumVarArray` are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(IloInt, const IloNumVar)</code></td>
<td>Add an element to the array</td>
</tr>
<tr>
<td><code>add(const IloNumVar)</code></td>
<td>Add an element to the array</td>
</tr>
<tr>
<td><code>add(const IloNumVarArray)</code></td>
<td>Add an element to the array</td>
</tr>
<tr>
<td><code>operator() (IloNumExprArg)</code></td>
<td>Operator overload</td>
</tr>
<tr>
<td><code>setBounds(const IloNumArray, const IloNumArray)</code></td>
<td>Set bounds for the array</td>
</tr>
<tr>
<td><code>toIntExprArray()</code></td>
<td>Convert to integer expression array</td>
</tr>
<tr>
<td><code>toIntVarArray()</code></td>
<td>Convert to integer variable array</td>
</tr>
<tr>
<td><code>toNumExprArray()</code></td>
<td>Convert to numeric expression array</td>
</tr>
</tbody>
</table>

**Inherited methods from IloNumExprArray**

- `add`, `add`, `add`, `operator()`

**Inherited methods from IloExtractableArray**

- `add`, `add`, `endElements`
Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

### See Also

- `IloAllDiff`, `IloModel`, `IloNumVar`, `operator>>`, `operator` 

### Constructors

**public `IloNumVarArray` (IloDefaultArrayI * i)**

This constructor creates an empty extensible array of numeric variables. You cannot create instances of the undocumented class `IloDefaultArrayI`. As a parameter in this default constructor, it allows you to pass 0 (zero) optionally or to use 0 (zero) as a default value of a parameter in a constructor.

**public `IloNumVarArray` (const IloEnv `env`, IloInt `n`)**

This constructor creates an extensible array of `n` numeric variables in `env`. Initially, the `n` elements are empty handles.

**public `IloNumVarArray` (const IloEnv `env`, const IloNumArray `lb`, const IloNumArray `ub`, IloNumVar::Type `type`)**

This constructor creates an extensible array of numeric variables in `env` with lower and upper bounds and type as indicated. The instances of `IloNumVar` to fill this array are constructed at the same time. The length of the array `lb` must be the same as the length of the array `ub`. In other words, `lb.getSize == ub.getSize`. This constructor will construct an array with the number of elements in the array `ub`.

**public `IloNumVarArray` (const IloEnv `env`, IloNum `lb`, const IloNumArray `ub`, IloNumVar::Type `type`)**

This constructor creates an extensible array of numeric variables in `env` with lower and upper bounds and type as indicated. The instances of `IloNumVar` to fill this array are constructed at the same time. The length of the new array will be the same as the length of the array `ub`.

**public `IloNumVarArray` (const IloEnv `env`, const IloNumArray `lb`, IloNum `ub`, IloNumVar::Type `type`)**

This constructor creates an extensible array of numeric variables in `env` with lower and upper bounds and type as indicated. The instances of `IloNumVar` to fill this array are constructed at the same time.

**public `IloNumVarArray` (const IloEnv `env`, IloInt `n`,**
This constructor creates an extensible array of \( n \) numeric variables in \( \text{env} \) with lower and upper bounds and type as indicated. The instances of \( \text{IloNumVar} \) to fill this array are constructed at the same time.

\[
\text{public } \text{IloNumVarArray}(\text{const IloEnv } \text{env},
\text{const IloNumColumnArray } \text{columnarray},
\text{IloNumVar::Type } \text{type})
\]

This constructor creates an extensible array of numeric variables with type as indicated. The instances of \( \text{IloNumVar} \) to fill this array are constructed at the same time.

\[
\text{public } \text{IloNumVarArray}(\text{const IloEnv } \text{env},
\text{const IloNumColumnArray } \text{columnarray},
\text{const IloNumArray } \text{lb},
\text{const IloNumArray } \text{ub},
\text{IloNumVar::Type } \text{type})
\]

This constructor creates an extensible array of numeric variables with lower and upper bounds and type as indicated. The instances of \( \text{IloNumVar} \) to fill this array are constructed at the same time.

\[
\text{Methods}
\]

\[
\text{public void } \text{add}(\text{IloInt } \text{more},
\text{const IloNumVar } \text{x})
\]

This member function appends \( x \) to the invoking array multiple times. The parameter \( \text{more} \) indicates how many times.

\[
\text{public void } \text{add}(\text{const IloNumVar } \text{x})
\]

This member function appends \( x \) to the invoking array.

\[
\text{public void } \text{add}(\text{const IloNumVarArray } \text{array})
\]

This member function appends the elements in \( \text{array} \) to the invoking array.

\[
\text{public IloNumExprArg } \text{operator}()\text{(IloNumExprArg } \text{anIntegerVariable})
\]

This subscripting operator returns an expression argument for use in a constraint or expression. For clarity, let's call \( A \) the invoking array. When \( \text{anIntegerVariable} \) is bound to the value \( i \), the domain of the expression is the domain of \( A[i] \). More generally, the domain of the expression is the union of the domains of the expressions \( A[i] \) where the \( i \) are in the domain of \( \text{anIntegerVariable} \).

\[
\text{public void } \text{setBounds}(\text{const IloNumArray } \text{lb},
\text{const IloNumArray } \text{ub})
\]

For each element in the invoking array, this member function sets \( \text{lb} \) as the lower bound and \( \text{ub} \) as the upper bound of the corresponding numeric variable in the invoking array.
public IloIntExprArray toIntExprArray()

This member function copies the invoking array to a new IloIntExprArray, checking the type of the variables during the copy.

public IloIntVarArray toIntVarArray()

This member function copies the invoking array to a new IloIntVarArray, checking the type of the variables during the copy.

public IloNumExprArray toNumExprArray()

This member function copies the invoking array to a new IloNumExprArray, checking the type of the variables during the copy.

Note: The member function setBounds notifies Concert Technology algorithms about this change of bounds in this array of numeric variables.
**IloNumVar::Type**

**Category**INNER Enumeration

**Definition File**ilconcert/iloexpression.h

**Synopsis**

```cpp
Type{
  Int,
  Float,
  Bool
};
```

**Description**

This nested enumeration enables you to indicate whether an instance of IloNumVar is of type integer (Int), Boolean (Bool), or floating-point (Float).

**Programming Hint**

For each enumerated value in IloNumVar::Type, there is a predefined equivalent C++ #define in the Concert Technology include files to make programming easier. For example, instead of writing IloNumVar::Int in your application, you can write ILOINT. Likewise, ILOFLOAT is defined for IloNumVar::Float and ILOBOOL for IloNumVar::Bool.

**See Also**IloNumVar

**Fields**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>Float</td>
</tr>
<tr>
<td></td>
<td>Bool</td>
</tr>
</tbody>
</table>
IloObjective

Category        Class
InheritancePath

Definition File ilconcert/ilolinear.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloObjective()</td>
</tr>
<tr>
<td>public IloObjective(ImplClass *)</td>
</tr>
<tr>
<td>public IloObjective(const IloEnv, IloNum, IloObjective::Sense, const char *)</td>
</tr>
<tr>
<td>public IloObjective(const IloEnv, const IloNumExprArg, IloObjective::Sense, const char *)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum getConstant()</td>
</tr>
<tr>
<td>public IloNumExprArg getExpr()</td>
</tr>
<tr>
<td>public ImplClass * getImpl()</td>
</tr>
<tr>
<td>public IloObjective::Sense getSense()</td>
</tr>
<tr>
<td>public IloObjective::operator()(IloNum)</td>
</tr>
<tr>
<td>public void setConstant(IloNum)</td>
</tr>
<tr>
<td>public void setExpr(const IloNumExprArg)</td>
</tr>
<tr>
<td>public void setLinearCoef(const IloNumVar, IloNum)</td>
</tr>
<tr>
<td>public void setLinearCoefs(const IloNumVarArray, const IloNumArray)</td>
</tr>
</tbody>
</table>
**Description**

An instance of this class is an objective in a model. An objective consists of its sense (indicating whether it is a minimization or maximization) and an expression. The expression may be a constant.

An objective belongs to the environment that the variables in its expression belong to. Generally, you will create an objective, add it to a model, and extract the model for an algorithm.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept Assert and NDEBUG.

**What Is Extracted**

All the variables (that is, instances of `IloNumVar` or one of its subclasses) in the objective (an instance of `IloObjective`) will be extracted when an algorithm such as `IloCplex`, documented in the ILOG CPLEX Reference Manual, extracts the objective.

**Multiple Objectives**

You may create more than one objective in a model, for example, by creating more than one group. However, certain algorithms, such as an instance of `IloCplex`, will throw an exception (on a platform that supports C++ exceptions, when exceptions are enabled) if you attempt to extract more than one objective at a time.

Also see the functions `IloMaximize` and `IloMinimize` for “short cuts” to create objectives.

```plaintext
public void setSense(IloObjective::Sense)
```

**Inherited methods from IloExtractable**

- IloExtractable::end, IloExtractable::getEnv, IloExtractable::getId,
- IloExtractable::getImpl, IloExtractable::getName,
- IloExtractable::getObject, IloExtractable::setName,
- IloExtractable::setObject

**Inner Enumeration**

<table>
<thead>
<tr>
<th>IloObjective::Sense</th>
</tr>
</thead>
</table>

- IloObjective::Sense

An instance of this class is an objective in a model. An objective consists of its sense (indicating whether it is a minimization or maximization) and an expression. The expression may be a constant.

An objective belongs to the environment that the variables in its expression belong to. Generally, you will create an objective, add it to a model, and extract the model for an algorithm.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept Assert and NDEBUG.

**What Is Extracted**

All the variables (that is, instances of `IloNumVar` or one of its subclasses) in the objective (an instance of `IloObjective`) will be extracted when an algorithm such as `IloCplex`, documented in the ILOG CPLEX Reference Manual, extracts the objective.

**Multiple Objectives**

You may create more than one objective in a model, for example, by creating more than one group. However, certain algorithms, such as an instance of `IloCplex`, will throw an exception (on a platform that supports C++ exceptions, when exceptions are enabled) if you attempt to extract more than one objective at a time.

Also see the functions `IloMaximize` and `IloMinimize` for “short cuts” to create objectives.
Normalizing Linear Expressions: Reducing the Terms

Normalizing is sometimes known as reducing the terms of a linear expression.

Linear expressions consist of terms made up of constants and variables related by arithmetic operations; for example, \( x + 3y \) is a linear expression of two terms consisting of two variables. In some linear expressions, a given variable may appear in more than one term, for example, \( x + 3y + 2x \). Concert Technology has more than one way of dealing with linear expressions in this respect, and you control which way Concert Technology treats linear expressions from your application.

In one mode (the default mode), Concert Technology analyzes expressions that your application passes it and attempts to reduce them so that a given variable appears in only one term in the expression. You set this mode with the member function `setNormalizer`.

Certain constructors and member functions in this class check this setting in the model and behave accordingly: they attempt to reduce expressions. This mode may require more time during preliminary computation, but it avoids the possibility of an assertion failing for certain member functions of this class in case of duplicates.

In the other mode, Concert Technology assumes that no variable appears in more than one term in any of the linear expressions that your application passes to Concert Technology. We call this mode assume no duplicates. You set this mode with the member function `setNormalizer`.

Certain constructors and member functions in this class check this setting in the model and behave accordingly: they assume that no variable appears in more than one term in an expression. This mode may save time during computation, but it entails the risk that an expression may contain one or more variables, each of which appears in one or more terms. This situation will cause certain assert statements in Concert Technology to fail if you do not compile with the flag `-DNDEBUG`.

See Also

`IloMaximize`, `IloMinimize`, `IloModel`, `IloObjective::Sense`

Constructors

```
public IloObjective()
```

This constructor creates an empty handle. You must initialize it before you use it.

```
public IloObjective(ImplClass * impl)
```

This constructor creates a handle object from a pointer to an implementation object.

```
public IloObjective(const IloEnv env,
                    IloNum constant,
                    IloObjective::Sense sense,
                    const char * name)
```

This constructor creates an objective consisting of a constant and belonging to `env`. The sense of the objective (whether it is a minimization or maximization) is indicated by
sense; by default, it is a minimization. You may supply a name for the objective; by default, its name is the empty string. This constructor is useful when you want to create an empty objective and fill it later by column-wise modeling.

```java
public IloObjective(const IloEnv env,
                     const IloNumExprArg expr,
                     IloObjective::Sense sense,
                     const char * name)
```

This constructor creates an objective to add to a model from expr. After you create an objective from an expression with this constructor, you must use the member function add explicitly to add your objective to your model or to a group in order for the objective to be taken into account.

**Note:** *When it accepts an expression as a parameter, this constructor checks the setting of setNormalizer to determine whether to assume the expression has already been reduced or to reduce the expression before using it.*

**Methods**

```java
public IloNum getConstant()
```

This member function returns the constant term from the expression of the invoking objective.

```java
public IloNumExprArg getExpr()
```

This member function returns the expression of the invoking IloObjective object.

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```java
public IloObjective::Sense getSense()
```

This member function returns the sense of the invoking objective, indicating whether the objective is a minimization (Minimize) or a maximization (Maximize).

```java
public IloAddValueToObj operator()(IloNum value)
```

This casting operator uses a floating-point value to create an instance of IloAddNumVar or one of its subclasses and to add that value to that instance. See the concept Column-Wise Modeling for an explanation of how to use this operator in column-wise modeling.

```java
public IloAddValueToObj operator()()
```

This casting operator uses a floating-point value to create an instance of IloAddNumVar or one of its subclasses and to add that value to that instance. If no
argument is given, it assumes 1.0. See the concept Column-Wise Modeling for an explanation of how to use this operator in column-wise modeling.

public void setConstant(IloNum constant)

This member function sets constant as the constant term in the invoking objective, and it creates the appropriate instance of the undocumented class IloChange to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setConstant` notifies Concert Technology algorithms about this change of this invoking object.

public void setExpr(const IloNumExprArg)

This member function sets the expression of the invoking IloObjective object.

public void setLinearCoef(const IloNumVar var, IloNum value)

This member function sets value as the linear coefficient of the variable var in the invoking objective, and it creates the appropriate instance of the undocumented class IloChange to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setLinearCoef` notifies Concert Technology algorithms about this change of this invoking object.

If you attempt to use `setLinearCoef` on a nonlinear expression, it will throw an exception on platforms that support C++ exceptions when exceptions are enabled.

public void setLinearCoefs(const IloNumVarArray vars, const IloNumArray values)

For each of the variables in vars, this member function sets the corresponding value of values (whether integer or floating-point) as its linear coefficient in the invoking objective, and it creates the appropriate instance of the undocumented class IloChange to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setLinearCoefs` notifies Concert Technology algorithms about this change of this invoking object.
If you attempt to use `setLinearCoef` on a non-linear expression, Concert Technology will throw an exception on platforms that support C++ exceptions when exceptions are enabled.

```java
public void setSense(IloObjective:: Sense sense)
```

This member function sets `sense` to indicate whether the invoking objective is a maximization (Maximize) or minimization (Minimize), and it creates the appropriate instance of the undocumented class `IloChange` to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setSense` notifies Concert Technology algorithms about this change of this invoking object.
IloObjective::Sense

Category: Inner Enumeration

Definition File: ilconcert/ilolinear.h

Synopsis:

```cpp
Sense(
    Minimize,
    Maximize
);
```

Description:

An instance of the class `IloObjective` represents an objective in a model. This nested enumeration is limited in scope to that class, and its values indicate the sense of an objective—whether it is a minimization (Minimize) or a maximization (Maximize).

See Also: `IloObjective`

Fields:

- Minimize
- Maximize
**IloOr**

**Category**  
Class

**Inheritance Path**

```
IloOr
    ↓
IloNumExprArg
    ↓
IloIntExprArg
    ↓
IloConstraint
    ↓
IloExtractable
```

**Definition File**  
ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IloOr()</td>
</tr>
<tr>
<td></td>
<td>IloOr(ImplClass *)</td>
</tr>
<tr>
<td></td>
<td>IloOr(const IloEnv, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public void</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add(const IloConstraintArray)</td>
</tr>
<tr>
<td></td>
<td>add(const IloConstraint)</td>
</tr>
<tr>
<td>ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td></td>
<td>remove(const IloConstraintArray)</td>
</tr>
<tr>
<td></td>
<td>remove(const IloConstraint)</td>
</tr>
</tbody>
</table>

**Inherited methods from IloConstraint**

IloConstraint::getImpl
IloOr

Description

An instance of IloOr represents a disjunctive constraint. In other words, it defines a disjunctive-OR among any number of constraints. Since an instance of IloOr is a constraint itself, you can build up extensive disjunctions by adding constraints to an instance of IloOr by means of the member function add. You can also remove constraints from an instance of IloOr by means of the member function remove.

The elements of a disjunctive constraint must be in the same environment.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

Disjunctive Goals

If you would like to represent a disjunctive-OR as a goal (rather than a constraint), then you should consider the function IloOrGoal, documented in the ILOG Solver Reference Manual.

What Is Extracted

All the constraints (that is, instances of IloConstraint or one of its subclasses) that have been added to a disjunctive constraint (an instance of IloOr) and that have not
been removed from it will be extracted when an algorithm such as IloSolver (documented in the ILOG Solver Reference Manual) extracts the constraint.

Example

For example, you may write:

```c++
IloOr or(env);
or.add(constraint1);
or.add(constraint2);
or.add(constraint3);
```

Those lines are equivalent to:

```c++
IloOr or = constraint1 || constraint2 || constraint3;
```

See Also

IloAnd, IloConstraint, operator ||

Constructors

public IloOr() 
This constructor creates an empty handle. You must initialize it before you use it.

public IloOr(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloOr(const IloEnv env, 
const char * name)
This constructor creates a disjunctive constraint for use in env. The optional argument name is set to 0 by default.

Methods

public void add(const IloConstraintArray cons)
This member function makes all the elements in array elements of the invoking disjunctive constraint. In other words, it applies the invoking disjunctive constraint to all the elements of array.

Note: The member function add notifies Concert Technology algorithms about this change of this invoking object.

public void add(const IloConstraint con)
This member function makes \texttt{constraint} one of the elements of the invoking disjunctive constraint. In other words, it applies the invoking disjunctive constraint to \texttt{constraint}.

\textbf{Note:} The member function \texttt{add} notifies Concert Technology algorithms about this change of this invoking object.

\begin{verbatim}
public ImplClass * getImpl()
\end{verbatim}

This member function returns a pointer to the implementation object of the invoking handle.

\begin{verbatim}
pUBLIC void remove(const IloConstraintArray cons)
\end{verbatim}

This member function removes all the elements of \texttt{array} from the invoking disjunctive constraint so that the invoking disjunctive constraint no longer applies to any of those elements.

\textbf{Note:} The member function \texttt{remove} notifies Concert Technology algorithms about this change of this invoking object.

\begin{verbatim}
pUBLIC void remove(const IloConstraint con)
\end{verbatim}

This member function removes \texttt{constraint} from the invoking disjunctive constraint so that the invoking disjunctive constraint no longer applies to \texttt{constraint}.

\textbf{Note:} The member function \texttt{remove} notifies Concert Technology algorithms about this change of this invoking object.
IloPack

Category     Class
InheritancePath

Definition File     ilconcert/ilomodel.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloPack()</td>
</tr>
<tr>
<td>public IloPack(ImplClass *)</td>
</tr>
<tr>
<td>public IloPack(const IloEnv, const IloIntExprArray, const IloIntExprArray,</td>
</tr>
<tr>
<td>const IloIntArray, const char *)</td>
</tr>
<tr>
<td>public IloPack(const IloEnv, const IloIntExprArray, const IloIntExprArray,</td>
</tr>
<tr>
<td>const IloInt, const char *)</td>
</tr>
<tr>
<td>public IloPack(const IloEnv, const IloIntExprArray, const IloIntExprArray,</td>
</tr>
<tr>
<td>const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>getImpl()</td>
</tr>
</tbody>
</table>
Description

The IloPack constraint maintains the load of a set of containers or bins, given a set of weighted items and an assignment of items to containers.

Consider that we have \( n \) items and \( m \) containers. Each item \( i \) has an integer weight \( w[i] \) and a constrained integer variable \( p[i] \) associated with it, indicating in which container (numbered contiguously from 0) item \( i \) is to be placed. No item can be split up, and so an item can go in only one container. Associated with each container \( j \) is an integer variable \( l[j] \) representing the load in that container; that is, the sum of the weights of the items which have been assigned to that container. A capacity can be set for each container placing an upper bound on this load variable. The constraint also ensures that the total sum of the loads of the containers is equal to the sum of the weights of the items being placed.

Constructors

public IloPack()

This constructor creates an empty handle. You must initialize it before you use it.

public IloPack(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

```java
public IloPack(const IloEnv env,
               const IloIntExprArray load,
               const IloIntExprArray where,
               const IloIntArray weight,
               const char * name)
```

This constructor creates a constraint that maintains the container loads subject to the assignments of weighted items to containers. `load` is the array of loads, one for each container. `where` indicates, for each item, which container it should be placed in, counting from 0. `weight` is an array of non-negative weights, one for each item. `where` and `weight` must be the same size, otherwise an instance of `IloException` is thrown. `name`, if specified becomes the name of the constraint.

```java
public IloPack(const IloEnv env,
               const IloIntExprArray load,
               const IloIntExprArray where,
               const IloInt weight,
               const char * name)
```

This constructor creates a constraint that maintains the container loads subject to the assignments of weighted items to containers. `load` is the array of loads, one for each container. `where` indicates, for each item, which container it should be placed in, counting from 0. `weight` is a non-negative weight; all items have this weight. `name`, if specified becomes the name of the constraint.

```java
public IloPack(const IloEnv env,
               const IloIntExprArray load,
               const IloIntExprArray where,
               const char * name)
```

This constructor creates a constraint that maintains the container loads subject to the assignments of weighted items to containers. `load` is the array of loads, one for each container. `where` indicates, for each item, which container it should be placed in, counting from 0. All items have unit weight. `name`, if specified becomes the name of the constraint.

**Methods**

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.
### IloPartition

**Category**  
Global Function

**Definition File**  
ilconcert/iloanyset.h

**Synopsis**

```c
public IloConstraint IloPartition(const IloEnv  env,
                                  const IloAnySetVarArray  vars)
public IloConstraint IloPartition(const IloEnv  env,
                                  const IloAnySetVarArray  vars,
                                  const IloAnyArray  vals)
public IloConstraint IloPartition(const IloEnv ,
                                  const IloIntSetVarArray  vars)
public IloConstraint IloPartition(const IloEnv ,
                                  const IloIntSetVarArray  vars,
                                  const IloIntArray  vals)
```

**Description**

This function creates and returns a constraint. When that constraint is posted, it insures that each value of the array `vals` will be required by exactly one set variable of the array `vars`.

If the argument `vals` is not mentioned, the array `vals` is formed by the union of the values involved in the set variables of `vars`.

In this context, a constraint will be posted after it has been added to a model and extracted by a solver (for example, an instance of `IloSolver` documented in the *ILOG Solver Reference Manual*).
IloPathLength

Category    Class
InheritancePath

Definition File    ilconcert/ilomodel.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public    IloPathLength()</td>
</tr>
<tr>
<td>public    IloPathLength(ImplClass *)</td>
</tr>
<tr>
<td>public    IloPathLength(const IloEnv, const IloIntVarArray, const IloNumVarArray, IloPathTransitFunction, IloInt, const char *)</td>
</tr>
<tr>
<td>public    IloPathLength(const IloEnv, const IloIntVarArray, const IloIntVarArray, IloPathTransitFunction, IloInt, const char *)</td>
</tr>
<tr>
<td>public    IloPathLength(const IloEnv, const IloIntVarArray, const IloNumVarArray, IloPathTransitI *, IloInt, const char *)</td>
</tr>
</tbody>
</table>
### Description

An instance of this class is a path constraint in Concert Technology.
In order for the constraint to take effect, you must add it to a model with the template
IloAdd or the member function IloModel::add and extract the model for an
algorithm with the member function IloAlgorithm::extract.

What IloPathLength Does Not Do

The path-length constraint does not determine whether there is a path between nodes in
a graph; rather, it constrains accumulations (such as flow) along a path. The filtering
algorithm associated with this constraint works on the accumulation variables in the
array lengths.

If you are looking for a Hamiltonian path, for example, (that is, one in which each node
is visited exactly once), consider using instead the constraint IloAllDiff on the
variables in the array next.

What IloPathLength Does

If we are given
◆ a set of n nodes, known as N,
◆ a maximum number of paths among those nodes, maxNbPaths,
◆ a set of maxNbPaths nodes, known as S, for starting nodes,
◆ a set of maxNbPaths nodes, known as E, for ending nodes,
then a path constraint insures that there exist at most maxNbPaths paths starting from
a node in S, visiting nodes in N, and ending at a node in E. Furthermore, each node will
be visited only once, has only one predecessor and only one successor, and each node
belongs to a path that starts from a node in S and ends at a node in E.

In particular, in an instance of IloPath, in the arrays next and cumul,
◆ the indices in [0, n-1] correspond to the nodes of N,
◆ the indices in [n, n+maxNbPaths-1] correspond to the nodes of E,
◆ and the indices in [n+maxNbPaths, n+2*maxNbPaths-1] correspond to the
nodes of S.

In other words, the size of next and cumul is n+2*maxNbPaths.

next[i] is the node following node i on the current path. cumul[i] is the
accumulated cost from the beginning of the path to node i. The argument transit
indicates the transition function.

When this constraint is satisfied, it insures that for all indices i in the range [0, n-1]
or in [n+maxNbPaths, n+2*maxNbPaths-1], if next[i]==j and j is in
[0, n+maxNbPaths-1], then cumul[i] + transit.transit(i,j) <=
cumul[j].
When $i$ is in the range $[n, n+\text{maxNbPaths}-1]$, $\text{next}[i]$ has no meaning because the nodes in $E$ do not have successors, of course. In this case, the constraint deals with them by setting $\text{next}[i]$ to $i+\text{maxNbPaths}$ (that is, nodes of $S$).

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also

IloAllDiff, IloConstraint, IloPathTransitFunction, IloPathTransitI

Constructors

public IloPathLength()

This constructor creates an empty handle. You must initialize it before you use it.

public IloPathLength(ImplClass * impl)

This constructor creates a handle object from a pointer to an implementation object.

public IloPathLength(const IloEnv env,
    const IloIntVarArray next,
    const IloNumVarArray cumul,
    IloPathTransitFunction transit,
    IloInt nbPaths,
    const char * name)

This constructor creates a path constraint in an environment.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

If the array $\text{next}$ or $\text{cumul}$ is not an appropriate length, then on platforms that support C++ exceptions when exceptions are enabled, this constructor will throw the exception InvalidArraysException.

public IloPathLength(const IloEnv env,
    const IloIntVarArray next,
    const IloIntVarArray cumul,
    IloPathTransitFunction transit,
    IloInt nbPaths,
    const char * name)

This constructor creates a path constraint in an environment.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

If the array $\text{next}$ or $\text{cumul}$ is not an appropriate length, then on platforms that support C++ exceptions when exceptions are enabled, this constructor will throw the exception InvalidArraysException.
public IloPathLength(const IloEnv env,
    const IloIntVarArray next,
    const IloNumVarArray cumul,
    IloPathTransitI * pathTransit,
    IloInt nbPaths,
    const char * name)

This constructor creates a path constraint in an environment.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

If the array next or cumul is not an appropriate length, then on platforms that support C++ exceptions when exceptions are enabled, this constructor will throw the exception InvalidArraysException.

public IloPathLength(const IloEnv env,
    const IloIntVarArray next,
    const IloIntVarArray cumul,
    IloPathTransitI * pathTransit,
    IloInt nbPaths,
    const char * name)

This constructor creates a path constraint in an environment.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.

If the array next or cumul is not an appropriate length, then on platforms that support C++ exceptions when exceptions are enabled, this constructor will throw the exception InvalidArraysException.

Methods

public ImplClass * getImpl()

This member function returns a pointer to the implementation object of the invoking handle.
IloPathTransitFunction

Category: Type Definition
Definition File: ilconcert/ilomodel.h
Synopsis: IloNum(* IloPathTransitFunction)(IloInt i, IloInt j)
Description: This C++ type definition represents a pointer to a function that takes two arguments and returns a floating-point number. The two arguments are the indices of nodes. The function should return a transit cost for connecting the two nodes. This transit cost can be the distance between the nodes or the cost of a path visiting either of the nodes. This kind of function is known as a transit function.
See Also: IloPathLength, IloPathTransit
IloPathTransitI

Category
Class

InheritancePath

IloPathTransitI

Definition File
ilconcert/ilomodel.h

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>IloPathTransitI()</th>
</tr>
</thead>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>makeClone(IloEnvI *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>public virtual</td>
<td>IloNum transit(IloInt, IloInt)</td>
</tr>
</tbody>
</table>

**Description**

You can define the transit function in a path constraint (the cost for linking two nodes together).

This class is the implementation class for IloPathTransit, the class of object that defines a transit function for the path constraint. The virtual member function transit in IloPathTransitI returns the transition cost for connecting two nodes together.

To express new transit functions, you can define a subclass of IloPathTransitI. If this transition can be expressed by an evaluation function, then you can use the predefined IloPathFunction for that purpose.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

**See Also**
IloPathLength, IloPathTransitFunction

**Constructors**

public IloPathTransitI()
This constructor creates an implementation of a transit function.

**Methods**

```cpp
public virtual IloPathTransitI * makeClone(IloEnvI *)
```

This virtual member function returns a copy of the invoking object.

```cpp
public virtual IloNum transit(IloInt i, IloInt j)
```

This virtual member function returns the transition cost from node \( i \) to node \( j \). Its default implementation returns 0 (zero) as the value of every transition.
**IloPi**

**Category**  
Global Variable

**Definition File**  
ilconcert/ilosys.h

Concert Technology predefines conventional trigonometric constants to conform to IEEE 754 standards for quarter pi, half pi, pi, three-halves pi, and two pi.

```c
extern const IloNum IloPi;       // = 3.14159265358979323846
```
IloPiecewiseLinear

Category: Global Function

Definition File: ilconcert/iloexpression.h

Synopsis:

public IloNumExprArg IloPiecewiseLinear(const IloNumExprArg node,
 const IloNumArray point,
 const IloNumArray slope,
 IloNum a,
 IloNum fa)

public IloNumExprArg IloPiecewiseLinear(const IloNumExprArg node,
 IloNum firstSlope,
 const IloNumArray point,
 const IloNumArray value,
 IloNum lastSlope)

public IloNumExprArg IloPiecewiseLinear(const IloNumExprArg node,
 const IloNumArray point,
 const IloNumArray value)

Description:

The function IloPiecewiseLinear creates an expression node to represent a continuous or discontinuous piecewise linear function $f$ of the variable $x$. The array point contains the $n$ breakpoints of the function such that $\text{point}[i-1] \leq \text{point}[i]$ for $i = 1, \ldots, n-1$. The array slope contains the $n+1$ slopes of the $n+1$ segments of the function. The values $a$ and $fa$ must be coordinates of a point such that $fa = f(a)$. When $\text{point}[i-1] = \text{point}[i]$, there is a step at the x-coordinate $\text{point}[i-1]$ and its height is $\text{slope}[i-1]$.

Example:

The expression

\[
\text{IloPiecewiseLinear}(x, \text{IloNumArray}(\text{env}, 2, 10., 20.),
\text{IloNumArray}(\text{env}, 3, 0.3, 1., 2.),
0, 0);
\]

defines a piecewise linear function $f$ having two breakpoints at $x = 10$ and $x = 20$, and three segments; their slopes are $0.3$, $1$, and $2$. The first segment has infinite length and ends at the point $(x = 10, f(x) = 3)$ since $f(0) = 0$. The second segment starts at the point $(x = 10, f(x) = 3)$ and ends at the point $(x = 20, f(x) = 13)$, where the third segment starts.
IloPower

Category                  Global Function
Definition File           ilconcert/iloexpression.h
Synopsis                   
public IloNumExprArg IloPower(const IloNumExprArg x,  
                              const IloNumExprArg y)
public IloNumExprArg IloPower(const IloNumExprArg x,  
                              IloNum y)
public IloNumExprArg IloPower(IloNum x,  
                              const IloNumExprArg y)
Description                Concert Technology offers predefined functions that return an expression from an  
                            algebraic function on expressions. These predefined functions also return a numeric  
                            value from an algebraic function on numeric values as well.  
                            IloPower returns the result of raising its base argument to the power of its  
                            exponent argument, that is, base^exponent. If base is a floating-point value or  
                            variable, then exponent must be greater than or equal to 0 (zero).
IloQuarterPi

Category: Global Variable
Definition File: ilconcert/ilosys.h

Concert Technology predefines conventional trigonometric constants to conform to IEEE 754 standards for quarter pi, half pi, pi, three-halves pi, and two pi.

extern const IloNum IloQuarterPi; // = 0.78539816339744830962
**IloRandom**

**Category**  
Class

**Inheritance Path**

**Definition File**  
ilconcert/ilorandom.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloRandom()</td>
<td></td>
</tr>
<tr>
<td>IloRandom(const IloEnv, IloInt)</td>
<td></td>
</tr>
<tr>
<td>IloRandom(IloRandomI *)</td>
<td></td>
</tr>
<tr>
<td>IloRandom(const IloRandom &amp;)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end()</td>
<td></td>
</tr>
<tr>
<td>getEnv()</td>
<td></td>
</tr>
<tr>
<td>getFloat()</td>
<td></td>
</tr>
<tr>
<td>getImpl()</td>
<td></td>
</tr>
<tr>
<td>getInt(IloInt)</td>
<td></td>
</tr>
<tr>
<td>getName()</td>
<td></td>
</tr>
<tr>
<td>getobject()</td>
<td></td>
</tr>
<tr>
<td>reSeed(IloInt)</td>
<td></td>
</tr>
<tr>
<td>setName(const char *)</td>
<td></td>
</tr>
<tr>
<td>setObject(IloAny)</td>
<td></td>
</tr>
</tbody>
</table>

**Description**  
This handle class produces streams of pseudo-random numbers. You can use objects of this class to create a search with a random element. You can create any number of instances of this class; these instances insure reproducible results in multithreaded applications, where the use of a single source for random numbers creates problems.

**See Also**  
Constructors

public IloRandom()

This constructor creates a random number generator; it is initially an empty handle. You must assign this handle before you use its member functions.

public IloRandom(const IloEnv env,
               IloInt seed)

This constructor creates an object that generates random numbers. You can seed the generator by supplying a value for the integer argument seed.

public IloRandom(IloRandomI * impl)

This constructor creates a handle object (an instance of the class IloRandom) from a pointer to an implementation object (an instance of the class IloRandomI).

public IloRandom(const IloRandom & rand)

This constructor creates a handle object from a reference to a random number generator. After execution, both the newly constructed handle and rand point to the same implementation object.

Methods

public void end()

This member function releases all memory used by the random number generator. After a call to this member function, you should not use the generator again.

public IloEnv getEnv()

This member function returns the environment associated with the implementation class of the invoking generator.

public IloNum getFloat()

This member function returns a floating-point number drawn uniformly from the interval $[0..1)$.

public IloRandomI * getImpl()

This member function returns the implementation object of the invoking handle.

public IloInt getInt(IloInt n)

This member function returns an integer drawn uniformly from the interval $[0..n)$.

public const char * getName()

This member function returns a character string indicating the name of the invoking object (if there is one).

public IloAny getObject()

This member function returns the object associated with the invoking object (if there is one). Normally, an associated object contains user data pertinent to the invoking object.

public void reSeed(IloInt seed)
This member function re-seeds the random number generator with seed.

```java
public void setName(const char * name)
```

This member function assigns `name` to the invoking object.

```java
public void setObject(IloAny object)
```

This member function associates `obj` with the invoking object. The member function `getObject` accesses this associated object afterwards. Normally, `obj` contains user data pertinent to the invoking object.
IloRange

Category Class

InheritancePath

Definition File ilconcert/ilolinear.h

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloRange()</td>
</tr>
<tr>
<td>public</td>
<td>IloRange(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloRange(const IloEnv, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloRange(const IloEnv, IloNum, const IloNumExprArg, IloNum, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloRange(const IloEnv, const IloNumExprArg, IloNum, const char *)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloNumExprArg</td>
<td>getExpr()</td>
</tr>
<tr>
<td>ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>IloNum</td>
<td>getLB()</td>
</tr>
</tbody>
</table>
**Description**

An instance of this class is a range in a model, that is, a constraint of the form:

- `public IloNum getUB();`
- `public IloAddValueToRange operator()(IloNum num);`
- `public void setBounds(IloNum num, IloNum num);`
- `public void setExpr(const IloNumExprArg arg);`
- `public void setLB(IloNum num);`
- `public void setLinearCoef(const IloNumVar var, IloNum num);`
- `public void setLinearCoefs(const IloNumVarArray vars, const IloNumArray coeffs);`
- `public void setUB(IloNum num);`

**Inherited methods from IloConstraint**

- `IloConstraint::getImpl`
lowerBound <= expression <= upperBound

You can create a range from the constructors in this class or from the arithmetic operators on numeric variables (instances of IloNumVar and its subclasses) or on expressions (instances of IloExpr and its subclasses):

- operator <=
- operator >=
- operator ==

After you create a constraint, such as an instance of IloRange, you must explicitly add it to the model in order for it to be taken into account. To do so, use the member function IloModel::add to add the range to a model and the member function IloAlgorithm::extract to extract the model for an algorithm.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

What Is Extracted

All the variables (that is, instances of IloNumVar or one of its subclasses) in the range (an instance of IloRange) will be extracted when an algorithm such as IloCplex, documented in the ILOG CPLEX Reference Manual, extracts the range.

Normalizing Linear Expressions: Reducing the Terms

Normalizing is sometimes known as reducing the terms of a linear expression.

Linear expressions consist of terms made up of constants and variables related by arithmetic operations; for example, x + 3y. In some linear expressions, a given variable may appear in more than one term, for example, x + 3y +2x. Concert Technology has more than one way of dealing with linear expressions in this respect, and you control which way Concert Technology treats linear expressions from your application.

In one mode (the default mode), Concert Technology analyzes linear expressions that your application passes it, and attempts to reduce them so that a given variable appears in only one term in the expression. You set this mode with the member function setNormalizer(IloTrue).

Certain constructors and member functions in this class check this setting in the model and behave accordingly: they attempt to reduce expressions. This mode may require more time during preliminary computation, but it avoids the possibility of an assertion in some of the member functions of this class failing in the case of duplicates.

In the other mode, Concert Technology assumes that no variable appears in more than one term in any of the linear expressions that your application passes to Concert
Technology. We call this mode assume normalized linear expressions. You set this mode with the member function \texttt{setNormalizer(IloFalse)}.

Certain constructors and member functions in this class check this setting in the model and behave accordingly: they assume that no variable appears in more than one term in an expression. This mode may save time during computation, but it entails the risk that an expression may contain one or more variables, each of which appears in one or more terms. This situation will cause certain assert statements in Concert Technology to fail if you do not compile with the flag \texttt{-DNDEBUG}.

\textbf{See Also} \hfill IloConstraint, IloRangeArray

\textbf{Constructors}

\begin{verbatim}
public IloRange()
This constructor creates an empty handle. You must initialize it before you use it.

public IloRange(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloRange(const IloEnv env,
                IloNum lb,
                IloNum ub,
                const char * name)
This constructor creates an empty range constraint. Before you use this constraint, you must fill the range. The optional argument \texttt{name} is set to 0 by default.

After you create a range constraint, you must explicitly add it to a model in order for it to be taken into account. To do so, use the member function \texttt{IloModel::add}.

public IloRange(const IloEnv env,
                IloNum lhs,
                const IloNumExprArg expr,
                IloNum rhs,
                const char * name)
This constructor creates a range constraint from an expression (an instance of the class IloNumExprArg) and its upper bound (\texttt{rhs}). The default bound for \texttt{rhs} is the symbolic constant IloInfinity. The optional argument \texttt{name} is set to 0 by default.

\end{verbatim}

\textbf{Note}: When it accepts an expression as a parameter, this constructor checks the setting of \texttt{setNormalizer} to determine whether to assume the expression has already been reduced or to reduce the expression before using it.

\begin{verbatim}
public IloRange(const IloEnv env,
                const IloNumExprArg expr,

\end{verbatim}
This constructor creates a range constraint from an expression (an instance of the class `IloNumExprArg`) and its upper bound (`rhs`). Its lower bound (`lhs`) will be `IloInfinity`. The default bound for `rhs` is `IloInfinity`. The optional argument `name` is set to 0 by default.

**Methods**

- **public IloNumExprArg getExpr()**
  This member function returns the expression of the invoking `IloRange` object.

- **public ImplClass * getImpl()**
  This member function returns a pointer to the implementation object of the invoking handle.

- **public IloNum getLB()**
  This member function returns the lower bound of the invoking range.

- **public IloNum getUB()**
  This member function returns the upper bound of the invoking range.

- **public IloAddValueToRange operator()(IloNum value)**
  This operator creates the objects needed internally to represent a range in column-wise modeling. See the concept Column-Wise Modeling for an explanation of how to use this operator in column-wise modeling.

- **public void setBounds(IloNum lb, IloNum ub)**
  This member function sets `lb` as the lower bound and `ub` as the upper bound of the invoking range, and it creates the appropriate instance of the undocumented class `IloChange` to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setBounds` notifies Concert Technology algorithms about this change of this invoking object.
public void setExpr(const IloNumExprArg expr)

This member function sets \( expr \) as the invoking range, and it creates the appropriate instance of the undocumented class \( IloChange \) to notify algorithms about this change of an extractable object in the model.

public void setLB(IloNum lb)

This member function sets \( lb \) as the lower bound of the invoking range, and it creates the appropriate instance of the undocumented class \( IloChange \) to notify algorithms about this change of an extractable object in the model.

Note: The member function \( \text{setLB} \) notifies Concert Technology algorithms about this change of this invoking object.

public void setLinearCoef(const IloNumVar var, IloNum value)

This member function sets \( value \) as the linear coefficient of the variable \( var \) in the invoking range, and it creates the appropriate instance of the undocumented class \( IloChange \) to notify algorithms about this change of an extractable object in the model.

Note: The member function \( \text{setLinearCoef} \) notifies Concert Technology algorithms about this change of this invoking object.

If you attempt to use \( \text{setLinearCoef} \) on a non linear expression, it will throw an exception on platforms that support C++ exceptions when exceptions are enabled.

public void setLinearCoefs(const IloNumVarArray vars, const IloNumArray values)

For each of the variables in \( vars \), this member function sets the corresponding value of values (whether integer or floating-point) as its linear coefficient in the invoking range, and it creates the appropriate instance of the undocumented class \( IloChange \) to notify algorithms about this change of an extractable object in the model.

Note: The member function \( \text{setLinearCoefs} \) notifies Concert Technology algorithms about this change of this invoking object.
If you attempt to use `setLinearCoef` on a non linear expression, it will throw an exception on platforms that support C++ exceptions when exceptions are enabled.

```java
public void setUB(IloNum ub)
```

This member function sets `ub` as the upper bound of the invoking range, and it creates the appropriate instance of the undocumented class `IloChange` to notify algorithms about this change of an extractable object in the model.

**Note:** The member function `setUB` notifies Concert Technology algorithms about this change of this invoking object.
IloRangeArray

Category: Class

Inheritance Path:

Definition File: ilconcert/ilolinear.h

Constructor Summary:

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloRangeArray(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, IloInt, IloNum, IloNum)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, const IloNumArray, const IloNumExprArray, const IloNumArray)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, const IloNumArray, const IloNumExprArray, const IloNumArray)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, const IloIntArray, const IloNumExprArray, const IloIntArray)</td>
</tr>
<tr>
<td>public IloRangeArray(const IloEnv, const IloIntArray, const IloNumExprArray, const IloIntArray)</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td><code>add(IloInt, const IloRange)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>add(const IloRange)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>add(const IloRangeArray)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloNumColumn operator[](const IloNumArray)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloNumColumn operator[](const IloIntArray)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloRange operator[](IloInt)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloRange &amp; operator[](IloInt)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>setBounds(const IloIntArray, const IloIntArray)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>setBounds(const IloNumArray, const IloIntArray)</code></td>
</tr>
</tbody>
</table>
Description
For each basic type, Concert Technology defines a corresponding array class. 
*IloRangeArray* is the array class of ranges for a model.

Instances of *IloRangeArray* are extensible. That is, you can add more elements to 
such an array. References to an array change whenever an element is added to or 
removed from the array.

Most member functions in this class contain *assert* statements. For an explanation of 
the macro *NDEBUG* (a way to turn on or turn off these *assert* statements), see the 
concept *Assert and NDEBUG*.

For information on arrays, see the concept *Arrays*.

See Also
*IloRange*, *operator*>>*, *operator*

Constructors

Public *IloRangeArray*(*IloDefaultArrayI* *i*)

This default constructor creates an empty range array. You cannot create instances of the 
undocumented class *IloDefaultArrayI*. As a parameter in this default constructor, 
it allows you to pass 0 (zero) as a value to an optional parameter in functions and 
member functions that accept an array as a parameter.

Public *IloRangeArray*(*const IloEnv* env, 
                 *IloInt* n)

This constructor creates an array of *n* elements, each of which is an empty handle.

Public *IloRangeArray*(*const IloEnv* env, 
                 *IloInt* n, 
                 *IloNum* lb, 
                 *IloNum* ub)
This constructor creates an array of \( n \) elements, each with the lower bound \( lb \) and the upper bound \( ub \).

```java
public IloRangeArray(const IloEnv env,
const IloNumArray lbs,
const IloNumExprArray rows,
const IloNumArray ubs)
```

This constructor creates an array of ranges from \( rows \), an array of expressions. It uses the corresponding elements of the arrays \( lbs \) and \( ubs \) to set the lower and upper bounds of elements in the new array. The length of \( rows \) must equal the length of \( lbs \) and \( ubs \).

```java
public IloRangeArray(const IloEnv env,
IloNum lb,
const IloNumExprArray rows,
const IloNumArray ubs)
```

This constructor creates an array of ranges from \( rows \), an array of expressions. The lower bound of every element in the new array will be \( lb \). The upper bound of each element of the new array will be the corresponding element of the array \( ubs \). The length of \( rows \) must equal the length of \( ubs \).

```java
public IloRangeArray(const IloEnv env,
const IloNumArray lbs,
const IloNumExprArray rows,
IloNum ub)
```

This constructor creates an array of ranges from \( rows \), an array of expressions. The upper bound of every element in the new array will be \( ub \). The lower bound of each element of the new array will be the corresponding element of the array \( lbs \). The length of \( rows \) must equal the length of \( lbs \).

```java
public IloRangeArray(const IloEnv env,
IloNum lb,
const IloNumExprArray rows,
IloNum ub)
```

This constructor creates an array of ranges from \( rows \), an array of expressions. The lower bound of every element in the new array will be \( lb \). The upper bound of every element in the new array will be \( ub \).

```java
public IloRangeArray(const IloEnv env,
const IloIntArray lbs,
const IloNumExprArray rows,
const IloIntArray ubs)
```

This constructor creates an array of ranges from \( rows \), an array of expressions. It uses the corresponding elements of the arrays \( lbs \) and \( ubs \) to set the lower and upper bounds of elements in the new array. The length of \( rows \) must equal the length of \( lbs \) and \( ubs \).
public IloRangeArray\(\text{const IloEnv env,}\)
\(\text{IloNum lb,}\)
\(\text{const IloNumExprArray rows,}\)
\(\text{const IloIntArray ubs}\)

This constructor creates an array of ranges from \(\text{rows, an array of expressions. The lower bound of every element in the new array will be lb. The upper bound of each element of the new array will be the corresponding element of the array ubs. The length of rows must equal the length of ubs.}\)

public IloRangeArray\(\text{const IloEnv env,}\)
\(\text{const IloIntArray lbs,}\)
\(\text{const IloNumExprArray rows,}\)
\(\text{IloNum  ub}\)

This constructor creates an array of ranges from \(\text{rows, an array of expressions. The upper bound of every element in the new array will be ub. The lower bound of each element of the new array will be the corresponding element of the array lbs. The length of rows must equal the length of lbs.}\)

public IloRangeArray\(\text{const IloEnv env,}\)
\(\text{const IloNumArray lbs,}\)
\(\text{const IloNumArray ubs}\)

This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the arrays lbs (or ubs). The number of elements in lbs must be equal to the number of elements in ubs. The lower bound of each element in the new array will be equal to the corresponding element in the array lbs. The upper bound of each element in the new array will be equal to the corresponding element in the array ubs.

public IloRangeArray\(\text{const IloEnv env,}\)
\(\text{const IloIntArray lbs,}\)
\(\text{const IloIntArray ubs}\)

This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the arrays lbs (or ubs). The number of elements in lbs must be equal to the number of elements in ubs. The lower bound of each element in the new array will be equal to the corresponding element in the array lbs. The upper bound of each element in the new array will be equal to the corresponding element in the array ubs.

public IloRangeArray\(\text{const IloEnv env,}\)
\(\text{IloNum lb,}\)
\(\text{const IloNumArray ubs}\)

This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the array ubs. The lower bound of every element in the new array will be equal to lb. The upper bound of each element in the new array will be equal to the corresponding element in the array ubs.
This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the array `ubs`. The lower bound of every element in the new array will be equal to `lb`. The upper bound of each element in the new array will be equal to the corresponding element in the array `ubs`.

```java
public IloRangeArray(const IloEnv env,
                     IloNum lb,
                     const IloIntArray  ubs)
```

This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the array `ubs`. The lower bound of every element in the new array will be equal to `lb`. The upper bound of each element in the new array will be equal to the corresponding element in the array `ubs`.

```java
public IloRangeArray(const IloEnv env,
                     const IloIntArray  lbs,
                     IloNum  ub)
```

This constructor creates an array of ranges. The number of elements in the new array will be equal to the number of elements in the array `ubs`. The lower bound of every element in the new array will be equal to `lb`. The upper bound of each element in the new array will be equal to the corresponding element in the array `ubs`.

```java
public void add(IloInt more,
                const IloRange  range)
```

This member function appends `range` to the invoking array multiple times. The parameter `more` indicates how many times.

```java
public void add(const IloRange  range)
```

This member function appends `range` to the invoking array.

```java
public void add(const IloRangeArray array)
```

This member function appends the elements in `array` to the invoking array.

```java
public IloNumColumn operator() (const IloNumArray  vals)
```

This operator constructs ranges in column representation. That is, it creates an instance of `IloNumColumn` that will add a newly created variable to all the ranged constraints in the invoking object, each as a linear term with the corresponding value specified in the array values.

```java
public IloNumColumn operator() (const IloIntArray  vals)
```

This operator constructs ranges in column representation. That is, it creates an instance of `IloNumColumn` that will add a newly created variable to all the ranged constraints in the invoking object, each as a linear term with the corresponding value specified in the array values.
public IloRange operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i. Concert Technology uses the const operator

IloRange operator[](IloInt i) const;

on const arrays.

public IloRange & operator[](IloInt i)

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.

public void setBounds(const IloIntArray lbs,
const IloIntArray ubs)

This member function does not change the array itself; instead, it changes the bounds of all the ranged constraints that are elements of the invoking array. At the same time, it also creates an instance of the undocumented class IloChange to notify Concert Technology algorithms about this change in an extractable object of the model. The elements of the arrays lbs and ubs may be integer or floating-point values. The size of the invoking array must be equal to the size of lbs and the size of ubs.

Note: The member function setBounds notifies Concert Technology algorithms about this change of bounds for all the elements in this invoking array.

public void setBounds(const IloNumArray lbs,
const IloNumArray ubs)

This member function does not change the array itself; instead, it changes the bounds of all the ranged constraints that are elements of the invoking array. At the same time, it also creates an instance of the undocumented class IloChange to notify Concert Technology algorithms about this change in an extractable object of the model. The elements of the arrays lbs and ubs may be integer or floating-point values. The size of the invoking array must be equal to the size of lbs and the size of ubs.

Note: The member function setBounds notifies Concert Technology algorithms about this change of bounds for all the elements in this invoking array.
**IloRound**

**Category**  
Global Function

**Definition File**  
ilconcert/iloenv.h

**Synopsis**  
public IloNum IloRound(IloNum val)

**Description**  
This function computes the nearest integer value to val. Halfway cases are rounded to the larger in magnitude.

**Examples:**

- IloRound(IloInfinity) is IloInfinity.
- IloRound(-IloInfinity) is -IloInfinity.
- IloRound(0) is 0.
- IloRound(0.4) is 0.
- IloRound(-0.4) is 0.
- IloRound(0.5) is 1.
- IloRound(-0.5) is -1.
- IloRound(0.6) is 1.
- IloRound(-0.6) is -1.
- IloRound(1e300) is 1e300.
- IloRound(1.0000001e6) is 1e6.
- IloRound(1.0000005e6) is 1.000001e6.
IloSOS1

Category      Class

InheritancePath

Definition File  ilconcert/ilolinear.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public  IloSOS1()</td>
</tr>
<tr>
<td>public  IloSOS1(ImplClass *)</td>
</tr>
<tr>
<td>public  IloSOS1(const IloEnv, const char *)</td>
</tr>
<tr>
<td>public  IloSOS1(const IloEnv, const IloNumVarArray, const char *)</td>
</tr>
<tr>
<td>public  IloSOS1(const IloEnv, const IloNumVarArray, const IloNumArray, const char *)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass * getImpl()</td>
</tr>
<tr>
<td>public void getNumVars(IloNumVarArray)</td>
</tr>
<tr>
<td>public void getValues(IloNumArray)</td>
</tr>
</tbody>
</table>
Description

This handle class represents special ordered sets of type 1 (SOS1). A special ordered set of type 1 specifies a set of variables, and only one among them may take a non-zero value. You may assign a weight to each variable in an SOS1. This weight indicates an order among the variables. If you do not specify any weights to indicate order among the variables, then Concert Technology considers the order in which you gave the variables to the constructor of this set and the order in which you added variables later.

When you extract a model (an instance of IloModel) for an instance of IloCplex (documented in the ILOG CPLEX Reference Manual), it will use the order for branching on variables.

For more details about SOS1, see the ILOG CPLEX Reference and User’s Manuals.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

See Also

IloSOS1Array, IloSOS2
Constructors

public IloSOS1()
This constructor creates an empty handle. You must initialize it before you use it.

public IloSOS1(ImplClass * impl)
This constructor creates a handle object from a pointer to an implementation object.

public IloSOS1(const IloEnv env, const char * name)
This constructor creates a special ordered set of type 1 (SOS1). You must add the variables to this set for them to be taken into account.

public IloSOS1(const IloEnv env, const IloNumVarArray vars, const char * name)
This constructor creates a special ordered set of type 1 (SOS1). The set includes each of the variables specified in the array vars.

public IloSOS1(const IloEnv env, const IloNumVarArray vars, const IloNumArray vals, const char * name)
This constructor creates a special ordered set of type 1 (SOS1). The set includes the variables specified in the array vars. The corresponding value in vals indicates the weight of each variable in vars.

Methods

public ImplClass * getImpl()
This member function returns a pointer to the implementation object of the invoking handle.

public void getNumVars(IloNumVarArray variables)
This member function accesses the variables in a special ordered set of type 1 (SOS1) and puts those variables into its parameter variables.

public void getValues(IloNumArray values)
This member function accesses the weights of the variables in a special ordered set of type 1 (SOS1) and puts those weights into its parameter values.
IloSOS1Array

Category: Class

InheritancePath:

\[ \text{IloSOS1Array} \rightarrow \text{IloExtractableArray} \rightarrow \text{IloConstraintArray} \]

Definition File: ilconcert/ilolinear.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloSOS1Array(IloDefaultArrayI *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloSOS1Array(const IloEnv, IloInt)</td>
</tr>
</tbody>
</table>

### Method Summary

| Public void | add(IloInt, const IloSOS1 &) |
| Public void | add(const IloSOS1 &) |
| Public void | add(const IloSOS1Array &) |
| Public IloSOS1 | operator[](IloInt) |
| Public IloSOS1 & | operator[](IloInt) |

### Inherited methods from IloConstraintArray

add, add, add, operator[], operator[]
IloSOS1Array

Description

For each basic type, Concert Technology defines a corresponding array class. IloSOS1Array is the array class of special ordered sets of type 1 (SOS1) for a model.

Instances of IloSOS1Array are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

For information on arrays, see the concept Arrays

See Also

IloSOS1, operator>>, operator

Constructors

public IloSOS1Array(IloDefaultArrayI * i)

This default constructor creates an empty array. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public IloSOS1Array(const IloEnv env, IloInt n)

This constructor creates an array of n empty elements in the environment env.

Methods

public void add(IloInt more, const IloSOS1 & x)

This member function appends x to the invoking array multiple times. The parameter more indicates how many times.

public void add(const IloSOS1 & x)

This member function appends x to the invoking array.

public void add(const IloSOS1Array & x)

This member function appends the elements in array to the invoking array.

public IloSOS1 operator[](IloInt i)
This operator returns a reference to the object located in the invoking array at the position indicated by the index \( i \). Concert Technology uses the \texttt{const} operator

\begin{verbatim}
IloSOS1 operator[] (IloInt i) const;
\end{verbatim}
on \texttt{const} arrays.

\begin{verbatim}
public IloSOS1 & operator[](IloInt i)
\end{verbatim}

This operator returns a reference to the object located in the invoking array at the position indicated by the index \( i \).
IloSOS2

Category    Class

InheritancePath

Definition File    ilconcert/ilolinear.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
</tbody>
</table>
This handle class represents special ordered sets of type 2 (SOS2). A special ordered set of type 2 specifies a set of variables, and only two among them may take a non zero value. These two variables must be adjacent. You may assign a weight to each variable in an SOS2. This weight indicates an order among the variables. Concert Technology asserts adjacency with respect to this assigned order. If you do not specify any weights to indicate order and adjacency among the variables, then Concert Technology considers the order in which you gave the variables to the constructor of this set and the order in which you added variables later (for example, by column generation).

When you extract a model (an instance of IloModel) for an instance of IloCplex (documented in the ILOG CPLEX Reference Manual), it will use the order of the SOS2 for branching on variables.

For more details about SOS2, see the ILOG CPLEX Reference and User's Manuals. Special ordered sets of type 2 (SOS2) commonly appear in models of piecewise linear functions. Concert Technology provides direct support for piecewise linear models in IloPiecewiseLinear.
Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

**See Also**

IloPiecewiseLinear, IloSOS1, IloSOS2Array

**Constructors**

public `IloSOS2()`  
This constructor creates an empty handle. You must initialize it before you use it.

public `IloSOS2(ImplClass * impl)`  
This constructor creates a handle object from a pointer to an implementation object.

public `IloSOS2(const IloEnv env,  
               const char * name)`  
This constructor creates a special ordered set of type 2 (SOS2). You must add the variables to this set for them to be taken into account.

public `IloSOS2(const IloEnv env,  
               const IloNumVarArray vars,  
               const char * name)`  
This constructor creates a special ordered set of type 2 (SOS2). The set includes each of the variables specified in the array `vars`.

public `IloSOS2(const IloEnv env,  
               const IloNumVarArray vars,  
               const IloNumArray vals,  
               const char * name)`  
This constructor creates a special ordered set of type 2 (SOS2). The set includes the variables specified in the array `vars`. The corresponding value in `vals` indicates the weight of each variable in `vars`.

**Methods**

public `ImplClass * getImpl()`  
This member function returns a pointer to the implementation object of the invoking handle.

public `void getNumVars(IloNumVarArray variables)`  
This member function accesses the variables in a special ordered set of type 2 (SOS2) and puts those variables into its parameter `variables`.

public `void getValues(IloNumArray values)`  
This member function accesses the weights of the variables in a special ordered set of type 2 (SOS2) and puts those weights into its parameter `values`.
IloSOS2Array

Category  Class

InheritancePath

Definition File  ilconcert/ilolinear.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloSOS2Array(IloDefaultArrayI *)</td>
</tr>
<tr>
<td>public</td>
<td>IloSOS2Array(const IloEnv, IloInt)</td>
</tr>
<tr>
<td>public</td>
<td>IloSOS2Array(const IloEnv, IloInt, const IloSOS2, const IloSOS2, ...)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>add(IloInt, const IloSOS2)</td>
</tr>
<tr>
<td>public void</td>
<td>add(const IloSOS2)</td>
</tr>
<tr>
<td>public void</td>
<td>add(const IloSOS2Array)</td>
</tr>
<tr>
<td>public IloSOS2</td>
<td>operator<a href="IloInt"></a></td>
</tr>
<tr>
<td>public IloSOS2 &amp;</td>
<td>operator<a href="IloInt"></a></td>
</tr>
</tbody>
</table>

Inherited methods from **IloConstraintArray**

add, add, add, operator[], operator[]
IloSOS2Array

**Description**
For each basic type, Concert Technology defines a corresponding array class. 
IloSOS2Array is the array class of special ordered sets of type 2 (SOS2) for a model. 
Instances of IloSOS2Array are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added to or removed from the array.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

For information on arrays, see the concept `Arrays`.

**See Also**
IloSOS2, operator>>, operator

**Constructors**
public `IloSOS2Array` (IloDefaultArrayI * i)
This default constructor creates an empty array. You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an optional parameter in functions and member functions that accept an array as a parameter.

public `IloSOS2Array` (const IloEnv env, 
 IloInt num)
This constructor creates an array of `num` empty elements in the environment `env`.

public `IloSOS2Array` (const IloEnv env, 
 IloInt num, 
 const IloSOS2 v0, 
 const IloSOS2 v1, 
 ...)
This constructor creates an array of `num` elements; these elements are initialized with the parameters `v0`, `v1`, and so forth.

**Methods**
public void `add` (IloInt more, 
 const IloSOS2 x)
This member function appends `x` to the invoking array multiple times. The parameter `more` indicates how many times.

public void `add` (const IloSOS2 x)
This member function appends x to the invoking array.

```java
public void add(\const \IloSOS2Array array)
```

This member function appends the elements in array to the invoking array.

```java
public \IloSOS2 operator[](\IloInt i)
```

This operator returns a reference to the object located in the invoking array at the position indicated by the index i. Concert Technology uses the \const operator

```java
\IloSOS2 operator[] (\IloInt i) const;
```

This operator returns a reference to the object located in the invoking array at the position indicated by the index i.

```java
public \IloSOS2 & operator[](\IloInt i)
```
IloScalProd

**Category**
Global Function

**Definition File**
ilconcert/iloexpression.h

**Synopsis**
public IloNumExprArg IloScalProd(const IloNumArray values, const IloNumVarArray vars)
public IloNumExprArg IloScalProd(const IloNumVarArray vars, const IloNumArray values)
public IloNumExprArg IloScalProd(const IloNumArray values, const IloIntVarArray vars)
public IloNumExprArg IloScalProd(const IloIntVarArray vars, const IloNumArray values)
public IloNumExprArg IloScalProd(const IloNumVarArray vars, const IloIntArray values)
public IloNumExprArg IloScalProd(const IloNumExprArray leftExprs, const IloNumVarArray rightExprs)

**Description**
This function returns an instance of IloNumExprArg, the internal building block of an expression, representing the scalar product of the variables in the arrays `vars` and `values` or the arrays `leftExprs` and `rightExprs`.
**IloScalProd**

**Category**
Global Function

**Definition File**
ilconcert/iloexpression.h

**Synopsis**

```c
public IloIntExprArg IloScalProd(const IloIntArray  values,
const IloIntVarArray  vars)

public IloIntExprArg IloScalProd(const IloIntVarArray  vars,
const IloIntArray  values)

public IloIntExprArg IloScalProd(const IloIntExprArray  leftExprs,
const IloIntExprArray  rightExprs)
```

**Description**

This function returns an instance of `IloIntExprArg`, the internal building block of an integer expression, representing the scalar product of the variables in the arrays `vars` and `values` or the arrays `leftExprs` and `rightExprs`. 
**IloScalProd**

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**

```
public IloNum IloScalProd(const IloNumArray vals1,  
const IloNumArray vals2)
public IloNum IloScalProd(const IloIntArray vals1,  
const IloNumArray vals2)
public IloNum IloScalProd(const IloNumArray vals1,  
const IloIntArray vals2)
```

**Description**  
This function returns a numeric value representing the scalar product of numeric values in the arrays `vals1` and `vals2`. 
**IloScalProd**

**Category**
Global Function

**Definition File**
ilconcert/iloexpression.h

**Synopsis**
public IloInt IloScalProd(const IloIntArray vals1, const IloIntArray vals2)

**Description**
This function returns an integer value representing the scalar product of integer values in the arrays vals1 and vals2.
IloSemaphore

Category                   Class
InheritancePath           
Definition File            ilconcert/ilothread.h

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IloSemaphore(int)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Public void</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>post()</td>
</tr>
<tr>
<td>Public int</td>
<td>tryWait()</td>
</tr>
<tr>
<td>Public void</td>
<td>wait()</td>
</tr>
</tbody>
</table>

Description

The class `IloSemaphore` provides synchronization primitives adapted to Concert Technology. This class supports inter-thread communication in multithread applications. An instance of this class, a semaphore, is a counter; its value is always positive. This counter can be incremented or decremented. You can always increment a semaphore, and incrementing is a non-blocking operation. However, the value of the counter cannot be negative, so any thread that attempts to decrement a semaphore whose counter is already equal to 0 (zero) is blocked until another thread increments the semaphore.

See ILOUSEMT for details about the compilation macro to use with instances of this class.

System Class

`IloSemaphore` is a system class.

Most Concert Technology classes are actually handle classes whose instances point to objects of a corresponding implementation class. For example, instances of the Concert Technology class `IloNumVar` are handles pointing to instances of the implementation class `IloNumVarI`. Their allocation and de-allocation in a Concert Technology environment are managed by an instance of `IloEnv`. 
However, system classes, such as IloSemaphore, differ from that pattern. IloSemaphore is an ordinary C++ class. Its instances are allocated on the C++ heap.

Instances of IloSemaphore are not automatically de-allocated by a call to the member function IloEnv::end. You must explicitly destroy instances of IloSemaphore by means of a call to the delete operator (which calls the appropriate destructor) when your application no longer needs instances of this class.

Furthermore, you should not allocate—neither directly nor indirectly—any instance of IloSemaphore in a Concert Technology environment because the destructor for that instance of IloSemaphore will never be called automatically by IloEnv::end when it cleans up other Concert Technology objects in that Concert Technology environment.

For example, it is not a good idea to make an instance of IloSemaphore part of a conventional Concert Technology model allocated in a Concert Technology environment because that instance will not automatically be de-allocated from the Concert Technology environment along with the other Concert Technology objects.

**De-allocating Instances of IloSemaphore**

Instances of IloSemaphore differ from the usual Concert Technology objects because they are not allocated in a Concert Technology environment, and their de-allocation is not managed automatically for you by IloEnv::end. Instead, you must explicitly destroy instances of IloSemaphore by calling the delete operator when your application no longer needs those objects.

**See Also**

IloBarrier, IloCondition, ILOUSEMT

**Constructors**

public IloSemaphore(int value)

This constructor creates an instance of IloSemaphore, initializes it to value, and allocates it on the C++ heap (not in a Concert Technology environment). If you do not pass a value parameter, the constructor initializes the semaphore at 0 (zero).

**Methods**

public void post()

This member function increments the invoking semaphore by 1 (one). If there are threads blocked at this semaphore, then this member function wakes one of them.

public int tryWait()

This member function attempts to decrement the invoking semaphore by 1 (one). If this decrement leaves the counter positive, then the call succeeds and returns 1 (one). If the decrement would make the counter strictly negative, then the decrement does not occur, the call fails, and the member function returns 0 (zero).

public void wait()

This member function decrements the invoking semaphore by 1 (one).
If this decrement would make the semaphore strictly negative, then this member function blocks the calling thread. The thread wakes up when the member function can safely decrement the semaphore without causing the counter to become negative (for example, if another entity increments the semaphore). If this member function cannot decrement the invoking semaphore, then it leads to deadlock.
IloSemiContVar

**Category**
Class

**Inheritance Path**

**Definition File**
`ilconcert/iloexpression.h`

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloSemiContVar()</td>
</tr>
<tr>
<td>public</td>
<td>IloSemiContVar(ImplClass *)</td>
</tr>
<tr>
<td>public</td>
<td>IloSemiContVar(const IloEnv, IloNum, IloNum, IloNumVar::Type, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloSemiContVar(const IloAddNumVar &amp;, IloNum, IloNum, IloNumVar::Type, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ImplClass *</td>
<td>getImpl()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getSemiContLB()</td>
</tr>
<tr>
<td>public void</td>
<td>setSemiContLB(IloNum)</td>
</tr>
</tbody>
</table>
Description

An instance of this class represents a constrained semicontinuous variable in a Concert Technology model. Semicontinuous variables derive from `IloNumVar`, the class of numeric variables.

A semicontinuous variable may be 0 (zero) or it may take a value within an interval defined by its semicontinuous lower and upper bound. Conventionally, semicontinuous variables are defined as floating-point variables, but you can designate an instance of `IloSemiContVar` as integer by using the type indicator it inherits from `IloNumVar`. In that case, Concert Technology will impose an integrality constraint on the semicontinuous variable for you, thus further restricting the feasible set of values to 0 (zero) and the integer values in the interval defined by the semicontinuous lower and upper bound.

Note: When numerical bounds are given to an integer variable (an `IloIntVar` or an `IloNumVar` with Type = Int) in the constructors of via a modifier (setUB, setLB, setBounds), they are inward rounded to an integer value. LB is rounded down and UB is rounded up.
In an instance of `IloNumVar`, `lb` denotes the lower bound of the variable, and `ub` denotes its upper bound. In an instance of the derived class `IloSemiContVar`, `sclb` denotes the semicontinuous lower bound.

In formal terms, if `lb <= 0`, then a semicontinuous variable is a numeric variable with the feasible set of `{0, [sclb, ub]}`, where `0 < sclb < ub`; otherwise, for other values of `lb`, the feasible set of a semicontinuous variable is the intersection of the interval `[lb, ub]` with the set `{0, [sclb, ub]}`. The semicontinuous lower bound `sclb` may differ from the lower bound of an ordinary numeric variable in that the semicontinuous variable is restricted to the semicontinuous region. For example, the table below shows you the bounds of a semicontinuous variable and the corresponding feasible region.

### Examples of bounds on semicontinuous variables and their feasible regions

<table>
<thead>
<tr>
<th>These conditions</th>
<th>define these feasible regions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lb == ub &lt; sclb</code></td>
<td><code>{0}</code> if <code>lb==ub==0</code> or empty set if <code>lb==ub!=0</code></td>
</tr>
<tr>
<td><code>lb &lt; 0 &lt; sclb &lt; ub</code></td>
<td><code>{0, [sclb, ub]}</code></td>
</tr>
<tr>
<td><code>0 &lt; lb &lt; sclb &lt; ub</code></td>
<td><code>[sclb, ub]</code></td>
</tr>
<tr>
<td><code>0 &lt; sclb &lt; lb &lt; ub</code></td>
<td><code>[lb, ub]</code></td>
</tr>
</tbody>
</table>

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept `Assert and NDEBUG`.

### See Also

- `IloNumVar`

### Constructors

- **Public** `IloSemiContVar()`

  This constructor creates an empty handle. You must initialize it before you use it.

- **Public** `IloSemiContVar(ImplClass * impl)`

  This constructor creates a handle object from a pointer to an implementation object.

- **Public** `IloSemiContVar(const IloEnv env, IloNum sclb, IloNum ub, IloNumVar::Type type, const char * name)`

  This constructor creates an instance of `IloSemiContVar` from its `sclb` (that is, its semicontinuous lower bound) and its upper bound `ub`. By default, its type is floating-
point, but you can use ILOINT to indicate integer; in that case, Concert Technology will impose an integrality constraint on the variable. The value for lb is set to zero.

```java
public IloSemiContVar(const IloAddNumVar & var,
                      IloNum sclb,
                      IloNum ub,
                      IloNumVar::Type type,
                      const char * name)
```

This constructor creates an instance of IloSemiContVar from the prototype var.

### Methods

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.

```java
public IloNum getSemiContLB()
```

This member function returns the semicontinuous lower bound (that is, its sclb) of the invoking semicontinuous variable.

```java
public void setSemiContLB(IloNum sclb)
```

This member function makes sclb the semicontinuous lower bound of the invoking semicontinuous variable.

---

**Note:** The member function setSemiContinuousLb notifies Concert Technology algorithms about this change of this invoking object.
**IloSemiContVarArray**

**Category**  
Class

**InheritancePath**

```
  IloSemContValArray  
  IloNumExprArray  
  IloNumVarArray  
  IloExtractableArray
```

**Definition File**  
ilconcert/iloexpression.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(IloDefaultArrayI *)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(const IloEnv)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(const IloEnv, IloInt)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(const IloEnv, IloInt, IloNum, IloNum, IloNumVar::Type)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(const IloEnv, const IloNumColumnArray, const IloNumArray, const IloNumArray, IloNumVar::Type)</code></td>
</tr>
<tr>
<td>public</td>
<td><code>IloSemiContVarArray(const IloEnv, IloInt, const IloSemContVar, const IloSemContVar v1...)</code></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td><code>add(IloInt, const IloSemContVar)</code></td>
</tr>
</tbody>
</table>
Description

For each basic type, Concert Technology defines a corresponding array class. 
IloSemiContVarArray is the array class of the semicontinuous numeric variable 
class for a model.

Instances of IloSemiContVarArray are extensible. That is, you can add more 
elements to such an array. References to an array change whenever an element is added 
to or removed from the array.

Most member functions in this class contain assert statements. For an explanation of 
the macro NDEBUG (a way to turn on or turn off these assert statements), see the 
concept Assert and NDEBUG.

See Also

IloSemiContVar, IloModel, IloNumVar, operator>>, operator

Constructors

public IloSemiContVarArray(IloDefaultArrayI * i)

This constructor creates an empty extensible array of semicontinuous numeric variables. 
You cannot create instances of the undocumented class IloDefaultArrayI. As a parameter in this default constructor, it allows you to pass 0 (zero) as a value to an
optional parameter in functions and member functions that accept an array as a parameter.

public IloSemiContVarArray(const IloEnv env)

This constructor creates an extensible array of semicontinuous numeric variables in env. Initially, the array contains zero elements.

public IloSemiContVarArray(const IloEnv env, IloInt n)

This constructor creates an extensible array of n semicontinuous numeric variables in env. Initially, the n elements are empty handles.

public IloSemiContVarArray(const IloEnv env, IloInt n, IloNum sclb, IloNum ub, IloNumVar::Type type)

This constructor creates an extensible array of n semicontinuous numeric variables in the environment env. Each element of the array has a semicontinuous lower bound of sclb and an upper bound of ub. The type (whether integer, Boolean, or floating-point) of each element is indicated by type. The default type is floating-point.

public IloSemiContVarArray(const IloEnv env, const IloNumColumnArray columnarray, const IloNumArray sclb, const IloNumArray ub, IloNumVar::Type type)

This constructor creates an extensible array of semicontinuous numeric variables from a column array in the environment env. The array sclb indicates the corresponding semicontinuous lower bound, and the array ub indicates the corresponding upper bound for each new element. The parameter type indicates the type (whether integer, Boolean, or floating point) of each new element. The default type is floating-point.

public IloSemiContVarArray(const IloEnv env, IloInt n, const IloSemiContVar v0, const IloSemiContVar v1...)

This constructor creates an extensible array of n semicontinuous numeric variables, as indicated by v0, v1, and so on.

Methods

public void add(IloInt more, const IloSemiContVar x)

This member function appends x to the invoking array multiple times. The parameter more indicates how many times.

public void add(const IloSemiContVar x)
This member function appends $x$ to the invoking array.

public void \textbf{add}(\textit{const IloSemiContVarArray array})

This member function appends the elements in \textit{array} to the invoking array.
IloSequence

Category
Class

Inheritance Path

Definition File: ilconcert/ilomodel.h

Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloSequence()</td>
<td>IloSequence()</td>
</tr>
<tr>
<td>ImplClass *</td>
<td>IloSequence(ImplClass *)</td>
</tr>
<tr>
<td>const IloEnv,</td>
<td>IloSequence(const IloEnv,</td>
</tr>
<tr>
<td>IloInt,</td>
<td>IloInt,</td>
</tr>
<tr>
<td>IloInt,</td>
<td>IloIntVarArray,</td>
</tr>
<tr>
<td>const IloIntVarArray,</td>
<td>const IloIntArray,</td>
</tr>
<tr>
<td>const IloIntVarArray,</td>
<td>const IloIntVarArray,</td>
</tr>
<tr>
<td>const char *</td>
<td>const char *)</td>
</tr>
</tbody>
</table>

Method Summary

| Public ImplClass * | getImpl()                                      |

Inherited methods from IloConstraint

IloConstraint::getImpl
Description

An instance of this class represents a sequence constraint in a model. As you can see from the parameters of its constructor, an instance of this class enables you to constrain:

◆ the minimum number of allowable values in the sequence,
◆ the maximum number of allowable values in the sequence,
◆ the frequency of allowable values (that is, how often a value occurs in the sequence),
◆ the number of elements in the sequence.

In order for the constraint to take effect, you must add it to a model with the template `IloAdd` or the member function `IloModel::add` and extract the model for an algorithm with the member function `IloAlgorithm::extract`.

Most member functions in this class contain `assert` statements. For an explanation of the macro `NDEBUG` (a way to turn on or turn off these `assert` statements), see the concept Assert and NDEBUG.

See Also

`IloConstraint`, `IloDistribute`

Constructors

```
public IloSequence()
```

This constructor creates an empty handle. You must initialize it before you use it.

```
public IloSequence(ImplClass * impl)
```
This constructor creates a handle object from a pointer to an implementation object.

```java
public IloSequence(const IloEnv env,
   IloInt nbMin,
   IloInt nbMax,
   IloInt seqWidth,
   const IloIntVarArray vars,
   const IloIntArray values,
   const IloIntVarArray cards,
   const char * name)
```

This constructor creates a sequence constraint in an environment. The parameter `nbMin` indicates a minimum number of allowable values, and `nbMax` indicates a maximum number of allowable values. The parameter `seqWidth` indicates the number of elements in a sequence. The parameter `cards` indicates an array of cardinalities (that is, how many occurrences).

In the new constraint created by this class, the constrained variables in the array `cards` will be equal to the number of occurrences in the array `vars` of the values in the array `values` such that for each sequence of `seqWidth` (a number) consecutive constrained variables of `vars`, at least `nbMin` and at most `nbMax` values are assigned to a constrained variable of the sequence.

The arrays `cards` and `values` must be the same length; otherwise, on platforms where C++ exceptions are supported and exceptions are enabled, Concert Technology throws the exception `InvalidArraysException`.

### Methods

```java
public ImplClass * getImpl()
```

This member function returns a pointer to the implementation object of the invoking handle.
IloSolution

Category       Class
InheritancePath

Definition File  ilconcert/ilosolution.h

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public IloBool</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>public IloEnv</td>
</tr>
<tr>
<td>public IloSolutionI *</td>
</tr>
<tr>
<td>public IloNum</td>
</tr>
<tr>
<td>public IloNum</td>
</tr>
<tr>
<td>public const char *</td>
</tr>
<tr>
<td>public IloAny</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>public IloObjective getObjective()</td>
</tr>
<tr>
<td>public IloNum getObjectiveValue()</td>
</tr>
<tr>
<td>public IloNumVar getObjectiveVar()</td>
</tr>
<tr>
<td>public IloAnySet getPossibleSet(IloAnySetVar)</td>
</tr>
<tr>
<td>public IloAnySet getRequiredSet(IloAnySetVar)</td>
</tr>
<tr>
<td>public IloAny getValue(IloAnyVar)</td>
</tr>
<tr>
<td>public IloNum getValue(IloNumVar)</td>
</tr>
<tr>
<td>public IloNum getValue(IloObjective)</td>
</tr>
<tr>
<td>public IloBool isBetterThan(IloSolution)</td>
</tr>
<tr>
<td>public IloBool isBound(IloAnySetVar)</td>
</tr>
<tr>
<td>public IloBool isBound(IloNumVar)</td>
</tr>
<tr>
<td>public IloBool isEquivalent(IloExtractable, IloSolution)</td>
</tr>
<tr>
<td>public IloBool isEquivalent(IloSolution)</td>
</tr>
<tr>
<td>public IloBool isObjectiveSet()</td>
</tr>
<tr>
<td>public IloBool isRestorable(IloExtractable)</td>
</tr>
<tr>
<td>public IloBool isRestorable(IloSolution)</td>
</tr>
<tr>
<td>public IloBool isWorseThan(IloSolution)</td>
</tr>
<tr>
<td>public IloSolution makeClone(IloEnv)</td>
</tr>
<tr>
<td>public void operator=(const IloSolution &amp;)</td>
</tr>
<tr>
<td>public void remove(IloExtractableArray)</td>
</tr>
<tr>
<td>public void remove(IloExtractable)</td>
</tr>
<tr>
<td>public void restore(IloExtractable, IloAlgorithm)</td>
</tr>
<tr>
<td>public void restore(IloAlgorithm)</td>
</tr>
<tr>
<td>public void setFalse(IloBoolVar)</td>
</tr>
<tr>
<td>public void setMax(IloNumVar, IloNum)</td>
</tr>
<tr>
<td>public void setMin(IloNumVar, IloNum)</td>
</tr>
<tr>
<td>public void setName(const char *)</td>
</tr>
<tr>
<td>public void setNonRestorable(IloExtractableArray)</td>
</tr>
<tr>
<td>public void setNonRestorable(IloExtractable)</td>
</tr>
<tr>
<td>public void setObject(IloAny)</td>
</tr>
<tr>
<td>public void setObjective(IloObjective)</td>
</tr>
<tr>
<td>public void setPossibleSet(IloAnySetVar, IloAnySet)</td>
</tr>
<tr>
<td>public void setRequiredSet(IloAnySetVar, IloAnySet)</td>
</tr>
<tr>
<td>public void setRestorable(IloExtractableArray)</td>
</tr>
<tr>
<td>public void setRestorable(IloExtractable)</td>
</tr>
<tr>
<td>public void setTrue(IloBoolVar)</td>
</tr>
<tr>
<td>public void setValue(IloAnyVar, IloAny)</td>
</tr>
</tbody>
</table>
Instances of this class store solutions to problems. The fundamental property of `IloSolution` is its ability to transfer stored values from or to the active objects associated with it. In particular, the member function `store` stores the values from algorithm variables while the member function `restore` instantiates the actual variables with stored values. Variables in the solution may be selectively restored. This class also offers member functions to copy and to compare solutions.

Information about these classes of variables can be stored in an instance of `IloSolution`:

- **IloAnySet**: the required and possible sets are stored; when the variable is bound, the required and possible sets are equivalent.
- **IloAnyVar**: the value of the variable is stored.
- **IloBoolVar**: the value (true or false) of the variable is stored. Some of the member functions for `IloBoolVar` are covered by the member function for `IloNumVar`, as `IloBoolVar` is a subclass of `IloNumVar`. For example, there is no explicit member function to add objects of type `IloBoolVar`.
- **IloIntSetVar**: the required and possible sets are stored; when the variable is bound, the required and possible sets are equivalent.
- **IloNumVar**: the lower and upper bounds are stored; when the variable is bound, the current lower and upper bound are equivalent.
- **IloObjective**: the value of the objective is stored. Objectives are never restored; operations such as `setRestorable` cannot change this. More than one instance of `IloObjective` can be added to a solution. In such cases, there is the notion of an active objective, which is returned by `IloSolution::getObjective`.

### Inner Class

**IloSolution::IloSolution::Iterator**

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setValue</td>
<td><code>setValue(IloNumVar, IloNum)</code></td>
</tr>
<tr>
<td>public void setValue</td>
<td><code>setValue(IloObjective, IloNum)</code></td>
</tr>
<tr>
<td>public void store</td>
<td><code>store(IloExtractable, IloAlgorithm)</code></td>
</tr>
<tr>
<td>public void store</td>
<td><code>store(IloAlgorithm)</code></td>
</tr>
<tr>
<td>public void unsetObjective</td>
<td><code>unsetObjective()</code></td>
</tr>
</tbody>
</table>
active objective typically specifies the optimization criterion for the problem to which the solution object is a solution. For example, the ILOG Solver class IloImprove uses the notion of an active objective.

Most member functions in this class contain assert statements. For an explanation of the macro NDEBUG (a way to turn on or turn off these assert statements), see the concept Assert and NDEBUG.

Objects of type IloSolution have a scope, comprising the set of variables that have their values stored in the solution. The scope is given before the basic operations of storing and restoring are performed, via add and remove methods. For example,

```cpp
IloNumVar var(env);
IloSolution soln(env);
solution.add(var);
```

creates a numerical variable and a solution and adds the variable to the solution. Arrays of variables can also be added to the solution. For example,

```cpp
IloNumVarArray arr(env, 10, 0, 1);
soln.add(arr);
```

adds 10 variables with range \([0...1]\). When an array of variables is added to the solution, the array object itself is not present in the scope of the solution; only the elements are present. If the solution is then stored by means of soln.store(algorithm), the values of variable var and arr[0] to arr[9] are saved. Any attempt to add a variable that is already present in a solution throws an exception, an instance of IloException.

Accessors allow access to the stored values of the variables, regardless of the state (or existence) of the algorithm they were stored from. For example,

```cpp
cout << "arr[3] = " << soln.getValue(arr[3]) << endl;
```

Any attempt to access a variable that is not present in the solution throws an instance of IloException.

A variable or an array of variables can be removed from a solution. For example,

```cpp
soln.remove(var);
```
removes \texttt{var} from the scope of the solution; and
\begin{verbatim}
soln.remove(arr);
\end{verbatim}
removes \texttt{arr[0]} to \texttt{arr[9]} from the solution.

Any attempt to remove a variable that is not present in the solution throws an \texttt{IloException}.

\textbf{See Also} \quad \texttt{IloAnySetVar}, \texttt{IloAnyVar}, \texttt{IloNumVar}, \texttt{IloIntSetVar}, \texttt{IloObjective}, \texttt{IloRestoreSolution} documented in the ILOG Solver Reference Manual, \texttt{IloStoreSolution} documented in the ILOG Solver Reference Manual

\textbf{Constructors}

\texttt{public IloSolution()}

This constructor creates a solution whose implementation pointer is 0 (zero). The handle must be assigned before its methods can be used.

\texttt{public IloSolution(IloSolutionI * impl)}

This constructor creates a handle object (an instance of the class \texttt{IloSolution}) from a pointer to an implementation object (an instance of the class \texttt{IloSolutionI}).

\texttt{public IloSolution(const IloSolution & solution)}

This constructor creates a handle object from a reference to a solution. After execution, both the newly constructed handle and \texttt{solution} point to the same implementation object.

\texttt{public IloSolution(IloEnv env,}
\begin{verbatim}
    const char * name)
\end{verbatim}

This constructor creates an instance of the \texttt{IloSolution} class. The optional argument \texttt{name}, if supplied, becomes the name of the created object.

\textbf{Methods}

\texttt{public void add(IloAnySetVarArray a)}

This member function adds each element of \texttt{array} to the invoking solution.

\texttt{public void add(IloAnySetVar var)}

This member function adds the set variable \texttt{var} to the invoking solution.

\texttt{public void add(IloAnyVarArray a)}

This member function adds each element of \texttt{array} to the invoking solution.

\texttt{public void add(IloAnyVar var)}

This member function adds the variable \texttt{var} to the invoking solution.

\texttt{public void add(IloNumVarArray a)}
This member function adds each element of array to the invoking solution.

```java
public void add(IloNumVar var)
```

This member function adds the variable var to the invoking solution.

```java
public void add(IloObjective objective)
```

This member function adds objective to the invoking solution. If the solution has no active objective, then objective becomes the active objective. Otherwise, the active objective remains unchanged.

```java
public IloBool contains(IloExtractable extr)
```

This member function returns IloTrue if extr is present in the invoking object. Otherwise, it returns IloFalse.

```java
public void copy(IloExtractable extr, IloSolution solution)
```

This member function copies the saved value of extr from solution to the invoking solution. If extr does not exist in either solution or the invoking object, this member function throws an instance of IloException. The restorable status of extr is not copied.

```java
public void copy(IloSolution solution)
```

For each variable that has been added to solution, this member function copies its saved data to the invoking solution. If a particular extractable does not already exist in the invoking solution, it is automatically added first. If variables were added to the invoking solution, their restorable status is the same as in solution. Otherwise, their status remains unchanged in the invoking solution.

```java
public void end()
```

This member function deallocates the memory used to store the solution. If you no longer need a solution, calling this member function can reduce memory consumption.

```java
public IloEnv getEnv()
```

This member function returns the environment specified when the invoking object was constructed.

```java
public IloSolutionI * getImpl()
```

This member function returns a pointer to the implementation object corresponding to the invoking solution.

```java
public IloNum getMax(IloNumVar var)
```

This member function returns the maximal value of the variable var in the invoking solution.

```java
public IloNum getMin(IloNumVar var)
```
This member function returns the minimal value of the variable \texttt{var} in the invoking solution.

\begin{verbatim}
public const char * getName()
\end{verbatim}

This member function returns a character string indicating the name of the invoking object (if there is one).

\begin{verbatim}
public IloAny getObject()
\end{verbatim}

This member function returns the object associated with the invoking object (if there is one). Normally, an associated object contains user data pertinent to the invoking object.

\begin{verbatim}
public IloObjective getObjective()
\end{verbatim}

This member function returns the active objective as set via a previous call to \texttt{add} or \texttt{setObjective(IloObjective)}. If there is no active objective, an empty handle is returned.

\begin{verbatim}
public IloNum getObjectiveValue()
\end{verbatim}

This member function returns the saved value of the current active objective. It can be seen as performing the action getValue(getObjective()).

\begin{verbatim}
public IloNumVar getObjectiveVar()
\end{verbatim}

If the active objective corresponds to a simple IloNumVar, this member function returns that variable. If there is no active objective or if the objective is not a simple variable, an empty handle is returned.

\begin{verbatim}
public IloAnySet getPossibleSet(IloAnySetVar var)
\end{verbatim}

This member function returns the set of possible values for the variable \texttt{var}, as stored in the invoking solution.

\begin{verbatim}
public IloAnySet getRequiredSet(IloAnySetVar var)
\end{verbatim}

This member function returns the set of required values for the variable \texttt{var}, as stored in the invoking solution.

\begin{verbatim}
public IloAny getValue(IloAnyVar var)
\end{verbatim}

This member function returns the value of the variable \texttt{var} in the invoking solution.

\begin{verbatim}
public IloNum getValue(IloNumVar var)
\end{verbatim}

This member function returns the value of the variable \texttt{var} in the invoking solution. If the saved minimum and maximum of the variable are not equal, this member function throws an instance of IloException.

\begin{verbatim}
public IloNum getValue(IloObjective obj)
\end{verbatim}

This member function returns the saved value of objective \texttt{objective} in the invoking solution.

\begin{verbatim}
public IloBool isBetterThan(IloSolution solution)
\end{verbatim}
This member function returns `IloTrue` if the invoking solution and `solution` have the same objective and if the invoking solution has a strictly higher quality objective value (according to the sense of the objective). In all other situations, it returns `IloFalse`.

```java
class IloSolution {
public IloBool isBound(IloAnySetVar var)
```
This member function returns `IloTrue` if the stored required and possible sets for the set variable `var` are equal in the invoking solution. Otherwise, it returns `IloFalse`.

```java
public IloBool isBound(IloNumVar var)
```
This member function returns `IloTrue` if `var` takes a single value in the invoking solution. Otherwise, it returns `IloFalse`.

```java
public IloBool isEquivalent(IloExtractable extr, IloSolution solution)
```
This member function returns `IloTrue` if the saved value of `extr` is the same in the invoking solution and `solution`. Otherwise, it returns `IloFalse`. If `extr` does not exist in either `solution` or the invoking object, the member function throws an instance of `IloException`.

```java
public IloBool isEquivalent(IloSolution solution)
```
This member function returns `IloTrue` if the invoking object and `solution` contain the same variables with the same saved values. Otherwise, it returns `IloFalse`.

```java
public IloBool isObjectiveSet()
```
This member function returns `IloTrue` if the invoking solution has an active objective. Otherwise, it returns `IloFalse`.

```java
public IloBool isRestorable(IloExtractable extr)
```
This member function returns `IloFalse` if `setNonRestorable(extr)` was called more recently than `setRestorable(extr)`. Otherwise, it returns `IloTrue`. This member function always returns `IloFalse` when it is passed an `IloObjective` object.

```java
public IloBool isWorseThan(IloSolution solution)
```
This member function returns `IloTrue` if the invoking solution and `solution` have the same objective and if the invoking solution has a strictly lower quality objective value (according to the sense of the objective). In all other situations, it returns `IloFalse`.

```java
public IloSolution makeClone(IloEnv env)
```
This member function allocates a new solution on `env` and adds to it all variables that were added to the invoking object. The “restorable” status of all variables in the clone is the same as that in the invoking solution. Likewise, the active objective in the clone is the same as that in the invoking solution. The newly created solution is returned.
public void \texttt{operator=}(\texttt{const IloSolution} \& \texttt{solution})

This operator assigns an address to the handle pointer of the invoking solution. That address is the location of the implementation object of \texttt{solution}. After the execution of this operator, the invoking solution and \texttt{solution} both point to the same implementation object.

public void \texttt{remove}(IloExtractableArray \texttt{extr})

This member function removes each element of \texttt{array} from the invoking solution. If the invoking solution does not contain all elements of \texttt{array}, the member function throws an instance of \texttt{IloException}.

public void \texttt{remove}(IloExtractable \texttt{extr})

This member function removes extractable \texttt{extr} from the invoking solution. If the invoking solution does not contain \texttt{extr}, the member function throws an instance of \texttt{IloException}.

public void \texttt{restore}(IloExtractable \texttt{extr},
\hspace{1em}IloAlgorithm \texttt{algorithm})

This member function restores the value of the extractable corresponding to \texttt{extr} by reference to \texttt{algorithm}. The use of this member function depends on the state of \texttt{algorithm}. If \texttt{algorithm} is an instance of the ILOG Solver class \texttt{IloSolver}, this member function can only be used during search. If \texttt{extr} does not exist in the invoking solution, the member function throws an instance of \texttt{IloException}.

public void \texttt{restore}(IloAlgorithm \texttt{algorithm})

This member function uses \texttt{algorithm} to instantiate the variables in the invoking solution with their saved values. The value of any objective added to the solution is not restored. The use of this member function depends on the state of \texttt{algorithm}. If \texttt{algorithm} is an instance of the ILOG Solver class \texttt{IloSolver}, this member function can only be used during search.

public void \texttt{setFalse}(IloBoolVar \texttt{var})

This member function sets the stored value of \texttt{var} to \texttt{IloFalse} in the invoking solution.

public void \texttt{setMax}(IloNumVar \texttt{var},
\hspace{1em}IloNum \texttt{max})

This member function sets the maximal value of the variable \texttt{var} in the invoking solution to \texttt{max}.

public void \texttt{setMin}(IloNumVar \texttt{var},
\hspace{1em}IloNum \texttt{min})

This member function sets the minimal value of the variable \texttt{var} in the invoking solution to \texttt{min}.

public void \texttt{setName}(const char * \texttt{name})
This member function assigns name to the invoking object.

```java
public void setNonRestorable(IloExtractableArray array)
```

This member function indicates to the invoking solution that when the solution is restored by means of `restore(IloAlgorithm)` or `restore(IloExtractable, IloAlgorithm)`, no elements of `array` will be restored. When an array of variables is added to a solution, each variable is added in a “restorable” state.

```java
public void setNonRestorable(IloExtractable extr)
```

This member function indicates to the invoking solution that when the solution is restored by means of `restore(IloAlgorithm)` or `restore(IloExtractable, IloAlgorithm)`, `extr` will not be restored. When a variable is added to a solution, it is added in a “restorable” state.

```java
public void setObject(IloAny object)
```

This member function associates `obj` with the invoking object. The member function `getObject` accesses this associated object afterwards. Normally, `obj` contains user data pertinent to the invoking object.

```java
public void setObjective(IloObjective objective)
```

This member function adds `objective` to the invoking solution, if it is not already present, and sets the active objective to `objective`.

```java
public void setPossibleSet(IloAnySetVar var, IloAnySet possible)
```

This member function sets the stored possible values for `var` as `possible` in the invoking solution.

```java
public void setRequiredSet(IloAnySetVar var, IloAnySet required)
```

This member function sets the stored required values for `var` as `required` in the invoking solution.

```java
public void setRestorable(IloExtractableArray array)
```

This member function indicates to the invoking solution that when the solution is restored by means of `restore(IloAlgorithm)` or `restore(IloExtractable, IloAlgorithm)`, the appropriate element(s) of `array` will be restored. When an array of variables is added to a solution, each variable is added in a “restorable” state. This call has no effect on objects of type `IloObjective`; objects of this type are never restored.

```java
public void setRestorable(IloExtractable ex)
```

This member function indicates to the invoking solution that when the solution is restored by means of `restore(IloAlgorithm)` or...
restore(IloExtractable, IloAlgorithm).extr will be restored. When a variable is added to a solution, it is added in a “restorable” state. This call has no effect on objects of type IloObjective; objects of this type are never restored.

public void setTrue(IloBoolVar var)

This member function sets the stored value of var to IloTrue in the invoking solution.

public void setValue(IloAnyVar var, IloAny value)

This member function sets the value of the variable var to value in the invoking solution.

public void setValue(IloNumVar var, IloNum value)

This member function sets the value (both minimum and maximum) of the variable var to value in the invoking solution.

public void setValue(IloObjective objective, IloNum value)

This member function sets the value of objective as stored in the invoking solution to value. This member function should be used with care and only when the objective value of the solution is known exactly.

public void store(IloExtractable extr, IloAlgorithm algorithm)

This member function stores the value of the extractable corresponding to extr by reference to algorithm. If extr does not exist in the invoking solution, the member function throws an instance of IloException.

public void store(IloAlgorithm algorithm)

This member function stores the values of the objects added to the solution by reference to algorithm.

public void unsetObjective()

This member function asserts that there should be no active objective in the invoking solution, although the previous active object is still present. A new active objective can be set via add or setObjective.
**IloSolutionArray**

<table>
<thead>
<tr>
<th>Category</th>
<th>Type Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition File</td>
<td>ilconcert/ilosolution.h</td>
</tr>
<tr>
<td>Synopsis</td>
<td>IloSimpleArray&lt; IloSolution &gt; IloSolutionArray</td>
</tr>
<tr>
<td>Description</td>
<td>This type definition represents arrays of instances of IloSolution. Instances of IloSolutionArray are extensible. That is, you can add more elements to such an array. References to an array change whenever an element is added or removed from the array.</td>
</tr>
</tbody>
</table>

**See Also**

IloSolution
**IloSolution::Iterator**

**Category**       Inner Class

**InheritancePath**

**Definition File**  ilconcert/ilosolution.h

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool ok()</td>
</tr>
<tr>
<td>public IloExtractable operator *()</td>
</tr>
<tr>
<td>public Iterator &amp; operator++()</td>
</tr>
</tbody>
</table>

**Description**  
Iterator is a class nested in the class IloSolution. It allows you to traverse the variables in a solution. The iterator scans the objects in the same order as they were added to the solution.

This iterator is not robust. If the variable at the current position is deleted from the solution being iterated over, the behavior of this iterator afterwards is undefined.

- *iter* can be safely used after the following code has executed:

  ```cpp
  IloExtractable elem = *iter;
  ++iter;
  solution.remove(elem);
  ```

- *iter* cannot be safely used after the following code has executed:

  ```cpp
  solution.remove(*iter);  // bad idea
  ++iter;
  ```

**See Also**  
IloIterator, IloSolution

**Methods**

public IloBool ok()  

This member function returns IloTrue if the current position of the iterator is a valid one. It returns IloFalse if all variables have been scanned by the iterator.
public IloExtractable operator *()

This operator returns the extractable object corresponding to the variable located at the current iterator position. If all variables have been scanned, this operator returns an empty handle.

public Iterator & operator++()

This operator moves the iterator to the next variable in the solution.
IloSolutionIterator

Category          Class
InheritancePath

IloSolutionIterator

Definition File  ilconcert/ilosolution.h

### Constructor Summary

| public | IloSolutionIterator(IloSolution) |

### Method Summary

| public E | operator *()                   |
| public void | operator++()                   |

### Description

This template class creates a typed iterator over solutions. You can use this iterator to discover all extractable objects added to a solution and of a particular type. The type is denoted by $E$ in the template.

This iterator is not robust. If the variable at the current position is deleted from the solution being iterated over, the behavior of this iterator afterwards is undefined.

An iterator created with this template differs from an instance of `IloSolution::Iterator`. An instance of `IloSolution::Iterator` works on all extractable objects within a given solution (an instance of `IloSolution`). In contrast, an iterator created with this template only iterates over extractable objects of the specified type.

### See Also

`IloSolution`, `IloSolution::Iterator`

### Constructors

public **IloSolutionIterator**(IloSolution s)

This constructor creates an iterator for instances of the class $E$.

### Methods

public E **operator *()**
This operator returns the current element, the one to which the invoking iterator points. This current element is a handle to an extractable object (not a pointer to the implementation object).

```java
public void operator++()
```

This operator advances the iterator by one position.
IloSolutionManip

Category            Class
InheritancePath

Definition File     ilconcert/ilosolution.h

Constructor Summary

| Public          | IloSolutionManip(IloSolution, IloExtractable) |

Description An instance of this class accesses a specific part of a solution so that you can display it. You construct the class IloSolutionManip from a solution and an extractable object. You use the operator<< with this constructed class to display information stored on the specified extractable object in the solution.

See Also          IloSolution, operator

Constructors public IloSolutionManip(IloSolution solution, IloExtractable extr)

This constructor creates an instance of IloSolutionManip from the solution indicated by solution and from the extractable object extr. The constructor throws an exception (an instance of IloException) if extr has not been added to solution. You can use the operator<< with the newly created object to display the information in extr stored in solution.
IloSquare

Category Global Function

Definition File ilconcert/iloexpression.h

Synopsis

public IloNumExprArg IloSquare(const IloNumExprArg arg)
public IloNum IloSquare(IloNum val)
public IloInt IloSquare(IloInt val)
public IloInt IloSquare(int val)
public IloIntExprArg IloSquare(const IloIntExprArg arg)

Description

Concert Technology offers predefined functions that return an expression from an algebraic function on expressions. These predefined functions also return a numeric value from an algebraic function on numeric values as well.

IloSquare returns the square of its argument (that is, val*val or expr*expr).

What Is Extracted

IloSquare is extracted by an instance of IloSolver (documented in the ILOG Solver Reference Manual) as an instance of IlcSquare.
IloSubset

Category                   Global Function
Definition File           ilconcert/iloanyset.h

Synopsis

public IloConstraint IloSubset(const IloEnv  env,
                               const IloAnySetVar  var1,
                               const IloAnySetVar  var2)
public IloConstraint IloSubset(const IloEnv  env,
                               const IloAnySet  var1,
                               const IloAnySetVar  var2)
public IloConstraint IloSubset(const IloEnv  env,
                               const IloAnySetVar  var1,
                               const IloAnySet  var2)
public IloConstraint IloSubset(const IloEnv  ,
                               const IloIntSetVar  var1,
                               const IloIntSetVar  var2)
public IloConstraint IloSubset(const IloEnv  ,
                               const IloIntSet  var1,
                               const IloIntSetVar  var2)
public IloConstraint IloSubset(const IloEnv  ,
                               const IloIntSetVar  var1,
                               const IloIntSet  var2)

Description

This function creates and returns a constraint (an instance of IloConstraint) for use in a model. That constraint forces var1 to be strictly a subset of var2. That is, there is at least one element of var2 not in var1, and all elements of var1 are in var2.

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
**IloSubsetEq**

**Category**  
Global Function

**Definition File**  
ilconcert/iloanyset.h

**Synopsis**

```cpp
public IloConstraint IloSubsetEq(const IloEnv env,
const IloAnySetVar var1,
const IloAnySetVar var2)
public IloConstraint IloSubsetEq(const IloEnv env,
const IloAnySet var1,
const IloAnySetVar var2)
public IloConstraint IloSubsetEq(const IloEnv env,
const IloAnySetVar var1,
const IloAnySet var2)
public IloConstraint IloSubsetEq(const IloEnv env,
const IloIntSetVar var1,
const IloIntSetVar var2)
public IloConstraint IloSubsetEq(const IloEnv var1,
const IloIntSet var1,
const IloIntSetVar var2)
public IloConstraint IloSubsetEq(const IloEnv var1,
const IloIntSetVar var1,
const IloIntSet var2)
```

**Description**

This function creates and returns a constraint (an instance of IloConstraint) for use in a model. That constraint forces var1 to be a subset of var2. (The set var1 may be equal to var2; every element of var1 is also an element of var2.)

In order for the constraint to take effect, you must add it to a model with the template IloAdd or the member function IloModel::add and extract the model for an algorithm with the member function IloAlgorithm::extract.
IloSum

Category: Global Function

Definition File: ilconcert/iloexpression.h

Synopsis:

public IloNumExprArg IloSum(const IloNumExprArray exprs)
public IloIntExprArg IloSum(const IloIntExprArray exprs)
public IloNum IloSum(const IloNumArray values)
public IloInt IloSum(const IloIntArray values)

Description:

These functions return a numeric value representing the sum of numeric values in the array vals, or an instance of IloNumExprArg, the internal building block of an expression, representing the sum of the variables in the arrays exprs or values.
IloTableConstraint

**Category**
Global Function

**Definition File**
ilconcert/ilotupleset.h

**Synopsis**

```cpp
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVarArray vars,
                                        const IloIntTupleSet set,
                                        IloBool compatible)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVar var1,
                                        const IloIntVar var2,
                                        const IloIntTupleSet set,
                                        IloBool compatible)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVar var1,
                                        const IloIntVar var2,
                                        const IloIntVar var3,
                                        const IloIntTupleSet set,
                                        IloBool compatible)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVarArray vars,
                                        const IloIntTernaryPredicate pred)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVarArray vars,
                                        const IloIntTernaryPredicate pred)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloIntVarArray vars,
                                        const IloIntBinaryPredicate pred)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloAnyVarArray vars,
                                        const IloAnyTupleSet set,
                                        IloBool compatible)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloAnyVarArray vars,
                                        const IloAnyTernaryPredicate pred)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloAnyVarArray vars,
                                        const IloAnyBinaryPredicate pred)
public IloConstraint IloTableConstraint(const IloEnv env,
                                        const IloAnyVarArray vars,
                                        const IloAnyTernaryPredicate pred)
```

ILOG Concert Technology 2.0 Reference Manual
const IloAnyVar y,
const IloAnyArray a,
const IloNumVar x)

Description

This function can be used to define simple constraints that are not predefined. It creates
and returns a constraint for use in an ILOG Concert Technology model. That constraint
is defined for all the constrained variables in the array vars or for the single
constrained variable y.

This kind of constraint is sometimes known as an element constraint.

The semantics of that generic constraint can be indicated in either one of several ways:

◆ by a predicate; in that case, the argument pred indicates that predicate;
◆ by the values that satisfy the constraint; in that case, the argument set indicates
the combinations of values that satisfy the constraint, and the argument
compatible must be IloTrue;
◆ by the values that do not satisfy the constraint; in that case, the argument set
indicates the unsatisfactory combinations of values, and the argument
compatible must be IloFalse;
◆ by making the constrained variable y equal to the element of the array a at the index
indicated by x. In other words, y=a[x];

The order of the constrained variables in the array vars is important because the same
order is respected in the predicate pred or the set. That is, IloTableConstraint
passes an array of values to the member function isTrue for a predicate or to the
member function isIn for a set, where the first such value is a value of vars[0], the
second is a value of vars[1], and in general, the ith value is a value of the constrained
variable vars[i].

To avoid exceptions, you must observe the following conditions:

◆ If the function is called with a predicate pred as an argument, the size of the array
of constrained variables must be three.
◆ The size of vars is must be the same as the size of the set.
IloThreeHalfPi

**Category**
Global Variable

**Definition File**
-ilconcert/ilosys.h

Concert Technology predefines conventional trigonometric constants to conform to IEEE 754 standards for quarter pi, half pi, pi, three-halves pi, and two pi.

```c
extern const IloNum IloThreeHalfPi;  // = 4.71238898038468986969
```
IloTimer

Category       Class
InheritancePath

Definition File ilconcert/iloenv.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloTimer (const IloEnv &amp; env)</td>
<td>This constructor creates a timer.</td>
</tr>
<tr>
<td>public IloTimer (IloEnvI *)</td>
<td>This constructor creates an instance of the class IloTimer</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloEnv getEnv()</td>
<td></td>
</tr>
<tr>
<td>public IloNum getTime()</td>
<td></td>
</tr>
<tr>
<td>public void reset()</td>
<td></td>
</tr>
<tr>
<td>public IloNum restart()</td>
<td></td>
</tr>
<tr>
<td>public IloNum start()</td>
<td></td>
</tr>
<tr>
<td>public IloNum stop()</td>
<td></td>
</tr>
</tbody>
</table>

Description
An instance of IloTimer represents a timer in a Concert Technology model. It works like a stop watch.

See Also
IloEnv

Constructors

public IloTimer (const IloEnv & env)
This constructor creates a timer.

public IloTimer (IloEnvI * env)
This constructor creates an instance of the class IloTimer

Methods

public IloEnv getEnv()
This member function returns the environment in which the invoking timer was constructed.

```java
public IloNum getTime()
```

This member function returns the accumulated time, in seconds, since one of these conditions:

- the first call of the member function `start` after construction of the invoking timer;
- the most recent call to the member function `restart`;
- a call to `reset`.

```java
public void reset()
```

This member function sets the elapsed time of the invoking timer to 0.0. It also stops the clock.

```java
public IloNum restart()
```

This member function returns the accumulated time, resets the invoking timer to 0.0, and starts the timer again. In other words, the member function `restart` is equivalent to the member function `reset` followed by `start`.

```java
public IloNum start()
```

This member function makes the invoking timer resume accumulating time. It returns the time accumulated so far.

```java
public IloNum stop()
```

This member function stops the invoking timer so that it no longer accumulates time.
**IloTwoPi**

**Category**
Global Variable

**Definition File**
ilinxconcert/ilosys.h

Concert Technology predefines conventional trigonometric constants to conform to IEEE 754 standards for quarter pi, half pi, pi, three-halves pi, and two pi.

```c
extern const IloNum IloTwoPi;        // = 6.28318530717958647692
```


IloTypedList

Category  
Class

InheritancePath

Definition File  
ilconcert/ilotypedlist.h

Description  
A list of extractables of the same type, with possibly a sibling and a first son list. It holds the extractables that are instances of a specific class: the instances of the sub-classes are in the son lists.
**IloTypedListManager**

<table>
<thead>
<tr>
<th>Category</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>InheritancePath</td>
<td></td>
</tr>
</tbody>
</table>

**Definition File**

ilconcert/ilotypedlist.h

**Description**

The class *IloTypedListManager* manages the *IloTypedList*. It classifies the extractables of a model to get a tree of lists of extractables grouped by type.

A model must have a pointer to an instance *IloTypedListManager* that is initialized to 0, and allocate such an instance when an iterator is created.

Creation and removal of extractable must check if an *IloTypedListManager* exists, and respectively call `add()` or `remove()` on it. The model may ask the *IloTypedListManager* to clear itself, which implies that all the extractables will be scanned again if an iterator is subsequently created.
**operator &&**

**Category**  |  Global Function
---|---
**Definition File**  |  ilconcert/ilomodel.h

**Synopsis**

```cpp
public IloAnd operator &&(const IloConstraint constraint1,
 const IloConstraint constraint2)
public IloAnd operator &&(const IloAnd constraint1,
 const IloConstraint constraint2)
public IloAnd operator &&(const IloConstraint constraint2,
 const IloAnd constraint1)
public IloAnd operator &&(const IloAnd constraint1,
 const IloAnd constraint2)
```

**Description**

This overloaded C++ operator creates a conjunctive constraint that represents the conjunction of its two parameters. The constraint can represent a conjunction of two constraints; of a constraint and another conjunction; or of two conjunctions. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator ***

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**

```c++
public IloNumLinExprTerm operator *(const IloNumVar x,
IloInt num)
public IloNumLinExprTerm operator *(IloInt num,
const IloNumVar x)
public IloNumLinExprTerm operator *(const IloIntVar x,
IloNum num)
public IloNumLinExprTerm operator *(IloNum num,
const IloIntVar x)
public IloIntLinExprTerm operator *(const IloIntVar x,
IloInt num)
public IloNumExprArg operator *(const IloNumExprArg x,
const IloNumExprArg y)
public IloNumExprArg operator *(const IloNumExprArg x,
IloNum y)
public IloNumExprArg operator *(IloNum x,
const IloNumExprArg y)
public IloIntExprArg operator *(const IloIntExprArg x,
const IloIntExprArg y)
public IloIntExprArg operator *(const IloIntExprArg x,
IloInt y)
```

**Description**  
This overloaded C++ operator returns an expression equal to the product of its arguments. Its arguments may be numeric values, numeric variables, or other expressions.
operator new

Category  Global Function
Definition File  ilconcert/iloenv.h
Synopsis  public void * operator new(size_t sz,
const IloEnv & env)
Description  ILOG Concert Technology offers this overloaded C++ new operator. This operator is
overloaded to allocate data on internal data structures associated with an invoking
environment (an instance of IloEnv). The memory used by objects allocated with this
overloaded operator is automatically reclaimed when you call the member function
IloEnv::end. As a developer, you must not delete objects allocated with this
operator because of this automatic freeing of memory.

In other words, you must not use the delete operator for objects allocated with this
overloaded new operator.

The use of this overloaded new operator is not obligatory in Concert Technology
applications. You will see examples of its use in the user’s manuals that accompany the
ILOG optimization products.
operator!

Category: Global Function

Definition File: ilconcert/ilomodel.h

Synopsis:

```cpp
public IloConstraint operator!(const IloConstraint constraint)
```

Description:

This overloaded C++ operator returns a constraint that is the negation of its argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator!=**

**Category**  
Global Function

**Definition File**  
ilconcert/iloany.h

**Synopsis**

```cpp
public IloConstraint operator!=(const IloAnyVar &var1, const IloAnyVar &var2)
public IloConstraint operator!=(const IloAnyVar &var1, IloAny val)
public IloConstraint operator!=(IloAny val, const IloAnyVar &var1)
public IloConstraint operator!=(const IloAnySetVar &var1, const IloAnySetVar &var2)
public IloConstraint operator!=(const IloAnySetVar &var1, const IloAnySet &set)
public IloConstraint operator!=(const IloAnySet &set, const IloAnySetVar &var1)
public IloDiff operator!=(IloNumExprArg arg1, IloNumExprArg arg2)
public IloDiff operator!=(IloNumExprArg arg, IloNum val)
public IloDiff operator!=(IloNum val, IloNumExprArg arg)
public IloConstraint operator!=(const IloIntSetVar &var1, const IloIntSetVar &var2)
public IloConstraint operator!=(const IloIntSetVar &var1, const IloIntSet &set)
public IloConstraint operator!=(const IloIntSet &set, const IloIntSetVar &var)
```

**Description**

This overloaded C++ operator constrains its two arguments to be unequal (that is, different from each other). In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator+**

**Category**  
Global Function

**Definition File**  
ilconcert/iloexpression.h

**Synopsis**

```c
public IloNumExprArg operator+(const IloNumExprArg x, const IloNumExprArg y)
public IloNumExprArg operator+(const IloNumExprArg x, IloNum y)
public IloNumExprArg operator+(IloNum x, const IloNumExprArg y)
public IloIntExprArg operator+(const IloIntExprArg x, const IloIntExprArg y)
public IloIntExprArg operator+(const IloIntExprArg x, IloInt y)
public IloIntExprArg operator+(IloInt x, const IloIntExprArg y)
```

**Description**

This overloaded C++ operator returns an expression equal to the sum of its arguments. Its arguments may be numeric values, numeric variables, or other expressions.
operator-

Category
Global Function

Definition File
ilconcert/iloexpression.h

Synopsis
public IloNumExprArg operator-(const IloNumExprArg x, const IloNumExprArg y)
public IloNumExprArg operator-(const IloNumExprArg x, IloNum y)
public IloNumExprArg operator-(IloNum x, const IloNumExprArg y)
public IloIntExprArg operator-(const IloIntExprArg x, const IloIntExprArg y)
public IloIntExprArg operator-(const IloIntExprArg x, IloInt y)
public IloIntExprArg operator-(IloInt x, const IloIntExprArg y)

Description
This overloaded C++ operator returns an expression equal to the difference of its arguments. Its arguments may be numeric values, numeric variables, or other expressions.
operator/

Category  Global Function

Definition File  ilconcert/iloexpression.h

Synopsis  
public IloNumExprArg operator/(const IloNumExprArg x, const IloNumExprArg y)
publlic IloNumExprArg operator/(const IloNumExprArg x, IloNum y)
public IloNumExprArg operator/(IloNum x, const IloNumExprArg y)

Description  
This overloaded C++ operator returns an expression equal to the quotient of its arguments. Its arguments may be numeric values or numeric variables. For integer division, use IloDiv.
operator<

Category                  Global Function
Definition File           ilconcert/ilolinear.h
Synopsis                   
public IloConstraint operator<(IloNumExprArg base, 
                             IloNumExprArg base2)
public IloConstraint operator<(IloNumExprArg base, 
                             IloNum val)
public IloConstraint operator<(IloNum val, 
                             const IloNumExprArg expr)
public IloConstraint operator<(IloIntExprArg base, 
                             IloIntExprArg base2)
public IloConstraint operator<(IloIntExprArg base, 
                             IloInt val)

Description          This overloaded C++ operator constrains its first argument to be strictly less than its second argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator<<**

**Category**  
Global Function

**Definition File**  
ilconcert/iloialg.h

**Synopsis**  
public ostream & operator<<(ostream & out,  
IloAlgorithm::Status st)  
public ostream & operator<<(ostream & out,  
const IloArray<X> & a)  
public ostream & operator<<(ostream & os,  
const IloRandom & r)  
public ostream & operator<<(ostream & os,  
const IloNumSet & set)  
public ostream & operator<<(ostream & stream,  
const IloSolution & solution)  
public ostream & operator<<(ostream & stream,  
const IloSolutionManip & fragment)  
public ostream & operator<<(ostream & os,  
const IloException & e)

**Description**  
This overloaded C++ operator directs output to an output stream.
operator<=

**Category** Global Function

**Definition File** ilconcert/ilolinear.h

**Synopsis**

public IloConstraint operator<=(IloNumExprArg base,
IloNumExprArg base2)

public IloRange operator<=(IloNumExprArg base,
IloNum val)

public IloRangeBase operator<=(IloNum val,
const IloNumExprArg expr)

public IloRange operator<=(const IloRangeBase base,
IloNum val)

**Description**

This overloaded C++ operator constrains its first argument to be less than or equal to its second argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator==**

**Category**  
Global Function

**Definition File**  
ilconcert/iloany.h

**Synopsis**

```cpp
class IloConstraint
{
public:
    // Equality comparison operator
    operator==(const IloAnyVar &var1, const IloAnyVar &var2);
    operator==(const IloAnyVar &var1, IloAny val);
    operator==(IloAny val, const IloAnyVar &var1);
    operator==(const IloAnySetVar &var1, const IloAnySetVar &var2);
    operator==(const IloAnySetVar &var1, const IloAnySet &set);
    operator==(const IloAnySet &set, const IloAnySetVar &var);
    operator==(const IloIntSetVar &var1, const IloIntSetVar &var2);
    operator==(const IloIntSetVar &var1, const IloIntSet &set);
    operator==(const IloIntSet &set, const IloIntSetVar &var);
    operator==(const IloNumExprArg &base, IloNumExprArg expr);
    operator==(IloNumExprArg base, IloNum val);
    operator==(IloNum val, IloNumExprArg eb);
    operator==(const IloIntSetVar &var1, const IloIntSetVar &var2);
    operator==(const IloIntSetVar &var1, const IloIntSet &set);
    operator==(const IloIntSet &set, const IloIntSetVar &var);
}
```

**Description**

This overloaded C++ operator constrains its two arguments to be equal. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
operator>

Category: Global Function

Definition File: ilconcert/ilolinear.h

Synopsis:

public IloConstraint operator>(IloNumExprArg base, IloNumExprArg base2)
public IloConstraint operator>(IloNumExprArg base, IloNum val)
public IloConstraint operator>(IloNum val, IloNumExprArg eb)
public IloConstraint operator>(IloIntExprArg base, IloIntExprArg base2)
public IloConstraint operator>(IloIntExprArg base, IloInt val)
public IloConstraint operator>(IloInt val, IloIntExprArg eb)

Description:

This overloaded C++ operator constrains its first argument to be strictly greater than its second argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
operator\geq

**Category**
Global Function

**Definition File**
ilconcert/ilolinear.h

**Synopsis**
public IloConstraint operator\geq(IloNumExprArg base, IloNumExprArg base2)
public IloRange operator\geq(IloNumExprArg expr, IloNum val)
public IloRange operator\geq(IloNum val, IloNumExprArg eb)

**Description**
This overloaded C++ operator constrains its first argument to be greater than or equal to its second argument. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.
**operator>>**

**Category**  
Global Function

**Definition File**  
ilconcert/iloenv.h

**Synopsis**  
public istream & operator>>(istream & in,  
IloNumArray & a)  
public istream & operator>>(istream & in,  
IloIntArray & a)  
public istream & operator>>(istream & stream,  
IloSolution & solution)  
public istream & operator>>(istream & stream,  
IloSolutionManip & fragment)

**Description**  
This overloaded C++ operator directs input to an input stream.
**operator||**

**Category**  
Global Function

**Definition File**  
ilconcert/ilomodel.h

**Synopsis**  
public IloOr operator||(const IloConstraint constraint1,  
const IloConstraint constraint2)

public IloOr operator||(const IloOr constraint1,  
const IloConstraint constraint2)

public IloOr operator||(const IloConstraint constraint2,  
const IloOr constraint1)

public IloOr operator||(const IloOr constraint1,  
const IloOr constraint2)

**Description**  
This overloaded C++ operator creates a disjunctive constraint that represents the disjunction of its two parameters. The constraint can represent a disjunction of two constraints; of a constraint and another disjunction; or of two disjunctions. In order to be taken into account, this constraint must be added to a model and extracted for an algorithm.

**Group optim.concert.extensions**

The ILOG Concert Extensions Library.

<table>
<thead>
<tr>
<th>Classes Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloColumnHeaderNotFoundException</td>
</tr>
<tr>
<td>IloCsvLine</td>
</tr>
<tr>
<td>IloCsvReader</td>
</tr>
<tr>
<td>IloCsvReader::LineIterator</td>
</tr>
<tr>
<td>IloCsvReaderParameterException</td>
</tr>
<tr>
<td>IloCsvReader::TableIterator</td>
</tr>
<tr>
<td>IloCsvTableReader</td>
</tr>
<tr>
<td>IloCsvTableReader::LineIterator</td>
</tr>
<tr>
<td>IloDuplicatedTableException</td>
</tr>
<tr>
<td>IloFieldNotFoundException</td>
</tr>
<tr>
<td>IloFileNotFoundException</td>
</tr>
<tr>
<td>IloIncorrectCsvReaderUseException</td>
</tr>
</tbody>
</table>
The ILOG Concert Extensions Library.

Global Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloDifference</td>
</tr>
<tr>
<td>IloDifference</td>
</tr>
<tr>
<td>IloIntersection</td>
</tr>
<tr>
<td>IloMax</td>
</tr>
<tr>
<td>IloMin</td>
</tr>
<tr>
<td>IloUnion</td>
</tr>
<tr>
<td>IloUnion</td>
</tr>
<tr>
<td>operator *</td>
</tr>
<tr>
<td>operator+</td>
</tr>
<tr>
<td>operator-</td>
</tr>
<tr>
<td>operator==</td>
</tr>
<tr>
<td>operator==</td>
</tr>
<tr>
<td>operator==</td>
</tr>
</tbody>
</table>
IloColumnHeaderNotFoundException

Category: Class

InheritancePath:

Definition File: ilconcert/ilocsvreader.h

Description: This exception is thrown by the member functions listed below if a header name that you use does not exist.
- IloCsvLine::getFloatByHeader
- IloCsvLine::getIntByHeader
- IloCsvLine::getStringByHeader
- IloCsvLine::getFloatByHeaderOrDefaultValue
- IloCsvLine::getIntByHeaderOrDefaultValue
- IloCsvLine::getStringByHeaderOrDefaultValue
- IloCsvReader::getPosition
- IloCsvTableReader::getPosition
# IloCsvLine

## Category
Class

## Inheritance Path

## Definition File
`ilconcert/ilocsvreader.h`

## Constructor Summary

<table>
<thead>
<tr>
<th>Public Constructor</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IloCsvLine()</code></td>
<td>IloCsvLine()</td>
</tr>
<tr>
<td><code>IloCsvLine(IloCsvLineI *)</code></td>
<td>IloCsvLine(IloCsvLineI *)</td>
</tr>
<tr>
<td><code>IloCsvLine(const IloCsvLine &amp;)</code></td>
<td>IloCsvLine(const IloCsvLine &amp;)</td>
</tr>
</tbody>
</table>

## Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy(const IloCsvLine)</code></td>
<td>copy(const IloCsvLine)</td>
</tr>
<tr>
<td><code>emptyFieldByHeader(const char *)</code></td>
<td>emptyFieldByHeader(const char *)</td>
</tr>
<tr>
<td><code>emptyFieldByPosition(IloInt)</code></td>
<td>emptyFieldByPosition(IloInt)</td>
</tr>
<tr>
<td><code>end()</code></td>
<td>end()</td>
</tr>
<tr>
<td><code>getFloatByHeader(const char *)</code></td>
<td>IloCsvLine::getFloatByHeader(const char *)</td>
</tr>
<tr>
<td><code>getFloatByHeaderOrDefaultValue(const char *, IloNum)</code></td>
<td>IloCsvLine::getFloatByHeaderOrDefaultValue(const char *, IloNum)</td>
</tr>
<tr>
<td><code>getFloatByPosition(IloInt)</code></td>
<td>IloCsvLine::getFloatByPosition(IloInt)</td>
</tr>
<tr>
<td><code>getFloatByPositionOrDefaultValue(IloInt, IloNum)</code></td>
<td>IloCsvLine::getFloatByPositionOrDefaultValue(IloInt, IloNum)</td>
</tr>
<tr>
<td><code>getImpl()</code></td>
<td>getImpl()</td>
</tr>
<tr>
<td><code>getIntByHeader(const char *)</code></td>
<td>IloCsvLine::getIntByHeader(const char *)</td>
</tr>
<tr>
<td><code>getIntByHeaderOrDefaultValue(const char *, IloInt)</code></td>
<td>IloCsvLine::getIntByHeaderOrDefaultValue(const char *, IloInt)</td>
</tr>
<tr>
<td><code>getIntByPosition(IloInt)</code></td>
<td>IloCsvLine::getIntByPosition(IloInt)</td>
</tr>
<tr>
<td><code>getIntByPositionOrDefaultValue(IloInt, IloInt)</code></td>
<td>IloCsvLine::getIntByPositionOrDefaultValue(IloInt, IloInt)</td>
</tr>
<tr>
<td><code>getLineNumber()</code></td>
<td>getLineNumber()</td>
</tr>
<tr>
<td><code>getNumberOfFields()</code></td>
<td>getNumberOfFields()</td>
</tr>
</tbody>
</table>
An instance of IloCsvLine represents a line in a csv file.

**Constructors**

**public IloCsvLine()**

This constructor creates a csv line object whose handle pointer is null. This object must be assigned before it can be used.

**public IloCsvLine(IloCsvLineI * impl)**

This constructor creates a handle object (an instance of IloCsvLine) from a pointer to an implementation object (an instance of the class IloCsvLineI).

**public IloCsvLine(const IloCsvLine & csvLine)**

This copy constructor creates a handle from a reference to a csv line object. The csv line object and csvLine both point to the same implementation object.

**Methods**

**public void copy(const IloCsvLine)**

This member function returns the real number of the invoking csv line in the data file.

**public IloBool emptyFieldByHeader(const char * name)**

This member function returns IloTrue if the field denoted by the string name in the invoking csv line is empty. Otherwise, it returns IloFalse.

**public IloBool emptyFieldByPosition(IloInt i)**

This member function returns IloTrue if the field denoted by i in the invoking csv line is empty. Otherwise, it returns IloFalse.

**public void end()**

This member function deallocates the memory used by the csv line. If you no longer need a csv line, you can call this member function to reduce memory consumption.

**public IloNum getFloatByHeader(const char * name)**

This member function returns the float contained in the field name in the invoking csv line.
If you have a loop in which you are getting a string, integer, or float by header on several lines with the same header name, it is better for performance to get the position of the header named name using the member function IloCsvReader::getPosition(name) than using getFloatByPosition (position of name in the header line).

public IloNum getFloatByHeaderOrDefaultValue (const char * name,
                                             IloNum defaultValue)

This member function returns the float contained in the field name in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.

public IloNum getFloatByPosition (IloInt i)

This member function returns the float contained in the field i in the invoking csv line.

public IloNum getFloatByPositionOrDefaultValue (IloInt i,
                                               IloNum defaultValue)

This member function returns the float contained in the field i in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.

public IloCsvLineI * getImpl ()

This member function returns a pointer to the implementation object corresponding to the invoking csv line.

public IloInt getIntByHeader (const char * name)

This member function returns the integer contained in the field name in the invoking csv line.

If you have a loop in which you are getting a string, integer, or float by header on several lines with the same header name, it is better for performance to get the position of the header named name using the member function IloCsvReader::getPosition(name) than using getIntByPosition (position of name in the header line).

public IloInt getIntByHeaderOrDefaultValue (const char * name,
                                            IloInt defaultValue)

This member function returns the integer contained in the field name in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.

public IloInt getIntByPosition (IloInt i)

This member function returns the integer contained in the field i in the invoking csv line.

public IloInt getIntByPositionOrDefaultValue (IloInt i,
                                               IloInt defaultValue)

This member function returns the integer contained in the field i in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.
public IloInt getLineNumber()

This member function returns the real number of the invoking csv line in the data file.

public IloInt getNumberOfFields()

This member function returns the number of fields in the line.

public char * getStringByHeader(const char * name)

This member function returns a reference to the string contained in the field name in the invoking csv line.

If you have a loop in which you are getting a string, integer, or float by header on several lines with the same header name, it is better for performance to get the position of the header named name using the member function IloCsvReader::getPosition(name) than using getStringByPosition (position of name in the header line).

public char * getStringByHeaderOrDefaultValue(const char * name, const char * defaultValue)

This member function returns the string contained in the field name in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.

public char * getStringByPosition(IloInt i)

This member function returns a reference to the string contained in the field number i in the invoking csv line.

public char * getStringByPositionOrDefaultValue(IloInt i, const char * defaultValue)

This member function returns the string contained in the field i in the invoking csv line if this field contains a value. Otherwise, it returns defaultValue.

public void operator=(const IloCsvLine & csvLine)

This operator assigns an address to the handle pointer of the invoking csv line. This address is the location of the implementation object of the argument csvLine.

After execution of this operator, the invoking csv line and csvLine both point to the same implementation object.

public IloBool printValueOfKeys()

This member function prints the values of the keys fields in this line.
**IloCsvReader**

**Category**  
Class

**Inheritance Path**  
IloCsvReader

**Definition File**  
ilconcert/ilocsvreader.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloCsvReader()</td>
<td>Constructor</td>
</tr>
<tr>
<td>IloCsvReader(IloCsvReaderI*)</td>
<td>Constructor</td>
</tr>
<tr>
<td>IloCsvReader(const IloCsvReader&amp;)</td>
<td>Constructor</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void end()</td>
<td></td>
</tr>
<tr>
<td>IloNum getCsvFormat()</td>
<td></td>
</tr>
<tr>
<td>IloCsvLine getCurrentLine()</td>
<td></td>
</tr>
<tr>
<td>IloEnv getEnv()</td>
<td></td>
</tr>
<tr>
<td>IloNum getFileVersion()</td>
<td></td>
</tr>
<tr>
<td>IloCsvReaderI* getImpl()</td>
<td></td>
</tr>
<tr>
<td>IloCsvLine getKeyByNumber(int, const char *, ...)</td>
<td></td>
</tr>
<tr>
<td>IloCsvLine getLineByNumber(int)</td>
<td></td>
</tr>
<tr>
<td>int getNumberOfColumns()</td>
<td></td>
</tr>
<tr>
<td>int getNumberOfItems()</td>
<td></td>
</tr>
<tr>
<td>int getNumberOfKeys()</td>
<td></td>
</tr>
<tr>
<td>int getNumberOfTables()</td>
<td></td>
</tr>
<tr>
<td>int IloCsvReader::getPosition(const char *)</td>
<td></td>
</tr>
<tr>
<td>IloCsvTableReader getReaderForUniqueTableFile()</td>
<td></td>
</tr>
<tr>
<td>const char * getRequiredBy()</td>
<td></td>
</tr>
<tr>
<td>IloCsvTableReader getTable()</td>
<td></td>
</tr>
<tr>
<td>IloCsvTableReader getTableByName(const char *)</td>
<td></td>
</tr>
<tr>
<td>IloCsvTableReader getTableByNumber(IloInt)</td>
<td></td>
</tr>
</tbody>
</table>
Description

An instance of IloCsvReader reads a csv file of a specified format.

The csv file can be a multitable or a single table file. Empty lines and commented lines are allowed everywhere in the file.

Format of multitable files

The first column of the table must contain the name of the table.

Each table can begin with a line containing column headers, the first field of this line must have this format: tableName|NAMES

The keys can be specified in the data file by adding a line at the beginning of the table. This line is formatted as follows:

◆ the first field is tableName|KEYS
◆ the other fields have the value 1 if the corresponding column is a key for the table; if not they have the value 0.

If this line doesn’t exist, all columns form a key. If you need to get a line having a specific value for a field, you must add the key line in which you specify that this field is a key for the table.

Any line containing '|' in its first field is ignored by the reader.

A table can be split in several parts in the file (for example, you have a part of table TA, then table TB, then the end of table TA).

Example

NODES|NAMES,node_type,node_name,xcoord,ycoord
NODES|KEYS,1,1,0,0
NODES,1,node1,0,1
NODES,1,node2,0,2
NODES,2,node1,0,4
Format of single table files

The line containing the column headers, if it exists, must have a first field of the following format: **Field|NAMES**.

Table keys can be specified by adding a line at the beginning of the table. This line must have a first field with this format: **tableName|KEYS**. If this line doesn't exist, all columns form a key.

**Example**

```
Field|NAMES,nodeName,xCoord,yCoord
Field|KEYS,1,0,0
node1,0,1
node2,0,2
```

**Constructors**

- **public IloCsvReader()**
  
  This constructor creates a csv reader object whose handle pointer is null. This object must be assigned before it can be used.

- **public IloCsvReader(IloCsvReaderI * impl)**
  
  This constructor creates a handle object (an instance of *IloCsvReader*) from a pointer to an implementation object (an instance of the class *IloCsvReaderI*).

- **public IloCsvReader(const IloCsvReader & csv)**
  
  This copy constructor creates a handle from a reference to a csv reader object. Both the csv reader object and csv point to the same implementation object.

**Methods**

- **public void end()**
  
  This member function deallocates the memory used by the csv reader. If you no longer need a csv reader, you can reduce memory consumption by calling this member function.

- **public IloNum getCsvFormat()**
  
  This member function returns the format of the csv data file. This format is identified in the data file by **ILOG_CSV_FORMAT**.

**Example**

```
ILOG_CSV_FORMAT;1
getCsvFormat() returns 1.
```
public IloCsvLine getCurrentLine()

This member function returns the last line read by getLineByKey or getLineByNumber.

**Note:** This member function can be used only if the parameter isMultiTable has the value IloTrue.

public IloEnv getEnv()

This member function returns the environment object corresponding to the invoking csv reader.

public IloNum getFileVersion()

This member function returns the version of the csv data file. This information is identified in the data file by ILOG_DATA_SCHEMA.

**Example**

```
ILOG_DATA_SCHEMA;PROJECTNAME;0.9
```

getFileVersion() returns 0.9.

**Note:** This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloCsvReaderI * getImpl()

This member function returns a pointer to the implementation object corresponding to the invoking csv reader.

public IloCsvLine getLineByKey(IloInt numberOfKeys, const char *, ...)

This member function takes numberOfKeys as parameters; these parameters are used as one key to identify a line. It returns an instance of IloCsvLine representing the
line having (key1, key2, ...) in the data file. If the number of keys specified is less than the number of keys in the table, this member function throws an exception. Each time getLineByNumber or getLineByKey is called, the previous line read by one of these methods is deleted.

Note: This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloCsvLine getLineByNumber(IloInt i)
This member function returns an instance of IloCsvLine representing the line numbered i in the data file. If i does not exist, this member function throws an exception. Each time getLineByNumber or getLineByKey is called, the previous line read by one of these methods is deleted.

Note: This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloInt getNumberOfColumns()
This member function returns the number of columns in the table. If the first column contains the name of the table it is ignored.

Note: This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloInt getNumberOfItems()
This member function returns the number of lines of the table excluding blank lines, commented lines, and the header line.

Note: This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloInt getNumberOfKeys()
This member function returns the number of keys for the table.
public IloInt getNumberOfTables()

This member function returns the number of tables in the data file.

public IloInt getPosition(const char * headingName)

This member function returns the position (column number) of the headingName in the file.

**Note:** This member function can be used only if the parameter isMultiTable has the value IloFalse.

public IloCsvTableReader getReaderForUniqueTableFile()

This member function returns an IloCsvTableReader for the unique table contained in the csv data file.

**Note:** This member function can be used only if the parameter isMultiTable has the value IloFalse.

public const char * getRequiredBy()

This member function returns the name of the project that uses the csv data file. This information is identified in the data file by ILOG_DATA_SCHEMA.

**Example**

```
ILOG_DATA_SCHEMA;PROJECTNAME;0.9
```

getRequiredBy() returns PROJECTNAME.

**Note:** This member function can be used only if the parameter isMultiTable has the value IloTrue.

public IloCsvTableReader getTable()
This member function returns an instance of `IloCsvTableReader` representing the unique table in the data file.

```
public IloCsvTableReader getTableByName(const char * name)
```

This member function returns an instance of `IloCsvTableReader` representing the table named `name` in the data file.

**Note:** This member function can be used only if the parameter `isMultiTable` has the value `IloFalse`.

```
public IloCsvTableReader getTableByNumber(IloInt i)
```

This member function returns an instance of `IloCsvTableReader` representing the table numbered `i` in the data file.

**Note:** This member function can be used only if the parameter `isMultiTable` has the value `IloTrue`.

```
public IloBool isHeadingExists(const char * headingName)
```

This member function returns `IloTrue` if the column header `headingName` exists. Otherwise, it returns `IloFalse`.

**Note:** This member function can be used only if the parameter `isMultiTable` has the value `IloTrue`.

```
public void operator=(const IloCsvReader & csv)
```

This operator assigns an address to the handle pointer of the invoking `csv` reader. This address is the location of the implementation object of the argument `csv`.

After execution of this operator, both the invoking `csv` reader and `csv` point to the same implementation object.

```
public IloBool printKeys()
```

Note: This member function can be used only if the parameter `isMultiTable` has the value `IloFalse`.
This member function prints the column header of keys if the header exists. Otherwise, it prints the column numbers of keys.

**Note:** This member function can be used only if the parameter `isMultiTable` has the value `IloFalse`. 
IloCsvReader::LineIterator

Category          Inner Class
InheritancePath   
Definition File   ilconcert/ilocsvreader.h

Constructor Summary
<table>
<thead>
<tr>
<th>Public</th>
<th>LineIterator()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>LineIterator(IloCsvReader)</td>
</tr>
</tbody>
</table>

Method Summary
| Public IloBool ok() |
| Public IloCsvLine operator*() |
| Public LineIterator & operator++() |

Description
LineIterator is a nested class of the class IloCsvReader. It is to be used only with csv reader objects built to read a unique-table data file.

IloCsvReader::LineIterator allows you to step through all the lines of the csv data file (except blank lines and commented lines) on which the csv reader was created.

Constructors
public LineIterator()
This constructor creates an empty LineIterator object. This object must be assigned before it can be used.

public LineIterator(IloCsvReader csv)
This constructor creates an iterator to traverse all the lines in the csv data file on which the csv reader csv was created.

The iterator does not traverse blank lines and commented lines.

Methods
public IloBool ok()

This member function returns IloTrue if the current position of the iterator is a valid one.

It returns IloFalse if the iterator reaches the end of the table.
public IloCsvLine operator *()

This operator returns the current instance of IloCsvLine (representing the current line in the csv file); the one to which the invoking iterator points.

public LineIterator & operator++()

This left-increment operator shifts the current position of the iterator to the next instance of IloCsvLine representing the next line in the file.
IloCsvReaderParameterException

Category: Class

InheritancePath: ilconcert/iilocsvreader.h

Definition File: ilconcert/iilocsvreader.h

Description: This exception is thrown in the constructor of the csv reader if the argument values used in the csv reader constructor are incorrect.
**IloCsvReader::TableIterator**

**Category**  Inner Class

**InheritancePath**

**Definition File**  ilconcert/ilocsvreader.h

---

**Constructor Summary**

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableIterator(IloCsvReader)</td>
<td>TableIterator(IloCsvReader)</td>
</tr>
</tbody>
</table>

---

**Method Summary**

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloBool</td>
<td>ok()</td>
</tr>
<tr>
<td>IloCsvTableReader</td>
<td>operator *()</td>
</tr>
<tr>
<td>TableIterator &amp;</td>
<td>operator++()</td>
</tr>
</tbody>
</table>

---

**Description**

TableIterator is a nested class of the class IloCsvReader. It is to be used only for multitable files.

IloCsvReader::TableIterator allows you to step through all the tables of the multitable csv data file on which the csv reader was created.

---

**Constructors**

public **TableIterator**(IloCsvReader csv)

This constructor creates an iterator to traverse all the tables in the csv data file on which the csv reader csv was created.

---

**Methods**

public **IloBool** **ok**()

This member function returns IloTrue if the current position of the iterator is a valid one.

It returns IloFalse if the iterator reaches the end of the table.

public **IloCsvTableReader** **operator ***()
This operator returns the current instance of `IloCsvTable` (representing the current table in the csv file); the one to which the invoking iterator points.

```java
public TableIterator & operator++()
```

This left-increment operator shifts the current position of the iterator to the next instance of `IloCsvTableReader` representing the next line in the file.
IloCsvTableReader

Category  Class

InheritancePath

Definition File  ilconcert/ilocsvreader.h

## Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloCsvTableReader()</td>
<td></td>
</tr>
<tr>
<td>public IloCsvTableReader(IloCsvTableReaderI *)</td>
<td></td>
</tr>
<tr>
<td>public IloCsvTableReader(const IloCsvTableReader &amp;)</td>
<td></td>
</tr>
<tr>
<td>public IloCsvTableReader(IloCsvReaderI *, const char *)</td>
<td></td>
</tr>
</tbody>
</table>

## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void end()</td>
<td></td>
</tr>
<tr>
<td>public IloCsvLine getCurrentLine()</td>
<td></td>
</tr>
<tr>
<td>public IloEnv getEnv()</td>
<td></td>
</tr>
<tr>
<td>public IloCsvTableReaderI * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public IloCsvLine getLineByKey(IloInt, const char *, ...)</td>
<td></td>
</tr>
<tr>
<td>public IloCsvLine getLineByNumber(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public const char * getNameOfTable()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getNumberOfColumns()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getNumberOfItems()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getNumberOfKeys()</td>
<td></td>
</tr>
<tr>
<td>public IloInt IloCsvTableReader::getPosition(const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloBool isHeadingExists(const char *)</td>
<td></td>
</tr>
<tr>
<td>public void operator=(const IloCsvTableReader &amp;)</td>
<td></td>
</tr>
<tr>
<td>public IloBool printKeys()</td>
<td></td>
</tr>
</tbody>
</table>
An instance of `IloCsvTableReader` is used to read a csv table with a specified format.

An instance is built using a pointer to an implementation class of `IloCsvReader`, which must be created first.

### Constructors

- **public `IloCsvTableReader()`**
  
  This constructor creates a table csv reader object whose handle pointer is null. This object must be assigned before it can be used.

- **public `IloCsvTableReader(IloCsvTableReaderI * impl)`**
  
  This constructor creates a handle object (an instance of `IloCsvReader`) from a pointer to an implementation object (an instance of the class `IloCsvReaderI`).

- **public `IloCsvTableReader(const IloCsvTableReader & csv)`**
  
  This copy constructor creates a handle from a reference to a table csv reader object. The table csv reader object and `csv` both point to the same implementation object.

- **public `IloCsvTableReader(IloCsvReaderI *, const char * name)`**
  
  This constructor creates a table csv reader object using the implementation class of a csv reader `csvimpl`. The second parameter is the name of the table.

### Methods

- **public void `end()`**
  
  This member function deallocates the memory used by the table csv reader.

  If you no longer need the table csv reader, calling this member function can reduce memory consumption.

- **public `IloCsvLine getCurrentLine()`**
  
  This member function returns the last line read using `getLineByKey` or `getLineByNumber`.

- **public `IloEnv getEnv()`**
  
  This member function returns the environment object corresponding to the invoking table csv reader.

- **public `IloCsvTableReaderI * getImpl()`**
This member function returns a pointer to the implementation object corresponding to the invoking table csv reader.

```java
public IloCsvLine getlineByKey(IloInt numberOfKeys,
                                 const char *,
                                 ...)
```

This member function takes numberOfKeys as parameters. These parameters are used as one key to identify a line. If the specified number of keys is less than the number of keys of the table, this member function throws an exception.

Otherwise, it returns an instance of IloCsvLine representing the line having (key1, key2, ...) in the data file.

```java
public IloCsvLine getlineByNumber(IloInt i)
```

This member function returns an instance of IloCsvLine representing the line number i in the data file if it exists. Otherwise, it throws an exception.

Each time getlineByNumber or getlineByKey is called, the previous line read by one of those methods is deleted.

```java
public const char * getNameOfTable()
```

This member function returns the name of the table.

```java
public IloInt getNumberOfColumns()
```

This member function returns the number of columns in the table. If the first column contains the name of the table, it is ignored.

```java
public IloInt getNumberOfItems()
```

This member function returns the number of lines of the table excluding blank lines, commented lines, and the header line.

---

**Note:** This member function can be used only if the parameter isMultiTable has the value IloFalse.

```java
public IloInt getNumberOfKeys()
```

This member function returns the number of keys in the table.

```java
public IloInt getPosition(const char *)
```

This member function returns the position (column number) of headingName in the table.

```java
public IloBool isHeadingExists(const char * headingName)
```

This member function returns IloTrue if the column header named headingName exists. Otherwise, it returns IloFalse.
public void operator=(const IloCsvTableReader & csv)

This operator assigns an address to the handle pointer of the invoking table csv reader.
This address is the location of the implementation object of the argument csv.
After execution of this operator, the invoking table csv reader and csv both point to the same implementation object.

public IloBool printKeys()

This member function prints the column headers of keys if they exist. Otherwise, it prints the column numbers of keys.
IloCsvTableReader::LineIterator

Category: Inner Class

InheritancePath:

Definition File: ilconcert/ilocsviewer.h

Constructor Summary:

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>LineIterator()</td>
</tr>
<tr>
<td>public</td>
<td>LineIterator(IloCsvTableReader)</td>
</tr>
</tbody>
</table>

Method Summary:

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool</td>
<td>ok()</td>
</tr>
<tr>
<td>public IloCsvLine</td>
<td>operator *()</td>
</tr>
<tr>
<td>public LineIterator &amp;</td>
<td>operator++()</td>
</tr>
</tbody>
</table>

Description:

LineIterator is a nested class of the class IloCsvTableReader. It allows you to step through all the lines of a table from a csv data file (except blank lines and commented lines) on which the table csv reader was created.

Constructors:

public LineIterator()

This constructor creates an empty LineIterator object. This object must be assigned before it can be used.

public LineIterator(IloCsvTableReader csv)

This constructor creates an iterator to traverse all the lines in the table csv data file on which the csv reader csv was created.

The iterator does not traverse blank lines and commented lines.

Methods:

public IloBool ok()
This member function returns \texttt{IloTrue} if the current position of the iterator is a valid one.

It returns \texttt{IloFalse} if the iterator reaches the end of the table.

\texttt{public IloCsvLine \operator\*()}

This operator returns the current instance of \texttt{IloCsvLine} (representing the current line in the csv file); the one to which the invoking iterator points.

\texttt{public LineIterator \& \operator++()}

This left-increment operator shifts the current position of the iterator to the next instance of \texttt{IloCsvLine} representing the next line in the file.
**IloDifference**

**Category**   | Global Function
**Definition File** | ilconcert/ilointervals.h
**Synopsis** | public IloIntervalList IloDifference(const IloIntervalList intervals1, const IloIntervalList intervals2)

**Description** | This operator creates and returns an interval list equal to the difference between the interval list `intervals1` and the interval list `intervals2`. The arguments `intervals1` and `intervals2` must be defined on the same interval. The resulting interval list is defined on the same interval as the arguments. See also: `IloIntervalList`. 


**IloDifference**

**Category**  
Global Function

**Definition File**  
ilconcert/ilosetfunc.h

**Synopsis**  
public IloNumToAnySetStepFunction IloDifference(const IloNumToAnySetStepFunction f1, const IloNumToAnySetStepFunction f2)

**Description**  
This operator creates and returns a function equal to the difference between the functions f1 and f2. The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToAnySetStepFunction.
IloDuplicatedTableException

Category Class

InheritancePath

Definition File ilconcert/ilocsvreader.h

Description This exception is thrown in the constructor of the csv reader if you read a multitable file in which two tables have the same name but table splitting has not been specified.
IloFieldNotFoundException

Category: Class

InheritancePath:

Definition File: ilconcert/ilocsvreader.h

Description: This exception is thrown by the IloCsvLine methods listed below if the corresponding field does not exist.

- `getFloatByPosition`
- `getIntByPosition`
- `getStringByPosition`
- `IloCsvLine::getFloatByHeader`
- `IloCsvLine::getIntByHeader`
- `IloCsvLine::getStringByHeader`
- `getFloatByPositionOrDefaultValue`
- `getIntByPositionOrDefaultValue`
- `getStringByPositionOrDefaultValue`
- `IloCsvLine::getFloatByHeaderOrDefaultValue`
- `IloCsvLine::getIntByHeaderOrDefaultValue`
- `IloCsvLine::getStringByHeaderOrDefaultValue`
## IloFileNotFoundException

**Category**  
Class

**InheritancePath**

**Definition File**  
ilconcert/ilocsvreader.h

**Description**  
This exception is thrown in the constructor of the csv reader if a specified file is not found.
IloIncorrectCsvReaderUseException

Category       Class
InheritancePath

Definition File ilconcert/ilocsvreader.h

Description
This exception is thrown in the following member functions if you call them from a reader built as a multitable csv reader.

- getLineByNumber
- getLineByKey
- getNumberOfItems
- getNumberOfColumns
- getNumberOfKeys
- getReaderForUniqueTableFile
- getTable
- isHeadingExists
- printKeys

This exception is throw in the following member functions if you call them from a reader built as a unique table csv reader.

- getCsvFormat
- getFileVersion
- getTableByName
- getTableByNumber
- getRequiredBy
IloIntersection

Category: Global Function

Definition File: ilconcert/ilosetfunc.h

Synopsis:
public IloNumToAnySetStepFunction IloIntersection(const IloNumToAnySetStepFunction f1, const IloNumToAnySetStepFunction f2)

Description:
This operator creates and returns a function equal to the intersection between the functions f1 and f2. The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToAnySetStepFunction.
**IloIntervalList**

**Category**
Class

**InheritancePath**

**Definition File**
ilconcert/ilointervals.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntervalList(const IloEnv, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntervalList(const IloEnv, const IloNumArray, const IloNumArray, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>addInterval(IloNum, IloNum, IloNum)</td>
</tr>
<tr>
<td>public void</td>
<td>addPeriodicInterval(IloNum, IloNum, IloNum, IloNum, IloNum)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>contains(const IloIntervalList)</td>
</tr>
<tr>
<td>public void</td>
<td>dilate(IloNum)</td>
</tr>
<tr>
<td>public void</td>
<td>empty()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getDefinitionIntervalMax()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getDefinitionIntervalMin()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isEmpty()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>isKeptOpen()</td>
</tr>
<tr>
<td>public void</td>
<td>keepOpen(IloBool)</td>
</tr>
</tbody>
</table>
An instance of the class `IloIntervalList` represents a list of non-overlapping intervals. Each interval \([\text{timeMin}, \text{timeMax})\) from the list is associated with a numerical type.

Note that if \(n\) is the number of intervals in the list, the random access to a given interval (see the member functions `addInterval`, `contains`, and `removeInterval`) has a worst-case complexity in \(O(\log(n))\).

Furthermore, when two consecutive intervals of the list have the same types, these intervals are merged so that the list is always represented with the minimal number of intervals.

### See Also

- `IloIntervalListCursor`, `IloUnion`, `IloDifference`

### Constructors

```java
public IloIntervalList(const IloEnv env,
                        IloNum min,
                        IloNum max,
                        const char * name)
```

This constructor creates a new instance of `IloIntervalList` and adds it to the set of interval lists managed in the given environment. The arguments `min` and `max` respectively represent the origin and the horizon of the interval list. The new interval list does not contain any intervals.

```java
public IloIntervalList(const IloEnv env,
                        const IloNumArray times,
                        const IloNumArray types,
                        const char * name)
```

This constructor creates an interval list whose intervals are defined by the two arrays `times` and `types`. More precisely, if \(n\) is the size of array `times`, then the size of array `types` must be \(n-1\) and the following contiguous intervals are created on the
interval list: \([\text{times}[i], \text{times}[i+1))\) with type \(\text{types}[i]\) for all \(i \in [0, n-1]\).

Methods

- **public void `addInterval(IloNum start, IloNum end, IloNum type)`**
  
  This member function adds an interval of type `type` to the invoking interval list. The start time and end time of that newly added interval are set to `start` and `end`. By default, the type of the interval is 0. Adding a new interval that overlaps with an already existing interval of a different type will override the existing type on the intersection.

- **public void `addPeriodicInterval(IloNum start, IloNum duration, IloNum period, IloNum end, IloNum type)`**
  
  This member function adds a set of intervals to the invoking interval list. For every \(i \geq 0\) such that \(\text{start} + i \times \text{period} < \text{end}\), an interval of \([\text{start} + i \times \text{period}, \text{start} + \text{duration} + i \times \text{period})\) is added. By default, the type of these intervals is 0. Adding a new interval that overlaps with an already existing interval of a different type will override the existing type on the intersection.

- **public IloBool `contains(const IloIntervalList intervals)`**
  
  This member function returns `IloTrue` if and only if each interval of `intervals` is included in an interval of the invoking interval list, regardless of interval type.

- **public void `dilate(IloNum k)`**
  
  This member function multiplies by `k` the scale of times for the invoking interval list. `k` must be a positive number.

- **public void `empty()`**
  
  This member function removes all the intervals from the invoking interval list.

- **public IloNum `getDefinitionIntervalMax()`**
  
  This member function returns the right most point (horizon) of the definition interval of the invoking interval list.

- **public IloNum `getDefinitionIntervalMin()`**
  
  This member function returns the left most point (origin) of the definition interval of the invoking interval list.

- **public IloBool `isEmpty()`**
  
  This member function returns `IloTrue` if and only if the invoking interval list is empty.

- **public IloBool `isKeptOpen()`**
  
  This member function returns `IloTrue` if and only if the invoking interval list is kept open.
This member function returns IloTrue if the interval list must be kept open. Otherwise, it returns IloFalse.

```java
public void keepOpen(IloBool val)
```

If the argument `val` is equal to IloTrue, this member function states that the invoking interval list must be kept open during the search for a solution to the problem. It means that additional intervals may be added during the search. Otherwise, if the argument `val` is equal to IloFalse, it states that all the intervals of the invoking interval list will be defined in the model before starting to solve the problem. By default, it is supposed that all the intervals of the invoking interval list are defined in the model before starting to solve the problem.

```java
public void removeInterval(IloNum start, IloNum end)
```

This member function removes all intervals on the invoking interval list between `start` and `end`. If `start` is placed inside an interval `[start1, end1)`, that is, `start1 < start < end1`, this results in an interval `[start1, start)`. If `end` is placed inside an interval `[start2, end2)` this results in an interval `[end, end2)`.

```java
public void removeIntervalOnDuration(IloNum start, IloNum duration)
```

This member function removes all intervals on the invoking resource between `start` and `start+duration`.

```java
public void removePeriodicInterval(IloNum start, IloNum duration, IloNum period, IloNum end)
```

This member function removes intervals from the invoking interval list. More precisely, for every `i >= 0` such that `start + i * period < end`, this function removes all intervals between `start + i * period` and `start + duration + i * period`.

```java
public void setDifference(const IloIntervalList intervals)
```

This member function removes from the invoking interval list all the intervals contained in the interval list `intervals`. The definition interval of the invoking interval list is not changed.

```java
public void setPeriodic(const IloIntervalList intervals, IloNum x0, IloNum n)
```

This member function initializes the invoking interval list as an interval list that repeats the interval list intervals `n` times after `x0`.

```java
public void setUnion(const IloIntervalList intervals)
```
This member function sets the invoking interval list to be the union between the current interval list and the interval list `intervals`. An instance of `IloException` is thrown if two intervals with different types overlap. The definition interval of the invoking interval list is set to the union between the current definition interval and the definition interval of `intervals`.

```java
public void shift(IloNum dx)
```

This member function shifts the intervals of the invoking interval list from `dx` to the right if `dx > 0` or from `-dx` to the left if `dx < 0`. It has no effect if `dx = 0`. 
**IloIntervalListCursor**

**Category**  
Class

**InheritancePath**

**Definition File**  
ilconcert/iloIntervals.h

**Constructor Summary**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloIntervalListCursor(const IloIntervalList)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntervalListCursor(const IloIntervalList, IloNum)</td>
</tr>
<tr>
<td>public</td>
<td>IloIntervalListCursor(const IloIntervalListCursor &amp;)</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum</td>
<td>getEnd()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getStart()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getType()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>ok()</td>
</tr>
<tr>
<td>public void</td>
<td>operator++()</td>
</tr>
<tr>
<td>public void</td>
<td>operator--()</td>
</tr>
<tr>
<td>public void</td>
<td>operator=(const IloIntervalListCursor &amp;)</td>
</tr>
<tr>
<td>public void</td>
<td>seek(IloNum)</td>
</tr>
</tbody>
</table>

**Description**  
An instance of the class `IloIntervalListCursor` allows you to inspect the intervals of an `IloIntervalList`. Cursors are intended to iterate forward or backward over the intervals of an interval list.
**IloIntervalListCursor**

**Constructors**

public **IloIntervalListCursor** (const IloIntervalList)

This constructor creates a cursor to inspect the interval list argument. This cursor lets you iterate forward or backward over the intervals of the interval list. The cursor initially indicates the first interval of the interval list.

public **IloIntervalListCursor** (const IloIntervalList, IloNum x)

This constructor creates a cursor to inspect the interval list intervals. This cursor lets you iterate forward or backward over the interval list. The cursor initially indicates the interval of the interval list that contains x.

Note that if n is the number of intervals of the interval list given as argument, the worst-case complexity of this constructor is $O(\log(n))$.

public **IloIntervalListCursor** (const IloIntervalListCursor &)

This constructor creates a new cursor that is a copy of the argument. The new cursor initially indicates the same interval and the same interval list as the argument cursor.

**Methods**

public **IloNum** getEnd()

This member function returns the end point of the interval currently indicated by the cursor.

public **IloNum** getStart()

This member function returns the start point of the interval currently indicated by the cursor.

public **IloNum** getType()

This member function returns the type of the interval currently indicated by the cursor.

public **IloBool** ok()

This member function returns IloFalse if the cursor does not currently indicate an interval included in the interval list. Otherwise, it returns IloTrue.

public void **operator++()**

**Note:** The structure of the interval list cannot be changed while a cursor is being used to inspect it. Therefore, functions that change the structure of the interval list, such as addInterval, should not be called while the cursor is being used.

**See Also**

IloIntervalList
This operator moves the cursor to the interval adjacent to the current interval (forward move).

```java
public void operator--()
```

This operator moves the cursor to the interval adjacent to the current interval (backward move).

```java
public void operator=(const IloIntervalListCursor &)
```

This operator assigns an address to the handle pointer of the invoking instance of `IloIntervalListCursor`. That address is the location of the implementation object of the argument `cursor`. After the execution of this operator, the invoking object and `cursor` both point to the same implementation object.

```java
public void seek(IloNum)
```

This member function sets the cursor to indicate the first interval of the interval list whose end is strictly greater than `x`. Note that if `n` is the number of intervals of the interval list traversed by the invoking iterator, the worst-case complexity of this member function is $O(\log(n))$. An instance of `IloException` is thrown if `x` does not belong to the interval of definition of the invoking interval list.
IloLineNotFoundException

**Category**
Class

**Inheritance Path**

**Definition File**
ilconcert/ilocsvreader.h

**Description**
This exception is thrown by the following member functions if the line is not found.
- `getLineByKey`
- `getLineByNumber`
- `getLineByKey`
- `getLineByNumber`
IloMax

Category: Global Function

Definition File: ilconcert/ilonumfunc.h

Synopsis: public IloNumToNumStepFunction IloMax(const IloNumToNumStepFunction f1, const IloNumToNumStepFunction f2)

Description: This operator creates and returns a function equal to the maximal value of the functions f1 and f2. That is, for all points x in the definition interval, the resulting function is equal to the \( \max(f1(x), f2(x)) \). The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToNumStepFunction.
**IloMin**

**Category**  
Global Function

**Definition File**  
ilconcert/ilonumfunc.h

**Synopsis**  
public IloNumToNumStepFunction IloMin(const IloNumToNumStepFunction f1,  
const IloNumToNumStepFunction f2)

**Description**  
This operator creates and returns a function equal to the minimal value of the functions f1 and f2. That is, for all points x in the definition interval, the resulting function is equal to the min(f1(x), f2(x)). The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToNumStepFunction.
IloNumToAnySetStepFunction

Category  Class

InheritancePath

Definition File  ilconcert/ilosetfunc.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloNumToAnySetStepFunction(const IloEnv, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloNumToAnySetStepFunction(const IloEnv, IloNum, IloNum, const IloAnySet, const char *)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>add(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public void</td>
<td>add(IloNum, IloNum, IloNum)</td>
</tr>
<tr>
<td>public void</td>
<td>add(IloNum, IloNum, const IloAnySet, IloBool)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>alwaysContains(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>alwaysContains(IloNum, IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>alwaysContains(IloNum, IloNum, IloAny)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>public IloBool alwaysIntersects</td>
<td>(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public IloBool alwaysIntersects</td>
<td>(IloNum, IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool contains</td>
<td>(IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool contains</td>
<td>(IloNum, IloAny)</td>
</tr>
<tr>
<td>public void dilate</td>
<td>(IloNum)</td>
</tr>
<tr>
<td>public void empty</td>
<td>(IloNum, IloNum)</td>
</tr>
<tr>
<td>public IloBool everContains</td>
<td>(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public IloBool everContains</td>
<td>(IloNum, IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool everContains</td>
<td>(IloNum, IloNum, IloAny)</td>
</tr>
<tr>
<td>public IloBool everIntersects</td>
<td>(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public IloBool everIntersects</td>
<td>(IloNum, IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool everIntersects</td>
<td>(IloNum, IloNum, IloAny)</td>
</tr>
<tr>
<td>public void fill</td>
<td>(IloNum, IloNum)</td>
</tr>
<tr>
<td>public IloAnySet getComplementSet</td>
<td>(IloNum)</td>
</tr>
<tr>
<td>public IloNum getDefinitionIntervalMax</td>
<td></td>
</tr>
<tr>
<td>public IloNum getDefinitionIntervalMin</td>
<td></td>
</tr>
<tr>
<td>public IloAnySet getSet</td>
<td>(IloNum)</td>
</tr>
<tr>
<td>public IloBool intersects</td>
<td>(IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool isEmpty</td>
<td>(IloNum)</td>
</tr>
<tr>
<td>public IloBool isFull</td>
<td>(IloNum)</td>
</tr>
<tr>
<td>public void remove</td>
<td>(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public void remove</td>
<td>(IloNum, IloNum, IloAny)</td>
</tr>
<tr>
<td>public void remove</td>
<td>(IloNum, IloNum, const IloAnySet, IloBool)</td>
</tr>
<tr>
<td>public void set</td>
<td>(IloNum, IloNum, IloAny)</td>
</tr>
</tbody>
</table>
IloNumToAnySetStepFunction

Description

An instance of IloNumToAnySetStepFunction represents a step function that associates sets with intervals. It is defined everywhere on an interval \([x_{\text{Min}}, x_{\text{Max}})\). Each interval \([x_1, x_2)\) on which the function has the same set is called a step.

Note that if \(n\) is the number of steps of the function, the random access to a given step (see the member functions add, alwaysIntersects, contains, empty, everContains, everIntersects, fill, getComplementSet, getSet, intersects, isEmpty, isFull, remove, set, and setIntersection) has a worst-case complexity of \(O(\log(n))\).

Complementary Representation of Values

IloNumToAnySetStepFunction allows the implicit representation of infinite sets through the representation of the complement of the actual set value. This, for example, allows you to completely fill a set (using the fill member function) and then specify the elements that are not in the set. Under normal circumstances, it is not necessary to know if the value of the step function at a particular point is represented by the set or its complement: all the member functions that manipulate the step function value will correctly adapt to either representation. The only case where it is necessary to know the internal representation is if you want to directly access the set that represents a value (using the getSet or getComplementSet member functions). In that circumstance only, it is necessary to use the usesComplementaryRepresentation member function to determine the internal representation, and then use either getSet or getComplementSet depending on the return value of usesComplementaryRepresentation. Note that getSet will raise an error if it

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
<td>set(IloNum, IloNum, const IloAnySet, IloBool)</td>
</tr>
<tr>
<td>public void</td>
<td>setIntersection(const IloNumToAnySetStepFunction)</td>
</tr>
<tr>
<td>public void</td>
<td>setIntersection(IloNum, IloNum, IloAny)</td>
</tr>
<tr>
<td>public void</td>
<td>setIntersection(IloNum, IloNum, const IloAnySet, IloBool)</td>
</tr>
<tr>
<td>public void</td>
<td>setPeriodic(const IloNumToAnySetStepFunction, IloNum, IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public void</td>
<td>shift(IloNum, const IloAnySet)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>usesComplementaryRepresentation(IloNum)</td>
</tr>
</tbody>
</table>
is used to access a set that is represented as a complement set. `getComplementSet` will raise an error if it is used to access a set that is directly represented.

**See Also**

- `IloNumToAnySetStepFunctionCursor`

**Constructors**

```java
public IloNumToAnySetStepFunction(const IloEnv env, 
                                    IloNum xmin, 
                                    IloNum xmax, 
                                    const char * name)
```

This constructor creates a step function defined everywhere on the interval \([x_{\text{Min}}, x_{\text{Max}})\) with empty set as the value.

```java
public IloNumToAnySetStepFunction(const IloEnv env, 
                                    IloNum xmin, 
                                    IloNum xmax, 
                                    const IloAnySet dval, 
                                    const char * name)
```

This constructor creates a step function defined everywhere on the interval \([x_{\text{Min}}, x_{\text{Max}})\) with the same set `dval`.

**Methods**

```java
public void add(const IloNumToAnySetStepFunction f)
```

This member function adds the value of `f` at point `x` to the value of the invoking step function at point `x`, for all points `x` in the definition interval of the invoking step function. An instance of `IloException` is thrown if the definition interval of `f` is not equal to the definition interval of the invoking step function.

```java
public void add(IloNum xMin, 
                IloNum xMax, 
                IloAny elt)
```

This member function adds `elt` to the value of the invoking step function on the interval \([x_{\text{Min}}, x_{\text{Max}})\).

```java
public void add(IloNum xMin, 
                IloNum xMax, 
                const IloAnySet elts, 
                IloBool complt)
```

This member function adds the elements of `elts` to the value of the invoking step function on the interval \([x_{\text{Min}}, x_{\text{Max}})\).

```java
public IloBool alwaysContains(const IloNumToAnySetStepFunction f)
```

This member function returns `IloTrue` if for all points `x` on the definition interval of the invoking step function, the value of `f` at point `x` is a subset of the value of the invoking step function at point `x`. An instance of `IloException` is thrown if the definition interval of `f` is not equal to the definition interval of the invoking step function.
public IloBool alwaysContains(IloNum xMin, 
   IloNum xMax, 
   const IloAnySet elts)

This member function returns IloTrue if elts is a subset of the value of the invoking 
step function at all points on the interval [xMin, xMax).

public IloBool alwaysContains(IloNum xMin, 
   IloNum xMax, 
   IloAny elt)

This member function returns IloTrue if at all points on the interval [xMin, 
xMax) the value of the invoking step function contains elt.

public IloBool alwaysIntersects(const IloNumToAnySetStepFunction f)

This member function returns IloTrue if for all points x in the definition interval of 
the invoking step function, the intersection of f and the invoking step function is not 
empty. An instance of IloException is thrown if the definition interval of f is not 
equal to the definition interval of the invoking step function.

public IloBool alwaysIntersects(IloNum xMin, 
   IloNum xMax, 
   const IloAnySet elts)

This member function returns IloTrue if for all x on the interval [xMin, xMax) 
the intersection of elts and the value of the invoking step function at point x is not 
empty.

public IloBool contains(IloNum x, 
   const IloAnySet elts)

This member function returns IloTrue if elts is a subset of the value of the invoking 
step function at point x.

public IloBool contains(IloNum x, 
   IloAny elt)

This member function returns IloTrue if the invoking step function contains element 
elt at point x.

public void dilate(IloNum k)

This member function multiplies by k the scale of x for the invoking step function. k 
must be a non-negative numerical value. More precisely, if the invoking step function 
was defined over an interval [xMin, xMax), it will be redefined over the interval 
[k*xMin, k*xMax) and the value at x will be the former value at x/k.

public void empty(IloNum xMin, 
   IloNum xMax)

This member function sets the value of the invoking step function on the interval 
[xMin, xMax) to be the empty set.

public IloBool everContains(const IloNumToAnySetStepFunction f)
This member function returns \texttt{IloTrue} if at any point \( x \) in the definition interval of the invoking step function, \( f \) at point \( x \) is a subset of the invoking step function at point \( x \). An instance of \texttt{IloException} is thrown if the definition interval of \( f \) is not equal to the definition interval of the invoking step function.

\begin{verbatim}
public IloBool everContains(IloNum xMin,
                           IloNum xMax,
                           const IloAnySet elts)
\end{verbatim}

This member function returns \texttt{IloTrue} if at any point on the interval \([xMin, xMax)\) \( elts \) is a subset of the value of the invoking step function.

\begin{verbatim}
public IloBool everContains(IloNum xMin,
                           IloNum xMax,
                           IloAny elt)
\end{verbatim}

This member function returns \texttt{IloTrue} if at any point on the interval \([xMin, xMax)\) the value of the invoking step function contains \( elt \).

\begin{verbatim}
public IloBool everIntersects(const IloNumToAnySetStepFunction f)
\end{verbatim}

This member function returns \texttt{IloTrue} if at some point \( x \) in the definition interval of the invoking step function, the intersection of \( f \) and the invoking step function is not empty. An instance of \texttt{IloException} is thrown if the definition interval of \( f \) is not equal to the definition interval of the invoking step function.

\begin{verbatim}
public IloBool everIntersects(IloNum xMin,
                              IloNum xMax,
                              const IloAnySet elts)
\end{verbatim}

This member function returns \texttt{IloTrue} if at any point \( x \) on the interval \([xMin, xMax)\) the intersection of \( elts \) and the value of the invoking step function at point \( x \) is not empty.

\begin{verbatim}
public void fill(IloNum xMin,
                 IloNum xMax)
\end{verbatim}

This member function sets the value of the invoking step function on the interval \([xMin, xMax)\) to be the full set.

\begin{verbatim}
public IloAnySet getComplementSet(IloNum x)
\end{verbatim}

This member function returns the complement of the value of the invoking step function at point \( x \). An instance of \texttt{IloException} is thrown if the invoking step function at point \( x \) does not use the complementary representation. See Complementary Representation of Values for more information.

\begin{verbatim}
public IloNum getDefinitionIntervalMax()
\end{verbatim}

This member function returns the right-most point of the definition interval of the invoking step function.

\begin{verbatim}
public IloNum getDefinitionIntervalMin()
\end{verbatim}
This member function returns the left-most point of the definition interval of the invoking step function.

```java
public IloAnySet getSet(IloNum x)
```

This member function returns the value of the invoking step function at point x. An instance of IloException is thrown if the invoking step function at point x uses the complementary representation. See Complementary Representation of Values for more information.

```java
public IloBool intersects(IloNum x, const IloAnySet elts)
```

This member function returns IloTrue if the intersection of elts and the value of the invoking step function at point x is not empty.

```java
public IloBool isEmpty(IloNum x)
```

This member function returns IloTrue if the function is empty at point x. In other words, a return of IloTrue means that the member function empty has been applied to point x and no elements have been subsequently added to the value of the invoking step function at point x.

```java
public IloBool isFull(IloNum x)
```

This member function returns IloTrue if the function is full at point x. In other words, a return of IloTrue means that the member function fill has been applied to point x and no elements have been subsequently removed from the value of the invoking step function at point x.

```java
public void remove(const IloNumToAnySetStepFunction f)
```

This member function removes the value of f from the value of the invoking step function at all points on the definition interval of the invoking step function. An instance of IloException is thrown if the definition interval of f is not equal to the definition interval of the invoking step function.

```java
public void remove(IloNum xMin, IloNum xMax, IloAny elt)
```

This member function removes elt from the value of the invoking step function on the interval [xMin, xMax).

```java
public void remove(IloNum xMin, IloNum xMax, const IloAnySet elts, IloBool complt)
```

This member function removes all the elements in elts from the value of the invoking step function on the interval [xMin, xMax).

```java
public void set(IloNum xMin, IloNum xMax,
```

...
This member function sets the value of the invoking step function to be \texttt{elt} on the interval \([x_{\text{Min}}, x_{\text{Max}})\).

```java
public void set(IloNum xMin,
                IloNum xMax,
                const IloAnySet elts,
                IloBool complt)
```

This member function sets the value of the invoking step function to be \texttt{elts} on the interval \([x_{\text{Min}}, x_{\text{Max}})\).

```java
public void setIntersection(const IloNumToAnySetStepFunction f)
```

This member function assigns the value of the invoking step function at all points \(x\) on the definition interval of the invoking step function to be the intersection of the value of \(f\) at point \(x\) and the value of the invoking step function at point \(x\). An instance of \texttt{IloException} is thrown if the definition interval of \(f\) is not equal to the definition interval of the invoking step function.

```java
public void setIntersection(IloNum xMin,
                            IloNum xMax,
                            IloAny elt)
```

This member function assigns the value of the invoking step function at all points \(x\) on the interval \([x_{\text{Min}}, x_{\text{Max}})\) to be the intersection of the set containing \texttt{elt} and the value of the invoking set function at point \(x\).

```java
public void setIntersection(IloNum xMin,
                            IloNum xMax,
                            const IloAnySet elts,
                            IloBool complt)
```

This member function assigns the value of the invoking step function at all points \(x\) on the interval \([x_{\text{Min}}, x_{\text{Max}})\) to be the intersection of \texttt{elts} and the value of the invoking set function at point \(x\).

```java
public void setPeriodic(const IloNumToAnySetStepFunction f,
                        IloNum x0,
                        IloNum n,
                        const IloAnySet dval)
```

This member function initializes the invoking step function as a function that repeats the step function \(f\), \(n\) times after \(x_0\). More precisely, if \(f\) is defined on \([x_{\text{fpMin}}, x_{\text{fpMax}})\) and if the invoking step function is defined on \([x_{\text{Min}}, x_{\text{Max}})\), the value of the invoking step function will be:

- \(dval\) on \([x_{\text{Min}}, x_0)\).
- \(f((x-x_0) \mod (x_{\text{fpMax}}-x_{\text{fpMin}}))\) for \(x\) in \([x_0, \text{Min}(x_0+n*(x_{\text{fpMax}}-x_{\text{fpMin}}), x_{\text{Max}}))\), and
public void shift(IloNum dx,
const IloAnySet dval)

This member function shifts the invoking step function from dx to the right if dx > 0,
or from -dx to the left if dx < 0. It has no effect if dx = 0. More precisely, if the
invoking step function is defined on [xMin, xMax) and dx > 0, the new value of the
invoking step function is:
◆ dval on the interval [xMin, xMin+dx),
◆ for all x in [xMin+dx, xMax), the former value at x-dx.
If dx < 0, the new value of the invoking step function is:
◆ for all x in [xMin, xMax+dx), the former value at x-dx,
◆ dval on the interval [xMax+dx, xMax).

public IloBool usesComplementaryRepresentation(IloNum x)

This member function returns IloTrue if the value of the invoking function at point x
is represented by a complementary set, rather than by directly representing the value as a
set itself. See Complementary Representation of Values for more information.
IloNumToAnySetStepFunctionCursor

Category  Class

InheritancePath

Definition File  ilconcert/ilosetfunc.h

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumToAnySetStepFunctionCursor (const IloNumToAnySetStepFunction)</td>
<td>Constructor</td>
</tr>
<tr>
<td>public IloNumToAnySetStepFunctionCursor (const IloNumToAnySetStepFunction, IloNum)</td>
<td>Constructor</td>
</tr>
<tr>
<td>public IloNumToAnySetStepFunctionCursor (const IloNumToAnySetStepFunctionCursor &amp;)</td>
<td>Constructor</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloAnySet getComplementSet()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloNum getSegmentMax()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloNum getSegmentMin()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloAnySet getSet()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloBool isEmpty()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloBool isFull()</td>
<td>Method</td>
</tr>
<tr>
<td>public IloBool ok()</td>
<td>Method</td>
</tr>
<tr>
<td>public void operator++()</td>
<td>Method</td>
</tr>
<tr>
<td>public void operator--()</td>
<td>Method</td>
</tr>
<tr>
<td>public void operator=(const IloNumToAnySetStepFunctionCursor &amp;)</td>
<td>Method</td>
</tr>
<tr>
<td>public void seek(IloNum)</td>
<td>Method</td>
</tr>
<tr>
<td>public IloBool usesComplementaryRepresentation()</td>
<td>Method</td>
</tr>
</tbody>
</table>

Description

An instance of the class IloNumToAnySetStepFunctionCursor allows you to inspect the contents of an IloNumToAnySetStepFunction. A step of a step function is defined as an interval [x1,x2) over which the value of the function is the
same. Cursors are intended to iterate forward or backward over the steps of a step function.

**Note:** The structure of the step function cannot be changed while a cursor is being used to inspect it. Therefore, functions that change the structure of the step function, such as set, should not be called while the cursor is being used.

### See Also

- [IloNumToAnySetStepFunction](#)

### Constructors

- **public IloNumToAnySetStepFunctionCursor(const IloNumToAnySetStepFunction)**
  
  This constructor creates a cursor to inspect the step function argument. This cursor lets you iterate forward or backward over the steps of the function. The cursor initially indicates the first step of the function.

- **public IloNumToAnySetStepFunctionCursor(const IloNumToAnySetStepFunction, IloNum x)**
  
  This constructor creates a cursor to inspect the step function argument. This cursor lets you iterate forward or backward over the steps of the function. The cursor initially indicates the step of the function that contains x.

  Note that if n is the number of steps of the function given as argument, the worst-case complexity of this constructor is $O(\log(n))$.

- **public IloNumToAnySetStepFunctionCursor(const IloNumToAnySetStepFunctionCursor i)**
  
  This constructor creates a new cursor that is a copy of the argument. The new cursor initially indicates the same step and the same function as the argument cursor.

### Methods

- **public IloSet getComplementSet()**
  
  This member function returns the set representing the complement of the value of the step currently indicated by the cursor. An instance of IloException is thrown if the value of the step does not use a complementary representation.

- **public IloNum getSegmentMax()**
  
  This member function returns the right-most point of the step currently indicated by the cursor.

- **public IloNum getSegmentMin()**
  
  This member function returns the right-most point of the step currently indicated by the cursor.
This member function returns the left-most point of the step currently indicated by the cursor.

```java
public IloAnySet getSet()
```

This member function returns the value of the step currently indicated by the cursor. An instance of IloException is thrown if the value of the step uses a complementary representation.

```java
public IloBool isEmpty()
```

This member function returns IloTrue if the value of the current step is the empty set.

```java
public IloBool isFull()
```

This member function returns IloTrue if the value of the current step is the full set. (See also: isFull).

```java
public IloBool ok()
```

This member function returns IloFalse if the cursor does not currently indicate a step included in the definition interval of the step function. Otherwise, it returns IloTrue.

```java
public void operator++()
```

This operator moves the cursor to the step adjacent to the current step (forward move).

```java
public void operator--()
```

This operator moves the cursor to the step adjacent to the current step (backward move).

```java
public void operator=(const IloNumToAnySetStepFunctionCursor &)
```

This operator assigns an address to the handle pointer of the invoking instance of IloNumToAnySetStepFunctionCursor. That address is the location of the implementation object of the argument cursor. After the execution of this operator, the invoking object and cursor both point to the same implementation object.

```java
public void seek(IloNum)
```

This member function sets the cursor to indicate the step of the function that contains x. Note that if n is the number of steps of the step function traversed by the invoking iterator, the worst-case complexity of this member function is O(log(n)). An instance of IloException is thrown if x does not belong to the definition interval of the invoking function.

```java
public IloBool usesComplementaryRepresentation()
```

This member function returns IloTrue if the value of the current step uses the complementary representation.
# IloNumToNumSegmentFunction

## Category
Class

## InheritancePath

### Definition File
ilconcert/ilosegfunc.h

## Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloNumToNumSegmentFunction(const IloEnv, IloNum, IloNum, IloNum, const char *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloNumToNumSegmentFunction(const IloEnv, const IloNumArray, const IloNumArray, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td>Public</td>
<td>IloNumToNumSegmentFunction(const IloNumToNumStepFunction &amp;)</td>
</tr>
</tbody>
</table>

## Method Summary

| Public void | addValue(IloNum, IloNum, IloNum) |
| Public void | dilate(IloNum) |
| Public IloNum | getArea(IloNum, IloNum) |
| Public IloNum | getDefinitionIntervalMax() |
| Public IloNum | getDefinitionIntervalMin() |
| Public IloNum | getMax(IloNum, IloNum) |
| Public IloNum | getMin(IloNum, IloNum) |
IloNumToNumSegmentFunction

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum getValue(IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void operator *=(IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void operator+=(const IloNumToNumSegmentFunction)</td>
<td></td>
</tr>
<tr>
<td>public void operator-=(const IloNumToNumSegmentFunction)</td>
<td></td>
</tr>
<tr>
<td>public void setMax(const IloNumToNumSegmentFunction)</td>
<td></td>
</tr>
<tr>
<td>public void setMax(IloNum, IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setMin(const IloNumToNumSegmentFunction)</td>
<td></td>
</tr>
<tr>
<td>public void setMin(IloNum, IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setMin(IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setPeriodic(const IloNumToNumSegmentFunction, IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setPeriodicValue(IloNum, IloNum, const IloNumToNumSegmentFunction, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setPoints(const IloNumArray, const IloNumArray)</td>
<td></td>
</tr>
<tr>
<td>public void setSlope(IloNum, IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void setValue(IloNum, IloNum, IloNum)</td>
<td></td>
</tr>
<tr>
<td>public void shift(IloNum, IloNum)</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

An instance of IloNumToNumSegmentFunction represents a piecewise linear function that is defined everywhere on an interval \([x_{\text{Min}}, x_{\text{Max}}]\). Each interval \([x_1, x_2)\) on which the function is linear is called a *segment*.

Note that if \(n\) is the number of segments of the function, the random access to a given segment (see the member functions `addValue`, `getArea`, `getValue`, `setValue`) has a worst-case complexity in \(O(\log(n))\).
Furthermore, when two consecutive segments of the function are co-linear, these segments are merged so that the function is always represented with the minimal number of segments.

See Also

IloNumToNumSegmentFunctionCursor

Constructors

public IloNumToNumSegmentFunction (const IloEnv  env,
IloNum  xmin,
IloNum  xmax,
IloNum  dval,
const char * name)

This constructor creates a piecewise linear function that is constant. It is defined everywhere on the interval \([\text{xmin}, \text{xmax})\) with the same value \(dval\).

public IloNumToNumSegmentFunction (const IloEnv  env,
const IloNumArray  x,
const IloNumArray  v,
IloNum  xmin,
IloNum  xmax,
const char * name)

This constructor creates a piecewise linear function defined everywhere on the interval \([\text{xmin}, \text{xmax})\) whose segments are defined by the two argument arrays \(x\) and \(v\). More precisely, the size \(n\) of array \(x\) must be equal to the size of array \(v\) and, if the created function is defined on the interval \([\text{xmin}, \text{xmax})\), its values will be:

- \(v[0]\) on interval \([\text{xmin}, x[0))\),
- \(v[i] + (t-x[i])*(v[i+1]-v[i])/(x[i+1]-x[i])\) for \(t\) in \([x[i], x[i+1))\) for all \(i\) in \([0, n-2]\) such that \(x[i-1] <> x[i]\), and
- \(v[n-1]\) on interval \([x[n-1], xmax)\).

public IloNumToNumSegmentFunction (const IloNumToNumStepFunction & numFunction)

This copy constructor creates a new piecewise linear function. The new piecewise linear function is a copy of the step function \(numFunction\). They point to different implementation objects.

Methods

public void addValue (IloNum  x1,
                   IloNum  x2,
                   IloNum  v)

This member function adds \(v\) to the value of the invoking piecewise linear function everywhere on the interval \([x1, x2)\).

public void dilate (IloNum  k)

This member function multiplies by \(k\) the scale of \(x\) for the invoking piecewise linear function. \(k\) must be a non-negative numerical value. More precisely, if the invoking
function was defined over an interval \([x_{\text{Min}}, x_{\text{Max}}]\), it will be redefined over the interval \([k \cdot x_{\text{Min}}, k \cdot x_{\text{Max}}]\) and the value at \(x\) will be the former value at \(x/k\).

```java
public IloNum getArea(IloNum x1, IloNum x2)
```

This member function returns the area of the invoking piecewise linear function on the interval \([x_1, x_2]\). An instance of `IloException` is thrown if the interval \([x_1, x_2]\) is not included in the definition interval of the invoking function.

```java
public IloNum getDefinitionIntervalMax()
```

This member function returns the right-most point of the definition interval of the invoking piecewise linear function.

```java
public IloNum getDefinitionIntervalMin()
```

This member function returns the left-most point of the definition interval of the invoking piecewise linear function.

```java
public IloNum getMax(IloNum x1, IloNum x2)
```

This member function returns the maximal value of the invoking piecewise linear function on the interval \([x_1, x_2]\). An instance of `IloException` is thrown if the interval \([x_1, x_2]\) is not included in the definition interval of the invoking function.

```java
public IloNum getMin(IloNum x1, IloNum x2)
```

This member function returns the minimal value of the invoking piecewise linear function on the interval \([x_1, x_2]\). An instance of `IloException` is thrown if the interval \([x_1, x_2]\) is not included in the definition interval of the invoking function.

```java
public IloNum getValue(IloNum x)
```

This member function returns the value of the function at point \(x\).

```java
public void operator *= (IloNum k)
```

This operator multiplies by a factor \(k\) the value of the invoking piecewise linear function everywhere on the definition interval.

```java
public void operator += (const IloNumToNumSegmentFunction fct)
```

This operator adds the argument function \(fct\) to the invoking piecewise linear function.

```java
public void operator -= (const IloNumToNumSegmentFunction fct)
```

This operator subtracts the argument function \(fct\) from the invoking piecewise linear function.

```java
public void setMax(const IloNumToNumSegmentFunction fct)
```

This member function sets the value of the invoking piecewise linear function to be the maximum between the current value and the value of \(fct\) everywhere on the definition interval.
interval of the invoking function. The interval of definition of \( f_{ct} \) must be the same as that of the invoking piecewise linear function.

```java
public void setMax(IloNum x1,
                 IloNum v1,
                 IloNum x2,
                 IloNum v2)
```

This member function sets the value of the invoking piecewise linear function to be the maximum between the current value and the value of the linear function:

\[
x \rightarrow v1 + (x-x1)*(v2-v1)/(x2-x1)
\]
everywhere on the interval \([x1, x2)\).

```java
public void setMax(IloNum x1,
                 IloNum x2,
                 IloNum v)
```

This member function sets the value of the invoking piecewise linear function to be the maximum between the current value and \( v \) everywhere on the interval \([x1, x2)\).

```java
public void setMin(const IloNumToNumSegmentFunction fct)
```

This member function sets the value of the invoking piecewise linear function to be the minimum between the current value and the value of \( f_{ct} \) everywhere on the definition interval of the invoking function. The definition interval of \( f_{ct} \) must be the same as the one of the invoking piecewise linear function.

```java
public void setMin(IloNum x1,
                 IloNum v1,
                 IloNum x2,
                 IloNum v2)
```

This member function sets the value of the invoking piecewise linear function to be the minimum between the current value and the value of the linear function:

\[
x \rightarrow v1 + (x-x1)*(v2-v1)/(x2-x1)
\]
everywhere on the interval \([x1, x2)\).

```java
public void setMin(IloNum x1,
                 IloNum x2,
                 IloNum v)
```

This member function sets the value of the invoking piecewise linear function to be the minimum between the current value and \( v \) everywhere on the interval \([x1, x2)\).

```java
public void setPeriodic(const IloNumToNumSegmentFunction f,
                        IloNum x0,
                        IloNum n,
                        IloNum dval)
```

This member function initializes the invoking function as a piecewise linear function that repeats the piecewise linear function \( f \), \( n \) times after \( x0 \). More precisely, if \( f \) is defined on \([xfpMin, xfpMax)\) and if the invoking function is defined on \([xMin, xMax)\), the value of the invoking function will be:
IloNumToNumSegmentFunction

- \( dval \) on \([xMin, x0)\).
- \( f((x-x0) \% (xfpMax-xfpMin)) \) for \( x \in [x0, \min(x0+n*(xfpMax-xfpMin)), xMax)) \), and
- \( dval \) on \([\min(x0+n*(xfpMax-xfpMin), xMax)), xMax)\).

public void setPeriodicValue(IloNum x1, IloNum x2, const IloNumToNumSegmentFunction f, IloNum offset)

This member function changes the value of the invoking function on the interval \([x1, x2)\). On this interval, the invoking function is set to equal a repetition of the pattern function \( f \) with an initial offset of \( offset \). The invoking function is not modified outside the interval \([x1, x2)\). More precisely, if \([min, max)\) denotes the definition interval of \( f \), for all \( t \) in \([x1, x2)\), the invoking function at \( t \) is set to equal \( f(min + (offset+t-x1) \% (max-min))) \) where \( \% \) denotes the modulo operator. By default, the offset is equal to 0.

public void setPoints(const IloNumArray x, const IloNumArray v)

This member function initializes the invoking function as a piecewise linear function whose segments are defined by the two parameters arrays \( x \) and \( v \).

More precisely, the size \( n \) of array \( x \) must be equal to the size of array \( v \), and if the created function is defined on the interval \([xmin, xmax)\), its values will be:

- \( v[0] \) on interval \([xmin, x[0))\).
- \( v[i] + (t-x[i])*v[i+1]-v[i])/(x[i+1]-x[i]) \) for \( t \) in \([x[i], x[i+1))\) for all \( i \) in \([0, n-2)\) such that \( x[i-1] \neq x[i]\).
- \( v[n-1] \) on interval \([x[n-1], xmax)\).

public void setSlope(IloNum x1, IloNum x2, IloNum v, IloNum slope)

This member function sets the value of the invoking piecewise linear function equal to \( f, \) associating for each \( x \) in \([x1, x2) \rightarrow f(x) = v + slope * (x-x1)\).

public void setValue(IloNum x1, IloNum x2, IloNum v)

This member function sets the value of the invoking piecewise linear function to be constant and equal to \( v \) on the interval \([x1, x2)\).

public void shift(IloNum dx, IloNum dval)
This member function shifts the invoking function from \( dx \) to the right if \( dx > 0 \) or \( -dx \) to the left if \( dx < 0 \). It has no effect if \( dx = 0 \). More precisely, if the invoking function is defined on \([x_{\text{Min}}, x_{\text{Max}})\) and \( dx > 0 \), the new value of the invoking function is:

- \( dval \) on the interval \([x_{\text{Min}}, x_{\text{Min}} + dx)\).
- for all \( x \) in \([x_{\text{Min}} + dx, x_{\text{Max}})\), the former value at \( x - dx \).

If \( dx < 0 \), the new value of the invoking function is:

- for all \( x \) in \([x_{\text{Min}}, x_{\text{Max}} + dx)\), the former value at \( x - dx \).
- \( dval \) on the interval \([x_{\text{Max}} + dx, x_{\text{Max}})\).
IloNumToNumSegmentFunctionCursor

Category Class

InheritancePath

Definition File ilconcert/ilosegfunc.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumToNumSegmentFunctionCursor</td>
<td>(const IloNumToNumSegmentFunction , IloNum)</td>
</tr>
<tr>
<td>public IloNumToNumSegmentFunctionCursor</td>
<td>(const IloNumToNumSegmentFunctionCursor &amp;)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum</td>
<td>getSegmentMax()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getSegmentMin()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValue(IloNum)</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValueLeft()</td>
</tr>
<tr>
<td>public IloNum</td>
<td>getValueRight()</td>
</tr>
<tr>
<td>public IloBool</td>
<td>ok()</td>
</tr>
<tr>
<td>public void</td>
<td>operator++()</td>
</tr>
<tr>
<td>public void</td>
<td>operator--()</td>
</tr>
<tr>
<td>public void</td>
<td>seek(IloNum)</td>
</tr>
</tbody>
</table>

### Description

An instance of the class IloNumToNumSegmentFunctionCursor allows you to inspect the contents of an IloNumToNumSegmentFunction. A segment of a piecewise linear function is defined as an interval \([x_1, x_2]\) over which the function is linear. Cursors are intended to iterate forward or backward over the segments of a piecewise linear function.
Note: The structure of the piecewise linear function cannot be changed while a cursor is being used to inspect it. Therefore, functions that change the structure of the piecewise linear function, such as `setValue`, should not be called while the cursor is being used.

See Also

*IloNumToNumSegmentFunction*

Constructors

```java
class IloNumToNumSegmentFunctionCursor {
    public IloNumToNumSegmentFunctionCursor(const IloNumToNumSegmentFunction &, IloNum x)
    { ... }  // This constructor creates a cursor to inspect the piecewise linear function argument. This cursor lets you iterate forward or backward over the segments of the function. The cursor initially indicates the segment of the function that contains x.
    public IloNumToNumSegmentFunctionCursor(const IloNumToNumSegmentFunctionCursor &)
    { ... }  // This constructor creates a new cursor that is a copy of the argument cursor. The new cursor initially indicates the same segment and the same function as the argument cursor.
}
```

Methods

```java
class IloNumToNumSegmentFunctionCursor {
    public IloNum getSegmentMax()
    { ... }  // This member function returns the right-most point of the segment currently indicated by the cursor.
    public IloNum getSegmentMin()
    { ... }  // This member function returns the left-most point of the segment currently indicated by the cursor.
    public IloNum getValue(IloNum t)
    { ... }  // This member function returns the value of the piecewise linear function at time t. t must be between the left-most and the right-most point of the segment currently indicated by the cursor.
    public IloNum getValueLeft()
    { ... }  // This member function returns the value of the function at the left-most point of the segment currently indicated by the cursor.
    public IloNum getValueRight()
    { ... }  // This member function returns the value of the function at the right-most point of the segment currently indicated by the cursor.
}
```
This member function returns the value of the function at the right-most point of the segment currently indicated by the cursor.

public IloBool ok()

This member function returns IloFalse if the cursor does not currently indicate a segment included in the definition interval of the piecewise linear function. Otherwise, it returns IloTrue.

public void operator++()

This operator moves the cursor to the segment adjacent to the current step (forward move).

public void operator--()

This operator moves the cursor to the segment adjacent to the current step (backward move).

public void seek(IloNum)

This member function sets the cursor to indicate the segment of the function that contains x. An IloException is thrown if x does not belong to the definition interval of the piecewise linear function associated with the invoking cursor.
IloNumToNumStepFunction

Category  Class

InheritancePath

Definition File  ilconcert/ilonumfunc.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IloNumToNumStepFunction(const IloEnv, IloNum, IloNum, IloNum, const char *)</td>
</tr>
<tr>
<td></td>
<td>IloNumToNumStepFunction(const IloEnv, const IloNumArray, const IloNumArray, IloNum, IloNum, const char *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>addValue(IloNum, IloNum, IloNum)</td>
</tr>
<tr>
<td>void</td>
<td>dilate(IloNum)</td>
</tr>
<tr>
<td>IloNum</td>
<td>getArea(IloNum, IloNum)</td>
</tr>
<tr>
<td>IloNum</td>
<td>getMax(IloNum, IloNum)</td>
</tr>
<tr>
<td>IloNum</td>
<td>getMin(IloNum, IloNum)</td>
</tr>
<tr>
<td>IloNum</td>
<td>getValue(IloNum)</td>
</tr>
<tr>
<td>void</td>
<td>operator *(IloNum)</td>
</tr>
</tbody>
</table>
Description

An instance of IloNumToNumStepFunction represents a step function that is defined everywhere on an interval \([x_{\text{Min}}, x_{\text{Max}}]\). Each interval \([x_1, x_2]\) on which the function has the same value is called a step.

Note that if \(n\) is the number of steps of the function, the random access to a given step (see the member functions `addValue`, `getArea`, `getValue`, `setValue`) has a worst-case complexity in \(O(\log(n))\).

Furthermore, when two consecutive steps of the function have the same value, these steps are merged so that the function is always represented with the minimal number of steps.

See Also

IloNumToNumStepFunctionCursor

Constructors

public IloNumToNumStepFunction(const IloEnv env,
                                 IloNum xmin,
                                 IloNum xmax,
                                 IloNum dval,
                                 const char * name)
This constructor creates a step function defined everywhere on the interval \([xmin, xmax)\) with the same value \(dval\).

```java
public IloNumToNumStepFunction(const IloEnv env,
                                const IloNumArray x,
                                const IloNumArray v,
                                IloNum xmin,
                                IloNum xmax,
                                const char * name)
```

This constructor creates a step function defined everywhere on the interval \([xmin, xmax)\) whose steps are defined by the two argument arrays \(x\) and \(v\). More precisely, if \(n\) is the size of array \(x\), size of array \(v\) must be \(n+1\) and, if the created function is defined on the interval \([xmin, xmax)\), its values will be:

- \(v[0]\) on interval \([xmin, x[0))\).
- \(v[i]\) on interval \([x[i-1], x[i))\) for all \(i\) in \([0, n-1]\), and
- \(v[n]\) on interval \([x[n-1], xmax))\).

The values in the array are copied, and no modification to the arrays will be taken into account once the constructor has been called.

### Methods

- **public void addValue(IloNum x1,**
  `IloNum x2,**
  `IloNum v)`

  This member function adds \(v\) to the value of the invoking step function everywhere on the interval \([x1, x2))\).

- **public void dilate(IloNum k)**

  This member function multiplies by \(k\) the scale of \(x\) for the invoking step function. \(k\) must be a non-negative numerical value. More precisely, if the invoking function was defined over an interval \([xMin, xMax)\), it will be redefined over the interval \([k*xMin, k*xMax))\) and the value at \(x\) will be the former value at \(x/k\).

- **public IloNum getArea(IloNum x1,**
  `IloNum x2)`

  This member function returns the sum of the invoking step function on the interval \([x1, x2])\). An instance of IloException is thrown if the interval \([x1, x2)\) is not included in the definition interval of the invoking function.

- **public IloNum getDefinitionIntervalMax()**

  This member function returns the right-most point of the definition interval of the invoking step function.

- **public IloNum getDefinitionIntervalMin()**
This member function returns the left-most point of the definition interval of the invoking step function.

```java
public IloNum getMax(IloNum x1,
                      IloNum x2)
```

This member function returns the maximal value of the invoking step function on the interval [x1, x2). An instance of IloException is thrown if the interval [x1, x2) is not included in the definition interval of the invoking function.

```java
public IloNum getMin(IloNum x1,
                      IloNum x2)
```

This member function returns the minimal value of the invoking step function on the interval [x1, x2). An instance of IloException is thrown if the interval [x1, x2) is not included in the definition interval of the invoking function.

```java
public IloNum getValue(IloNum x)
```

This member function returns the value of the invoking step function at x. An instance of IloException is thrown if x does not belong to the definition interval of the invoking function.

```java
public void operator *= (IloNum k)
```

This operator multiplies by a factor k the value of the invoking step function everywhere on the definition interval.

```java
public void operator += (const IloNumToNumStepFunction fct)
```

This operator adds the argument function fct to the invoking step function.

```java
public void operator -= (const IloNumToNumStepFunction fct)
```

This operator subtracts the argument function fct from the invoking step function.

```java
public void setMax(const IloNumToNumStepFunction fct)
```

This member function sets the value of the invoking step function to be the maximum between the current value and the value of fct everywhere on the definition interval of the invoking function. The interval of definition of fct must be the same as that of the invoking step function.

```java
public void setMax(IloNum x1,
                   IloNum x2,
                   IloNum v)
```

This member function sets the value of the invoking step function to be the maximum between the current value and v everywhere on the interval [x1, x2).

```java
public void setMin(const IloNumToNumStepFunction fct)
```

This member function sets the value of the invoking step function to be the minimum between the current value and the value of fct everywhere on the definition interval of
the invoking function. The definition interval of fct must be the same as the one of the invoking step function.

```java
public void setMin(IloNum x1,
                   IloNum x2,
                   IloNum v)
```

This member function sets the value of the invoking step function to be the minimum between the current value and v everywhere on the interval [x1, x2).

```java
public void setPeriodic(const IloNumToNumStepFunction f,
                        IloNum x0,
                        IloNum n,
                        IloNum dval)
```

This member function initializes the invoking function as a step function that repeats the step function f, n times after x0. More precisely, if f is defined on [xfpMin, xfpMax) and if the invoking function is defined on [xMin, xMax), the value of the invoking function will be:

- dval on [xMin, x0),
- fp((x-x0) % (xfpMax-xfpMin)) for x in [x0, Min(x0+n*(xfpMax-xfpMin), xMax)), and
- dval on [Min(x0+n*(xfpMax-xfpMin), xMax), xMax)

```java
public void setPeriodicValue(IloNum x1,
                             IloNum x2,
                             const IloNumToNumStepFunction f,
                             IloNum offset)
```

This member function changes the value of the invoking function on the interval [x1, x2). On this interval, the invoking function is set to equal a repetition of the pattern function f with an initial offset of offset. The invoking function is not modified outside the interval [x1, x2). More precisely, if [min, max) denotes the definition interval of f, for all t in [x1, x2), the invoking function at t is set to equal f(min + (offset+t-x1) % (max-min)) where % denotes the modulo operator. By default, the offset is equal to 0.

```java
public void setSteps(const IloNumArray x,
                     const IloNumArray v)
```

This member function initializes the invoking function as a step function whose steps are defined by the two arguments arrays x and v. More precisely, if n is the size of array x, size of array v must be n+1 and, if the invoking function is defined on the interval [xMin, xMax), its values will be:

- v[0] on interval [xMin, x[0]),
- v[i] on interval [x[i-1], x[i]) for all i in [0, n-1], and
- v[n] on interval [x[n-1], xMax).
public void setValue(IloNum x1,  
        IloNum x2,  
        IloNum v)

This member function sets the value of the invoking step function to be \( v \) on the interval \([x1, x2)\).

public void shift(IloNum dx,  
        IloNum dval)

This member function shifts the invoking function from \( dx \) to the right if \( dx > 0 \) or from \(-dx\) to the left if \( dx < 0 \). It has no effect if \( dx = 0 \). More precisely, if the invoking function is defined on \([xMin, xMax)\) and \( dx > 0 \), the new value of the invoking function is:

\[
\begin{align*}
\text{◆} & \quad \text{dval on the interval } [xMin, xMin+dx), \\
\text{◆} & \quad \text{for all } x \text{ in } [xMin+dx, xMax), \text{ the former value at } x-dx.
\end{align*}
\]

If \( dx < 0 \), the new value of the invoking function is:

\[
\begin{align*}
\text{◆} & \quad \text{for all } x \in [xMin, xMax+dx), \text{ the former value at } x-dx, \\
\text{◆} & \quad \text{dval on the interval } [xMax+dx, xMax).
\end{align*}
\]
IloNumToNumStepFunctionCursor

Category Class

InheritancePath

Definition File ilconcert/ilonumfunc.h

**Constructor Summary**

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNumToNumStepFunctionCursor(const IloNumToNumStepFunction, IloNum)</td>
</tr>
<tr>
<td>public IloNumToNumStepFunctionCursor(const IloNumToNumStepFunctionCursor&amp;)</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloNum getSegmentMax()</td>
</tr>
<tr>
<td>public IloNum getSegmentMin()</td>
</tr>
<tr>
<td>public IloNum getValue()</td>
</tr>
<tr>
<td>public IloBool ok()</td>
</tr>
<tr>
<td>public void operator++()</td>
</tr>
<tr>
<td>public void operator--()</td>
</tr>
<tr>
<td>public void seek(IloNum)</td>
</tr>
</tbody>
</table>

**Description**

An instance of the class IloNumToNumStepFunctionCursor allows you to inspect the contents of an IloNumToNumStepFunction. A step of a step function is defined as an interval \([x_1, x_2)\) over which the value of the function is the same. Cursors are intended to iterate forward or backward over the steps of a step function.
See Also

IloNumToNumStepFunction

Constructors

public IloNumToNumStepFunctionCursor(const IloNumToNumStepFunction , IloNum x)

This constructor creates a cursor to inspect the step function argument. This cursor lets you iterate forward or backward over the steps of the function. The cursor initially indicates the step of the function that contains x.

Note that if n is the number of steps of the function given as argument, the worst-case complexity of this constructor is $O(\log(n))$.

public IloNumToNumStepFunctionCursor(const IloNumToNumStepFunctionCursor &)

This constructor creates a new cursor that is a copy of the argument cursor. The new cursor initially indicates the same step and the same function as the argument cursor.

Methods

public IloNum getSegmentMax()

This member function returns the right-most point of the step currently indicated by the cursor.

public IloNum getSegmentMin()

This member function returns the left-most point of the step currently indicated by the cursor.

public IloNum getValue()

This member function returns the value of the step currently indicated by the cursor.

public IloBool ok()

This member function returns IloFalse if the cursor does not currently indicate a step included in the definition interval of the step function. Otherwise, it returns IloTrue.

public void operator++()

This operator moves the cursor to the step adjacent to the current step (forward move).

public void operator--()

This operator moves the cursor to the step adjacent to the current step (backward move).

Note: The structure of the step function cannot be changed while a cursor is being used to inspect it. Therefore, functions that change the structure of the step function, such as setValue, should not be called while the cursor is being used.
public void seek(IloNum)

This member function sets the cursor to indicate the step of the function that contains x. An IloException is thrown if x does not belong to the definition interval of the step function associated with the invoking cursor.
IloTableNotFoundException

Category: Class

InheritancePath:

Definition File: ilconcert/ilocsvreader.h

Description:
This exception is thrown by the constructor
IloCsvTableReader(IloCsvReaderI *, const char * name = 0)
and by the member functions listed below if the table you want to construct or to get is not found.

◆ getTableByNumber
◆ getTableByName
◆ getTable
IloUnion

Category: Global Function

Definition File: ilconcert/ilointervals.h

Synopsis:

```cpp
public IloIntervalList IloUnion(const IloIntervalList intervals1,
                                 const IloIntervalList intervals2)
```

Description:

This operator creates and returns an interval list equal to the union of the interval lists `intervals1` and `intervals2`. The arguments `intervals1` and `intervals2` must be defined on the same interval. An instance of `IloException` is thrown if two intervals with different types overlap. The resulting interval list is defined on the same interval as the arguments. See also: `IloIntervalList`. 
IloUnion

Category                  Global Function
Definition File           ilconcert/ilosetfunc.h
Synopsis                  public IloNumToAnySetStepFunction IloUnion(const IloNumToAnySetStepFunction f1,
                                      const IloNumToAnySetStepFunction f2)
Description               This operator creates and returns a function equal to the union of the functions f1 and f2. The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToAnySetStepFunction.
**operator ***

**Category**  
Global Function

**Definition File**  
ilconcert/ilonumfunc.h

**Synopsis**  

public IloNumToNumStepFunction operator *(const IloNumToNumStepFunction f1, IloNum k)

public IloNumToNumStepFunction operator *(IloNum k, const IloNumToNumStepFunction f1)

**Description**  
These operators create and return a function equal to the function f1 multiplied by a factor k everywhere on the definition interval. The resulting function is defined on the same interval as the argument function f1. See also: `IloNumToNumStepFunction`. 
**operator+**

**Category**  
Global Function

**Definition File**  
ilconcert/ilonumfunc.h

**Synopsis**  
public IloNumToNumStepFunction operator+(const IloNumToNumStepFunction f1,  
const IloNumToNumStepFunction f2)

**Description**  
This operator creates and returns a function equal the sum of the functions f1 and f2. The argument functions f1 and f2 must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToNumStepFunction.
operator-

Category: Global Function

Definition File: ilconcert/ilonumfunc.h

Synopsis:

```c
public IloNumToNumStepFunction operator-(const IloNumToNumStepFunction f1,
const IloNumToNumStepFunction f2)
```

Description:

This operator creates and returns a function equal to the difference between functions $f_1$ and $f_2$. The argument functions $f_1$ and $f_2$ must be defined on the same interval. The resulting function is defined on the same interval as the arguments. See also: IloNumToNumStepFunction.
**operator<<**

**Category**  Global Function

**Definition File**  ilconcert/ilocsvreader.h

**Synopsis**  

public ostream & operator<<(ostream & out,  
const IloCsvLine & line)

**Description**  

This operator has been overloaded to treat an IloCsvLine object appropriately as output. It directs its output to an output stream (normally, standard output) and displays information about its second argument line.
**operator==**

**Category**
Global Function

**Definition File**
ilconcert/ilointervals.h

**Synopsis**
public IloBool operator==(const IloIntervalList intervals1, const IloIntervalList intervals2)

**Description**
This operator returns IloTrue if the interval lists are the same. That is, IloTrue is returned if they have the same definition interval and if they contain the same intervals. Note that it compares the content of the interval lists as well as the equality of implementation pointer. See also: IloIntervalList.
### operator==

<table>
<thead>
<tr>
<th>Category</th>
<th>Global Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition File</td>
<td>ilconcert/ilonumfunc.h</td>
</tr>
<tr>
<td>Synopsis</td>
<td>public IloBool operator==(const IloNumToNumStepFunction f1, const IloNumToNumStepFunction f2)</td>
</tr>
<tr>
<td>Description</td>
<td>This operator returns IloTrue if the functions f1 and f2 are the same. That is, IloTrue is returned if they have the same definition interval and if they have the same value over time. Note that it compares the content of the functions as well as the equality of implementation pointer. See also: IloNumToNumStepFunction.</td>
</tr>
</tbody>
</table>
operator==

Category  Global Function
Definition File  ilconcert/ilosetfunc.h
Synopsis  public IloBool operator==(const IloNumToAnySetStepFunction f1,
                     const IloNumToAnySetStepFunction f2)
Description  This operator returns IloTrue if the functions are the same. That is, IloTrue is
              returned if they have the same definition interval and if they have the same value over
              time. Note that it compares the content of the functions as well as the equality of
              implementation pointer. See also: IloNumToAnySetStepFunction.

Group optim.concert.xml

The ILOG Concert Serialization API.

<table>
<thead>
<tr>
<th>Classes Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloXmlContext</td>
</tr>
<tr>
<td>IloXmlInfo</td>
</tr>
<tr>
<td>IloXmlReader</td>
</tr>
<tr>
<td>IloXmlWriter</td>
</tr>
</tbody>
</table>

Description  The ILOG Concert Serialization API.
## IloXmlContext

**Category**  
Class

**Inheritance Path**

- **IloXmlContext**

**Definition File**  
ilconcert/iloxmlcontext.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloXmlContext(IloEnv)</td>
<td></td>
</tr>
<tr>
<td>public IloXmlContext(IloXmlContextI *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void end()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getChildIdReadError()</td>
<td></td>
</tr>
<tr>
<td>public const char * getChildTagReadError()</td>
<td></td>
</tr>
<tr>
<td>public IloIntArray getIdListReadError()</td>
<td></td>
</tr>
<tr>
<td>public const IloXmlContextI * getImpl()</td>
<td></td>
</tr>
<tr>
<td>public IloInt getParentIdReadError()</td>
<td></td>
</tr>
<tr>
<td>public const char * getParentTagReadError()</td>
<td></td>
</tr>
<tr>
<td>public IloAnyArray getTagListReadError()</td>
<td></td>
</tr>
<tr>
<td>public const char * getWriteError()</td>
<td></td>
</tr>
<tr>
<td>public int getWritePrecision()</td>
<td></td>
</tr>
<tr>
<td>public IloBool readExtractable(IloXmlReader, IloXmlElement *)</td>
<td></td>
</tr>
<tr>
<td>public IloBool readModel(IloModel, istream &amp;)</td>
<td></td>
</tr>
<tr>
<td>public IloBool readModel(IloModel, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloBool readModelAndSolution(IloModel, const char *, IloSolution, const char *)</td>
<td></td>
</tr>
</tbody>
</table>
Description

An instance of IloXmlContext allows you to serialize an IloModel or an IloSolution in XML.

You can write an IloModel using `writeModel`, write an IloSolution using `writeSolution`, or write both using `writeModelAndSolution`.

You can read an IloModel in XML using `readModel`, read an IloSolution in XML using `readSolution`, or read both using `readModelAndSolution`.

Other products should add their own serialization class and add them to the plug-in using the member functions `registerXML` and `registerXMLArray`.

Examples

For example, you can write:

```cpp
IloModel model(env);
IloSolution solution(env);
...;
IloXmlcontext context(env);
context.writeModel(model, "model.xml");
```
context.writeSolution(solution, "solution.xml");

or you can write

IloModel model(env);
IloSolution solution(env);
IloXmlContext context(env);
context.readModel(model, "model.xml");
context.readSolution(solution, "solution.xml");

See Also
IloXmlReader, IloXmlWriter, IloXmlInfo

Constructors
public IloXmlContext(IloEnv env)
This constructor creates an XML context and makes it part of the environment env.

public IloXmlContext(IloXmlContextI * impl)
This constructor creates a XML context from its implementation object.

Methods
public void end()
This member function deletes the invoking XML context.

public IloInt getChildIdReadError()
This member function returns the XML ID of the child unparsed XML element in cases where a problem occurs when reading an IloModel.

public const char * getChildTagReadError()
This member function returns the XML tag of the child unparsed XML element in cases where a problem occurs when reading an IloModel.

public IloIntArray getIdListReadError()
This member function returns the XML ID list of the unparsed XML elements in cases where a problem occurs when reading an IloModel. The list is composed of the tags from the parent to the child elements.

public const IloXmlContextI * getImpl()
This member function returns the IloXmlContextI implementation.

public IloInt getParentIdReadError()
This member function returns the XML ID of the parent unparsed XML element in cases where a problem occurs when reading an IloModel.

public const char * getParentTagReadError()
This member function returns the XML tag of the parent unparsed XML element in cases where a problem occurs when reading an IloModel.

```java
public IloAnyArray getTagListReadError()
```

This member function returns the XML tag list of the unparsed XML elements in cases where a problem occurs when reading an IloModel. The list is composed of the tags from the parent to the child elements.

```java
public const char * getWriteError()
```

This member function returns the name of the extractable called in cases where a problem occurs when reading an IloModel.

```java
public int getWritePrecision()
```

This member function returns the write precision for floats.

```java
public IloBool readExtractable(IloXmlReader reader,
                              IloXmlElement * element)
```

This member function tries to read all extractables from the XML element.

```java
public IloBool readModel(IloModel model,
                        istream & file)
```

This member function reads model from an XML stream.

```java
public IloBool readModel(IloModel model,
                        const char * fileName)
```

This member function reads model from the XML file fileName.

```java
public IloBool readModelAndSolution(IloModel model,
                                   const char * modelFileName,
                                   IloSolution solution,
                                   const char * solutionFileName)
```

This member function reads model and solution from their respective XML files, modelFileName and solutionFileName.

```java
public IloBool readSolution(IloSolution solution,
                            istream & file)
```

This member function reads solution from an XML stream.

---

**Note:** This member function only works if a model has already been serialized.

```java
public IloBool readSolution(IloSolution solution,
                            const char * fileName)
```

This member function reads solution from the XML file fileName.
public IloBool readSolutionValue(IloSolution solution,
   IloXmlElement * root,
   IloXmlReader reader)

This member function reads an IloSolution object from an XML element.

public void registerXML(IloTypeIndex index,
                       IloXmlInfo * xmlinfo)

This member function registers the serialization class of an extractable with a linked ID, usually its RTTI index. In write mode, the RTTI index is used to catch the correct serialization class.

In read mode, IloXmlInfo::getTagName is used to link the correct serialization class to the correct tag.

   IloXmlcontext context(env);
   context.registerXML(IloAllDiffI::GetTypeIndex(), new (env) IloXmlInfo_AllDiff(context));

public void registerXMLArray(IloXmlInfo * xmlinfo)

This member function registers the serialization class of an array of extractables with a linked ID.

   context.registerXMLArray(new (env) IloXmlInfo_SOS2Array(context));

public IloBool setWriteMode(IloInt mode)

This member function sets the write mode. The write mode can be set to NoUnknown or EvenUnknown. NoUnknown throws an exception if an attempt is made to serialize an unknown extractable. EvenUnknown writes a Unknown tag with the name of the extractable in a type attribute.

public void setWritePrecision(int writePrecision)

This member function sets the write precision for floats. By default, there is no rounding mode on an IloNum or an IloNumArray. You can also choose the no rounding mode with the IloNoRoundingMode constant.

public IloBool writeExtractable(const IloExtractableI * it,
                                IloXmlWriter writer,
This member function writes a specified extractable. It is used from the serialization class of an extractable to write a embedded extractable.

The IloOr object calls this method on its constrained vars.

See Also

IloXmlInfo::writeExtractable

public IloBool writeModel(const IloModel model,
                           const char * fileName)

This member function writes model to the file fileName in XML format.

public IloBool writeModelAndSolution(const IloModel model,
                                       const char * modelFileName,
                                       const IloSolution solution,
                                       const char * solutionFileName)

This member function writes model to the file modelFileName and solution to the file solutionFileName in XML format.

public IloBool writeSolution(const IloSolution solution,
                             const char * fileName)

This member function writes solution to the file fileName in XML format.

public void writeSolutionValue(const IloExtractable it,
                                const IloSolution solution,
                                IloXmlWriter writer)

This member function writes a specified extractable of a solution in XML. It is used from the serialization class of an extractable to write an embedded extractable.

See Also

IloXmlInfo::writeSolutionValue
## IloXmlInfo

**Category**: Class  
**Inheritance Path**: IloXmlInfo

**Definition File**: ilconcert/iloxmlabstract.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>IloXmlInfo(IloXmlContextI *, const char *)</td>
</tr>
<tr>
<td>public</td>
<td>IloXmlInfo()</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloBool</td>
<td>checkAttExistence(IloXmlReader, IloXmlElement *, const char *)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>checkExprExistence(IloXmlReader, IloXmlElement *, const char *, IloInt &amp;)</td>
</tr>
<tr>
<td>public IloXmlContextI *</td>
<td>getContext()</td>
</tr>
<tr>
<td>protected IloBool</td>
<td>getIntValArray(IloXmlReader, IloXmlElement *, IloIntArray &amp;)</td>
</tr>
<tr>
<td>protected IloBool</td>
<td>getNumValArray(IloXmlReader, IloXmlElement *, IloNumArray &amp;)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>getRefInChild(IloXmlReader, IloXmlElement *, IloInt &amp;)</td>
</tr>
<tr>
<td>public virtual const char *</td>
<td>getTag()</td>
</tr>
<tr>
<td>public static const char *</td>
<td>IloXmlInfo::getTagName()</td>
</tr>
<tr>
<td>protected IloNumVar::Type</td>
<td>getVarType(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>protected const char *</td>
<td>getVersion()</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>public virtual IloExtractableArray *</td>
<td>readArrayFromXml(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>readExtractable(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>public virtual IloExtractableI *</td>
<td>readFromXml(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>public virtual IloBool</td>
<td>readSolution(IloXmlReader, IloSolution &amp;).IloXmlElement *)</td>
</tr>
<tr>
<td>protected virtual IloExtractableI *</td>
<td>readXml(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>protected virtual IloExtractableArray *</td>
<td>readXmlArray(IloXmlReader, IloXmlElement *)</td>
</tr>
<tr>
<td>protected IloXmlElement *</td>
<td>setBoolArray(IloXmlWriter, const IloBoolArray)</td>
</tr>
<tr>
<td>public IloXmlElement *</td>
<td>setCommonArrayXml(IloXmlWriter, const IloExtractableArray *)</td>
</tr>
<tr>
<td>public IloXmlElement *</td>
<td>setCommonValueXml(IloXmlWriter, const IloExtractableI *)</td>
</tr>
<tr>
<td>public IloXmlElement *</td>
<td>setCommonXml(IloXmlWriter, const IloExtractableI *)</td>
</tr>
<tr>
<td>protected IloXmlElement *</td>
<td>setIntArray(IloXmlWriter, const IloIntArray)</td>
</tr>
<tr>
<td>protected IloXmlElement *</td>
<td>setIntSet(IloXmlWriter, const IloIntSet)</td>
</tr>
<tr>
<td>protected IloXmlElement *</td>
<td>setNumArray(IloXmlWriter, const IloNumArray)</td>
</tr>
<tr>
<td>protected IloXmlElement *</td>
<td>setNumSet(IloXmlWriter, const IloNumSet)</td>
</tr>
<tr>
<td>protected void</td>
<td>setVersion(const char *)</td>
</tr>
<tr>
<td>public void</td>
<td>setXml(IloXmlWriter, IloXmlElement *, const IloExtractableI *)</td>
</tr>
<tr>
<td>public IloBool</td>
<td>IloXmlInfo::writeExtractable(IloXmlWriter, IloXmlElement *, const char *)</td>
</tr>
<tr>
<td>public virtual void</td>
<td>writeSolution(IloXmlWriter, const IloSolution, const IloExtractable)</td>
</tr>
<tr>
<td>public void</td>
<td>IloXmlInfo::writeSolutionValue(IloXmlWriter, const IloSolution, const IloXmlElement *, const char *)</td>
</tr>
</tbody>
</table>
**Description**

The class `IloXmlInfo` allows you to serialize an `IloModel` or an `IloSolution` in XML.

**Constructors**

```cpp
public IloXmlInfo(IloXmlContextI * context,
```

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloSOS2Array)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloSOS1Array)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloSemiContVarArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloConstraintArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloRangeArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloNumVarArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloIntSetVarArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloNumExprArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloIntExprArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloBoolVarArray)</td>
</tr>
<tr>
<td>protected IloBool writeVarArray(IloXmlWriter, IloXmlElement *, IloIntVarArray)</td>
</tr>
<tr>
<td>public virtual int writeXml(IloXmlWriter, const IloExtractableArray *, IloXmlElement *)</td>
</tr>
<tr>
<td>public virtual IloBool writeXml(IloXmlWriter, const IloExtractableI *, IloXmlElement *)</td>
</tr>
<tr>
<td>public virtual IloBool writeXmlRef(IloXmlWriter, const IloExtractableI *, IloXmlElement *)</td>
</tr>
</tbody>
</table>
This constructor creates an instance of the handle class `IloXmlInfo` from a pointer to an instance of the undocumented implementation class `IloXmlContextI`.

**public IloXmlInfo()**

This constructor creates an empty instance of the handle class `IloXmlInfo`.

### Methods

**public IloBool checkAttExistence(IloXmlReader reader, IloXmlElement * element, const char * attribute)**

Given a specified attribute, this member function checks `element` to establish whether the attribute exists. If the attribute does not exist, this member function throws an exception.

You can use this member function to dynamically validate an XML element.

**public IloBool checkExprExistence(IloXmlReader reader, IloXmlElement * element, const char * attribute, IloInt & id)**

Given a specified attribute, this member function checks `element` to establish whether the attribute exists, fills the `id`, and checks in the XML context memory whether an object with this `id` exists.

You can use this member function to dynamically validate an XML element.

Example: in the `read` method of the `IloDiff`, check that the `IdRef` object is already serialized

**public IloXmlContextI * getContext()**

This member function returns the related `IloXmlContextI` of the constructor.

**protected IloBool getIntValArray(IloXmlReader reader, IloXmlElement * element, IloIntArray & intArray)**

This member function returns the contained `IloIntArray` in the XML element `element`.

**See Also**

`IloXmlReader::string2IntArray`

**protected IloBool getNumValArray(IloXmlReader reader, IloXmlElement * element, IloNumArray & numArray)**

This member function returns the `IloNumArray` in the XML element `element`.

**See Also**

`IloXmlReader::string2NumArray`
public IloBool getRefInChild(IloXmlReader reader,
   IloXmlElement * element,
   IloInt & id)

Given an XML element, this member function checks for the first value id or RefId in the element and its children.

public virtual const char * getTag()

This member function returns the related XML tag.

public static const char * getTagName()

This static member function returns the linked XML tag of this serialization class.

protected IloNumVar::Type getVarType(IloXmlReader reader,
   IloXmlElement * element)

This member function returns the type of an IloNumVar - IloFloat, IloInt, or IloBool - in the XML element element.

protected const char * getVersion()

This member function returns the version of the object.

public virtual IloExtractableArray * readArrayFromXml(IloXmlReader reader,
   IloXmlElement * element)

This member function reads an array of IloExtractableI* from the given XML element.

This is the method to specialize when writing a serialization class for an array of extractables.

public IloBool readExtractable(IloXmlReader reader,
   IloXmlElement * element)

This member function asks the XML context to read the IloExtractableI in the child element and then calls readFromXml to read the parent extractable.

public virtual IloExtractableI * readFromXml(IloXmlReader reader,
   IloXmlElement * element)

This member function reads an IloExtractableI from the given XML element. It asks the XML context to read the extractable in the XML child element using a call to readExtractable; it then calls readXml.

public virtual IloBool readSolution(IloXmlReader reader,
   IloSolution & solution,
   IloXmlElement * element)

This member function reads a variable for IloSolution from the XML element element.

protected virtual IloExtractableI * readXml(IloXmlReader reader,
   IloXmlElement * element)
This member function reads an `IloExtractableI` from the given XML element.
This is the method to specialize for each serialization class.

```cpp
protected virtual IloExtractableArray * readXmlArray(IloXmlReader reader, IloXmlElement * element)
```

This member function reads an array of `IloExtractableI*` from the given XML element.
It is called by the XML context. It first asks the XML context to read from XML child elements using a call to `readExtractable` and then calls `readArrayFromXml`.

```cpp
protected IloXmlElement * setBoolArray(IloXmlWriter writer, const IloBoolArray Array)
```

This member function creates an XML element containing the `IloBoolArray`.

**See Also**

`IloXmlWriter::IntArray2String`

```cpp
public IloXmlElement * setCommonArrayXml(IloXmlWriter writer, const IloExtractableArray * extractable)
```

This member function creates a XML element with the common header for `IloExtractable` arrays.

```cpp
public IloXmlElement * setCommonValueXml(IloXmlWriter writer, const IloExtractableI * exprI)
```

This member function creates a XML element with the given header for `IloExtractableI` from `IloSolution`.

```cpp
public IloXmlElement * setCommonXml(IloXmlWriter writer, const IloExtractableI * exprI)
```

This member function creates an XML element with the common header for `IloExtractableI`.

```cpp
protected IloXmlElement * setIntArray(IloXmlWriter writer, const IloIntArray Array)
```

This member function creates an XML element containing the `IloIntArray`.

**See Also**

`IloXmlWriter::IntArray2String`

```cpp
protected IloXmlElement * setIntSet(IloXmlWriter writer, const IloIntSet Array)
```

This member function creates an XML element containing the `IloIntSet`.

**See Also**

`IloXmlWriter::IntSet2String`

```cpp
protected IloXmlElement * setNumArray(IloXmlWriter writer, const IloNumArray Array)
```

This member function creates an XML element containing the `IloNumArray`.
This member function creates an XML element containing the IloNumArray.

**See Also**

*IloXmlWriter::NumArray2String*

protected IloXmlElement * setNumArray(IloXmlWriter writer, const IloNumArray Array)

This member function creates an XML element containing the IloNumArray.

**See Also**

*IloXmlWriter::NumArray2String*

protected void setNumSet(IloXmlWriter writer, const IloNumSet Array)

This member function creates an XML element containing the IloNumSet.

**See Also**

*IloXmlWriter::NumSet2String*

generated void setVersion(const char * version)

This member function sets the version of the object.

public void setXml(IloXmlWriter writer, IloXmlElement * element, const IloExtractableI * exprI)

This member function adds a name attribute and a ID attribute to the XML element.

public IloBool writeExtractable(IloXmlWriter writer, IloXmlElement * element, const IloExtractableI * extractableI, const char * attribute)

This member function writes an embedded extractable. Using the getId() method of the extractable, it adds an attribute with the ID in the XML element.

For example, used with IloDiff, this member function writes the expression and links it to the XML element via an IdRef attribute.

```c++
// using an IloDiffI* exprI:
writeExtractable(writer, element, (IloExtractableI*)exprI->getExpr1(), IloXmlAttributeDef::Expr1Id);
writeExtractable(writer, element, (IloExtractableI*)exprI->getExpr2(), IloXmlAttributeDef::Expr2Id);
```

**See Also**

writeExtractable

public virtual void writeSolution(IloXmlWriter writer, const IloSolution solution, const IloExtractable extractable)

This member function writes the specified extractable extractable from the IloSolution in XML format.

public void writeSolutionValue(IloXmlWriter writer, const IloSolution solution, IloXmlElement * element,
This member function writes an embedded extractable of a solution in xml. Using the getId() method of the extractable, it adds an attribute with the ID in the xml element.

For example, used with IloDiff, this member function writes the expression and links it to the XML element via an IdRef attribute.

See Also

* writeSolutionValue

protected IloBool writeVarArray(IloXmlWriter writer,
IloXmlElement * element,
IloSOS2Array array)

This member function writes an IloSOS2Array. It adds an attribute in the XML element element with the ID of array, serializes array, and, if necessary, serializes the IloSOS2s of array.

protected IloBool writeVarArray(IloXmlWriter writer,
IloXmlElement * element,
IloSOS1Array array)

This member function writes an IloSOS1Array. It adds an attribute in the XML element element with the ID of array, serializes array, and, if necessary, serializes the IloSOS1s of array.

protected IloBool writeVarArray(IloXmlWriter writer,
IloXmlElement * element,
IloSemiContVarArray array)

This member function writes an IloSemiContVarArray. It adds an attribute in the XML element element with the ID of array, serializes array, and, if necessary, serializes the IloSemiContVars of array.

protected IloBool writeVarArray(IloXmlWriter writer,
IloXmlElement * element,
IloConstraintArray array)

This member function writes an IloConstraintArray. It adds an attribute in the XML element element with the ID of array, serializes array, and, if necessary, serializes the IloConstraints of array.

protected IloBool writeVarArray(IloXmlWriter writer,
IloXmlElement * element,
IloRangeArray array)

This member function writes an IloRangeArray. It adds an attribute in the XML element element with the ID of array, serializes array, and, if necessary, serializes the IloRanges of array.
protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloNumVarArray array)

This member function writes an IloNumVarArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloNumVars of array.

protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloIntSetVarArray array)

This member function writes an IloIntSetVarArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloIntSetVars of array.

protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloNumExprArray array)

This member function writes an IloNumExprArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloNumExprs of array.

protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloIntExprArray array)

This member function writes an IloIntExprArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloIntExprs of array.

protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloBoolVarArray array)

This member function writes an IloBoolVarArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloBoolVars of array.

protected IloBool writeVarArray(IloXmlWriter writer,
   IloXmlElement * element,
   IloIntVarArray array)

This member function writes an IloIntVarArray. It adds an attribute in the XML
element element with the ID of array, serializes array, and, if necessary, serializes the IloIntVars of array.

Example using IloSos containing an IloIntVarArray:

   // Using an IloSOS1I* exprI;
   this.writeVarArray(writer,
      element,
      exprI->getVarArray(),
      IloXmlAttributeDef::IdRef);
This sample adds an IdRef attribute on the SOS XML element, creates an XML element containing the IloIntVarArray with the list of IloIntVar IDs, and creates a list of XML elements for the IloIntVars.

```csharp
public virtual int writeXml(IloXmlWriter writer,
                             const IloExtractableArray * extractable,
                             IloXmlElement * masterElement)
```

This member function writes the given IloExtractableArray in XML and adds it to the XML document of writer. This is the method to specialize when writing a serialization class.

```csharp
public virtual IloBool writeXml(IloXmlWriter writer,
                                  const IloExtractableI * exprI,
                                  IloXmlElement * masterElement)
```

This member function writes the IloExtractableI object exprI in XML and adds it to the XML document of the IloXmlWriter object writer.

```csharp
public virtual IloBool writeXmlRef(IloXmlWriter writer,
                                   const IloExtractableI * exprI,
                                   IloXmlElement * masterElement)
```

This member function writes the IloExtractableI object exprI in XML as a reference.
### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public IloXmlReader(IloEnv, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloXmlReader(IloXmlReaderI *)</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void addSerialized(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public void addSolutionSerialized(IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloBool checkRttiOfObjectById(IloTypeIndex, IloExtractableI *)</td>
<td></td>
</tr>
<tr>
<td>public IloBool checkRttiOfObjectById(IloTypeIndex, IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloBool checkTypeOfObjectById(IloTypeInfo, IloInt)</td>
<td></td>
</tr>
<tr>
<td>public IloBool checkTypeOfObjectById(IloTypeInfo, IloExtractableI *)</td>
<td></td>
</tr>
<tr>
<td>public void deleteAllocatedMemory(const char *)</td>
<td></td>
</tr>
<tr>
<td>public void deleteAllocatedMemory(char *)</td>
<td></td>
</tr>
<tr>
<td>public IloXmlElement * findElement(IloXmlElement *, const char *, const char *, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloXmlElement * findElementByTag(IloXmlElement *, const char *)</td>
<td></td>
</tr>
<tr>
<td>public IloInt getChildrenCardinal(IloXmlElement *)</td>
<td></td>
</tr>
<tr>
<td>public IloEnv getEnv()</td>
<td></td>
</tr>
<tr>
<td>public IloEnvI * getEnvImpl()</td>
<td></td>
</tr>
</tbody>
</table>
You can use an instance of IloXmlReader to read an IloModel or a IloSolution in XML format.

This constructor creates an IloXmlReader object and makes it part of the environment env.

The fileName is set to 0 by default.

This constructor creates an XML reader from its implementation object.

This member function tells the IloXmlReader that an object in the model has been serialized.
public void addSolutionSerialized(IloInt id)
This member function tells the IloXmlReader that an object of the solution has been serialized.

public IloBool checkRttiOfObjectById(IloTypeIndex RTTI, IloExtractableI * exprI)
This method checks the RTTI of the given object.

public IloBool checkRttiOfObjectById(IloTypeIndex RTTI, IloInt Xml_Id)
This method checks the RTTI of the object referenced by the identifier Xml_Id in the XML. This object must already be serialized.

public IloBool checkTypeOfObjectById(IloTypeInfo type, IloInt Xml_Id)
This method checks the TypeInfo of the object referenced by the id in the xml. This object must have been already serialized.

public IloBool checkTypeOfObjectById(IloTypeInfo type, IloExtractableI * exprI)
This method checks the TypeInfo of the given object

public void deleteAllocatedMemory(const char * pointer)
This member function frees the memory that has been allocated by the XML reader using, for example, the IloXmlWriter::Int2String member function.

public void deleteAllocatedMemory(char * pointer)
This member function frees the memory that has been allocated by the XML reader using, for example, the IloXmlWriter::Int2String member function.

public IloXmlElement * findElement(IloXmlElement * root, const char * tag, const char * attribute, const char * value)
This member function examines the XML element root to identify the XML child element denoted by tag, attribute, and value.

public IloXmlElement * findElementByTag(IloXmlElement * element, const char * tag)
This member function examines the XML element element to identify the XML child element denoted by tag.

public IloInt getChildrenCardinal(IloXmlElement * element)
This member function counts the number of child elements of the XML element element.

public IloEnv getEnv()
This member function gets the `IloEnv` of the object.

```java
public IloEnvI * getEnvImpl()
```

This member function gets the implementation of the `IloEnv` of the object.

```java
public IloXmlElement * getFirstSubElement(IloXmlElement * element)
```

This member function gets the first child in the XML element `element`.

```java
public IloBool getIntAttribute(IloXmlElement * element,
                               const char * attribute,
                               IloInt & value)
```

This member function checks the existence of `attribute` in the XML element `element` and converts it to an `IloInt`.

```java
public IloBool getNumAttribute(IloXmlElement * element,
                               const char * attribute,
                               IloNum & value)
```

This member function checks the existence of `attribute` in the XML element `element` and converts it to an `IloNum`.

```java
public IloAny getObjectById(IloInt id)
```

This member function gets the already serialized object of the given identifier `id`.

```java
IloDiff Diff{reader.getEnv(),
             IloExpr((IloNumExprI*)reader.getObjectById(IdExpr1)),
             IloExpr((IloNumExprI*)reader.getObjectById(IdExpr2)),
             reader.readAttribute(element, IloXmlAttributeDef::Name)};
```

The sample code creates an `IloDiff` from a XML element referencing its two expressions with the attributes `IdRef1` and `IdRef2`.

```java
public IloXmlElement * getRoot()
```

This member function gets the XML root, that is, the XML document without the header.

```java
public IloIntArray * getSerialized()
```

This member function gets the IDs of the serialized extractables and the unique IDs of the array of extractables that were serialized from the model.

```java
public IloIntArray * getSolutionSerialized()
```

This member function gets the IDs of the serialized extractables and the unique IDs of the array of extractables that were serialized from the solution.

```java
public IloBool isSerialized(IloInt id)
```
This member function checks whether the extractable with the ID `id` in the model has already been serialized.

```java
public IloBool isSolutionSerialized(IloInt id)
```

This member function checks whether an object is serialized using its `getId()` method.

```java
public IloBool openDocument()
```

This member function opens the XML document specified in the constructor or with the `setFileName` method.

```java
public const char * readAttribute(IloXmlElement * element, const char * attribute)
```

This member function returns the value of the `attribute` in the XML element `element`.

```java
public const char * readCData(IloXmlElement * element)
```

This member function reads the CDATA of the XML element `element`.

```java
public const char * readComment(IloXmlElement * element)
```

This member function returns the value of the comment in the XML element `element`.

```java
public const char * readData(IloXmlElement * element)
```

This member function reads the data of the XML element `element`.

```java
public const char * readText(IloXmlElement * element)
```

This member function returns the value of the text contained in the XML element `element`, independently of its origin (data or CDATA).

```java
public void setFileName(const char * fileName)
```

This member function sets `fileName` as the file from which to read the XML.

```java
public IloInt string2Int(const char * string)
```

This member function converts `string` into an `IloInt`.

```java
public IloIntArray string2IntArray(const char * string)
```

This member function converts `string` into an `IloIntArray`.

```java
public IloIntSet string2IntSet(const char * string)
```

This member function converts `string` into an `IloIntSet`.

```java
public IloNum string2Num(const char * string)
```

This member function converts `string` into an `IloNum`.

```java
public IloNumArray string2NumArray(const char * string)
```

This member function converts `string` into an `IloNumArray`. 
This member function converts string into an IloNumArray.
IloXmlWriter

Category: Class

InheritancePath

Definition File: ilconcert/ilowriter.h

### Constructor Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>IloXmlWriter(IloEnv, const char *, const char *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>IloXmlWriter(IloXmlWriterI *)</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Public void</th>
<th>addAttribute(IloXmlElement *, const char *, const char *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public void</td>
<td>addCData(IloXmlElement *, const char *)</td>
</tr>
<tr>
<td>Public void</td>
<td>addComment(IloXmlElement *, const char *)</td>
</tr>
<tr>
<td>Public void</td>
<td>addElement(IloXmlElement *)</td>
</tr>
<tr>
<td>Public void</td>
<td>addSerialized(IloInt)</td>
</tr>
<tr>
<td>Public void</td>
<td>addSolutionSerialized(IloInt)</td>
</tr>
<tr>
<td>Public void</td>
<td>addSubElement(IloXmlElement *, IloXmlElement *)</td>
</tr>
<tr>
<td>Public void</td>
<td>addText(IloXmlElement *, const char *)</td>
</tr>
<tr>
<td>Public IloXmlElement *</td>
<td>createElement(const char *)</td>
</tr>
<tr>
<td>Public void</td>
<td>deleteAllocatedMemory(const char *)</td>
</tr>
<tr>
<td>Public void</td>
<td>deleteAllocatedMemory(char *)</td>
</tr>
<tr>
<td>Public IloEnv</td>
<td>getEnv()</td>
</tr>
<tr>
<td>Public IloEnvI *</td>
<td>getEnvImpl()</td>
</tr>
<tr>
<td>Public const char *</td>
<td>getfileName()</td>
</tr>
</tbody>
</table>
You can use an instance of `IloXmlWriter` to serialize an `IloModel` or an `IloSolution` in XML.

**Constructors**

```cpp
public IloXmlWriter(IloEnv env, 
                    const char * rootTag, 
                    const char * fileName)
```

This constructor creates an `IloXmlWriter` object and makes it part of the environment `env`.

The `fileName` is set to `0` by default.

```cpp
public IloXmlWriter(IloXmlWriterI * impl)
```

This constructor creates a XML writer object from its implementation object.

**Methods**

```cpp
public void addAttribute(IloXmlElement * element, 
                         const char * attribute, 
                         const char * value)
```

This member function adds an attribute of the specified value to the XML element.

```cpp
public void addCData(IloXmlElement * element, 
                     const char * CData)
```

This member function adds a CDATA section to the XML element.

```cpp
public void addComment(IloXmlElement * element, 
                       const char * comment)
```

This member function adds comment to the XML element.
public void addElement(IloXmlElement * element)
This member function adds the XML element element to the end of the XML.

public void addSerialized(IloInt id)
This member function informs the XML writer object that the model object has been serialized.

public void addSolutionSerialized(IloInt id)
This member function informs the XML writer object that the solution object has been serialized.

public void addSubElement(IloXmlElement * element, 
IloXmlElement * subElement)
This member function adds a child element, subElement, to the XML element element.

public void addText(IloXmlElement * element, 
const char * text)
This member function adds text to the specified element.

public IloXmlElement * createElement(const char * element)
This member function creates an empty element with the given tag, element.

public void deleteAllocatedMemory(const char * pointer)
This member function frees the memory that has been allocated by the XML reader using, for example, the IloXmlWriter::Int2String member function.

public void deleteAllocatedMemory(char * pointer)
This member function frees the memory that has been allocated by the XML reader using, for example, the IloXmlWriter::Int2String member function.

public IloEnv getEnv()
This member function gets the IloEnv of the object.

public IloEnvI * getEnvImpl()
This member function gets the implementation of the IloEnv of the object.

public const char * getFileName()
This member function returns the name of the XML file

public IloXmlElement * getRoot()
This member function gets the root XML element of the XML document.

public IloIntArray * getSerialized()
This member function gets the IDs of the serialized objects of an IloModel.
public IloIntArray * getSolutionSerialized()
This member function gets the IDs of the serialized objects of an IloSolution.

public const char * Int2String(const IloInt number)
This member function converts the IloInt object number into a string, const char *.

public const char * IntArray2String(const IloIntArray intArray)
This member function converts the IloIntArray object intArray into a string, const char *.

public const char * IntSet2String(const IloIntSet intSet)
This member function converts the IloIntSet object intSet into a string, const char *.

public IloBool isSerialized(IloInt id)
This member function checks whether an object has been serialized.

public IloBool isSolutionSerialized(IloInt id)
This member function checks whether a solution object has already been serialized.

public const char * Num2String(const IloNum number)
This member function converts the IloNum object number into a string, const char *.

public const char * NumArray2String(const IloNumArray numArray)
This member function converts the IloNumArray object numArray into a string, const char *.

public const char * NumSet2String(const IloNumSet numSet)
This member function converts the IloNumSet object numSet into a string, const char *.

public void setfileName(const char * fileName)
This member function specifies fileName as the name of the XML file.

public IloInt string2Int(const char * string)
This member function converts string into an IloInt.

public void writeDocument()
This member function outputs the XML to the file specified in the constructor or using the setFileName method. If null, this member function outputs on the cout io.
# Index

<table>
<thead>
<tr>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>CannotExtractException 49</td>
<td>Exception 53</td>
</tr>
<tr>
<td>CannotRemoveException 51</td>
<td>ILO_NO_MEMORY_MANAGER 37</td>
</tr>
<tr>
<td></td>
<td>IloAbs 38</td>
</tr>
<tr>
<td></td>
<td>IloAbstraction 39</td>
</tr>
<tr>
<td></td>
<td>IloAdd 41</td>
</tr>
<tr>
<td></td>
<td>IloAlgorithm 42</td>
</tr>
<tr>
<td></td>
<td>IloAllDiff 59</td>
</tr>
<tr>
<td></td>
<td>IloAllMinDistance 62</td>
</tr>
<tr>
<td></td>
<td>IloAllNullIntersect 65</td>
</tr>
<tr>
<td></td>
<td>IloAnd 66</td>
</tr>
<tr>
<td></td>
<td>IloAny 70</td>
</tr>
<tr>
<td></td>
<td>IloAnyArray 71</td>
</tr>
<tr>
<td></td>
<td>IloAnyBinaryPredicate 73</td>
</tr>
<tr>
<td></td>
<td>ILOANYBINARYPREDICATENAME0 33</td>
</tr>
<tr>
<td></td>
<td>IloAnySet 75</td>
</tr>
<tr>
<td></td>
<td>IloAnySetVar 80</td>
</tr>
<tr>
<td></td>
<td>IloAnySetVarArray 84</td>
</tr>
<tr>
<td></td>
<td>IloAnyTernaryPredicate 87</td>
</tr>
<tr>
<td></td>
<td>ILOANYTERNARYPREDICATENAME0 34</td>
</tr>
<tr>
<td></td>
<td>IloAnyTupleSet 89</td>
</tr>
<tr>
<td></td>
<td>IloAnyTupleSetIterator 92</td>
</tr>
<tr>
<td></td>
<td>IloAnyVar 94</td>
</tr>
<tr>
<td></td>
<td>IloAnyVarArray 96</td>
</tr>
<tr>
<td></td>
<td>IloArcCos 99</td>
</tr>
<tr>
<td></td>
<td>IloArray 100</td>
</tr>
<tr>
<td></td>
<td>IloBarrier 104</td>
</tr>
<tr>
<td></td>
<td>IloBaseEnvMutex 107</td>
</tr>
<tr>
<td></td>
<td>IloBool 108</td>
</tr>
<tr>
<td></td>
<td>IloBoolAbstraction 109</td>
</tr>
<tr>
<td></td>
<td>IloBoolArray 111</td>
</tr>
<tr>
<td></td>
<td>IloBoolVar 114</td>
</tr>
<tr>
<td></td>
<td>IloBoolVarArray 117</td>
</tr>
<tr>
<td></td>
<td>IloBox 120</td>
</tr>
<tr>
<td></td>
<td>IloCard 124</td>
</tr>
<tr>
<td></td>
<td>IloCeil 125</td>
</tr>
<tr>
<td></td>
<td>IloColumnHeaderNotFoundException 414</td>
</tr>
<tr>
<td></td>
<td>IloCondition 126</td>
</tr>
<tr>
<td></td>
<td>IloConstraint 129</td>
</tr>
<tr>
<td></td>
<td>IloConstraintArray 132</td>
</tr>
<tr>
<td></td>
<td>IloConversion 135</td>
</tr>
<tr>
<td></td>
<td>IloCsvLine 415</td>
</tr>
<tr>
<td></td>
<td>IloCsvReader 419</td>
</tr>
<tr>
<td></td>
<td>IloCsvReaderParameterException 429</td>
</tr>
<tr>
<td></td>
<td>IloCsvTableReader 432</td>
</tr>
<tr>
<td></td>
<td>IloDeleterMode 139</td>
</tr>
<tr>
<td></td>
<td>IloDiff 140</td>
</tr>
<tr>
<td></td>
<td>IloDifference 438, 439</td>
</tr>
<tr>
<td></td>
<td>IloDisableNANDetection 143</td>
</tr>
<tr>
<td></td>
<td>IloDistribute 144</td>
</tr>
<tr>
<td></td>
<td>IloDiv 148</td>
</tr>
<tr>
<td>IloDuplicatedTableException</td>
<td>440</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>IloEmptyHandleException</td>
<td>149</td>
</tr>
<tr>
<td>IloEnableNANDetection</td>
<td>150</td>
</tr>
<tr>
<td>IloEndMT</td>
<td>151</td>
</tr>
<tr>
<td>IloEnv</td>
<td>152</td>
</tr>
<tr>
<td>IloEnvironmentMismatch</td>
<td>157</td>
</tr>
<tr>
<td>IloEqIntersection</td>
<td>158</td>
</tr>
<tr>
<td>IloEqPartition</td>
<td>159</td>
</tr>
<tr>
<td>IloEqUnion</td>
<td>160</td>
</tr>
<tr>
<td>IloException</td>
<td>161</td>
</tr>
<tr>
<td>IloExponent</td>
<td>163</td>
</tr>
<tr>
<td>IloExpr</td>
<td>164</td>
</tr>
<tr>
<td>IloExprArray</td>
<td>171</td>
</tr>
<tr>
<td>IloExtractable</td>
<td>174</td>
</tr>
<tr>
<td>IloExtractableArray</td>
<td>177</td>
</tr>
<tr>
<td>IloFastMutex</td>
<td>180</td>
</tr>
<tr>
<td>IloFieldNotFoundException</td>
<td>441</td>
</tr>
<tr>
<td>IloFilePathNotFoundException</td>
<td>442</td>
</tr>
<tr>
<td>IloFloatArray</td>
<td>184</td>
</tr>
<tr>
<td>IloFloatVar</td>
<td>185</td>
</tr>
<tr>
<td>IloFloor</td>
<td>187</td>
</tr>
<tr>
<td>IloFunction</td>
<td>188</td>
</tr>
<tr>
<td>IloGetClone</td>
<td>189</td>
</tr>
<tr>
<td>IloHalfPi</td>
<td>190</td>
</tr>
<tr>
<td>IloIfThen</td>
<td>191</td>
</tr>
<tr>
<td>IloIncorrectCsvReaderUseException</td>
<td>443</td>
</tr>
<tr>
<td>IloInfinity</td>
<td>194</td>
</tr>
<tr>
<td>IloInitMT</td>
<td>195</td>
</tr>
<tr>
<td>IloInt</td>
<td>196</td>
</tr>
<tr>
<td>IloIntArray</td>
<td>197</td>
</tr>
<tr>
<td>IloIntBinaryPredicate</td>
<td>199</td>
</tr>
<tr>
<td>ILOINTBINARYPREDICATENAME0</td>
<td>35</td>
</tr>
<tr>
<td>IloIntersection</td>
<td>444</td>
</tr>
<tr>
<td>IloIntervalList</td>
<td>445</td>
</tr>
<tr>
<td>IloIntervalListCursor</td>
<td>450</td>
</tr>
<tr>
<td>IloIntExpr</td>
<td>201</td>
</tr>
<tr>
<td>IloIntExprArg</td>
<td>204</td>
</tr>
<tr>
<td>IloIntExprArray</td>
<td>206</td>
</tr>
<tr>
<td>IloIntSet</td>
<td>209</td>
</tr>
<tr>
<td>IloIntSetVar</td>
<td>214</td>
</tr>
<tr>
<td>IloIntSetVarArray</td>
<td>218</td>
</tr>
<tr>
<td>IloIntTernaryPredicate</td>
<td>221</td>
</tr>
<tr>
<td>ILOINTTERNARYPREDICATENAME0</td>
<td>36</td>
</tr>
<tr>
<td>IloIntTupleSet</td>
<td>223</td>
</tr>
<tr>
<td>IloIntTupleSetIterator</td>
<td>226</td>
</tr>
<tr>
<td>IloIntVar</td>
<td>228</td>
</tr>
<tr>
<td>IloIntVarArray</td>
<td>233</td>
</tr>
<tr>
<td>IloInverse</td>
<td>237</td>
</tr>
<tr>
<td>IloIsNAN</td>
<td>240</td>
</tr>
<tr>
<td>IloIterator</td>
<td>241</td>
</tr>
<tr>
<td>IloLineNotFoundException</td>
<td>453</td>
</tr>
<tr>
<td>IloLog</td>
<td>243</td>
</tr>
<tr>
<td>IloMax</td>
<td>244, 454</td>
</tr>
<tr>
<td>IloMaximize</td>
<td>245</td>
</tr>
<tr>
<td>IloMember</td>
<td>246</td>
</tr>
<tr>
<td>IloMin</td>
<td>247, 455</td>
</tr>
<tr>
<td>IloMinimize</td>
<td>248</td>
</tr>
<tr>
<td>IloModel</td>
<td>249</td>
</tr>
<tr>
<td>IloMonotonicDecreasingNumExpr</td>
<td>254</td>
</tr>
<tr>
<td>IloMonotonicIncreasingNumExpr</td>
<td>255</td>
</tr>
<tr>
<td>IloMutexDeadlock</td>
<td>256</td>
</tr>
<tr>
<td>IloMutexNotOwner</td>
<td>257</td>
</tr>
<tr>
<td>IloMutexProblem</td>
<td>258</td>
</tr>
<tr>
<td>IloNot</td>
<td>260</td>
</tr>
<tr>
<td>IloNotMember</td>
<td>262</td>
</tr>
<tr>
<td>IloNullIntersect</td>
<td>263</td>
</tr>
<tr>
<td>IloNum</td>
<td>264</td>
</tr>
<tr>
<td>IloNumArray</td>
<td>265</td>
</tr>
<tr>
<td>IloNumColumn</td>
<td>267</td>
</tr>
<tr>
<td>IloNumColumnArray</td>
<td>270</td>
</tr>
<tr>
<td>IloNumExpr</td>
<td>272</td>
</tr>
<tr>
<td>IloNumExprArg</td>
<td>275</td>
</tr>
<tr>
<td>IloNumExprArray</td>
<td>277</td>
</tr>
<tr>
<td>IloNumFunction</td>
<td>280</td>
</tr>
<tr>
<td>IloNumToAnySetStepFunction</td>
<td>456</td>
</tr>
<tr>
<td>IloNumToAnySetStepFunctionCursor</td>
<td>465</td>
</tr>
<tr>
<td>IloNumToNumSegmentFunction</td>
<td>468</td>
</tr>
<tr>
<td>IloNumToNumSegmentFunctionCursor</td>
<td>475</td>
</tr>
<tr>
<td>IloNumToNumStepFunction</td>
<td>478</td>
</tr>
<tr>
<td>IloNumToNumStepFunctionCursor</td>
<td>484</td>
</tr>
<tr>
<td>IloNumVar</td>
<td>281</td>
</tr>
<tr>
<td>IloNumVarArray</td>
<td>287</td>
</tr>
<tr>
<td>IloObjective</td>
<td>293</td>
</tr>
<tr>
<td>IloOr</td>
<td>300</td>
</tr>
<tr>
<td>IloPack</td>
<td>304</td>
</tr>
<tr>
<td>IloPartition</td>
<td>307</td>
</tr>
<tr>
<td>IloPathLength</td>
<td>308</td>
</tr>
<tr>
<td>IloPathTransitFunction</td>
<td>313</td>
</tr>
<tr>
<td>IloPathTransit</td>
<td>314</td>
</tr>
</tbody>
</table>

ILOG CONCERT TECHNOLOGY 2.0 REFERENCE MANUAL 524
## INDEX

<table>
<thead>
<tr>
<th>IloPi</th>
<th>316</th>
</tr>
</thead>
<tbody>
<tr>
<td>IloPiecewiseLinear</td>
<td>317</td>
</tr>
<tr>
<td>IloPower</td>
<td>318</td>
</tr>
<tr>
<td>IloQuarterPi</td>
<td>319</td>
</tr>
<tr>
<td>IloRandom</td>
<td>320</td>
</tr>
<tr>
<td>IloRange</td>
<td>323</td>
</tr>
<tr>
<td>IloRangeArray</td>
<td>330</td>
</tr>
<tr>
<td>IloRound</td>
<td>337</td>
</tr>
<tr>
<td>IloScalProd</td>
<td>350, 351, 352, 353</td>
</tr>
<tr>
<td>IloSemaphore</td>
<td>354</td>
</tr>
<tr>
<td>IloSemiContVar</td>
<td>357</td>
</tr>
<tr>
<td>IloSemiContVarArray</td>
<td>361</td>
</tr>
<tr>
<td>IloSequence</td>
<td>365</td>
</tr>
<tr>
<td>IloSolution</td>
<td>368</td>
</tr>
<tr>
<td>IloSolutionArray</td>
<td>379</td>
</tr>
<tr>
<td>IloSolutionIterator</td>
<td>382</td>
</tr>
<tr>
<td>IloSolutionManip</td>
<td>384</td>
</tr>
<tr>
<td>IloSOS1</td>
<td>338</td>
</tr>
<tr>
<td>IloSOS1Array</td>
<td>341</td>
</tr>
<tr>
<td>IloSOS2</td>
<td>344</td>
</tr>
<tr>
<td>IloSOS2Array</td>
<td>347</td>
</tr>
<tr>
<td>IloSquare</td>
<td>385</td>
</tr>
<tr>
<td>IloSubset</td>
<td>386</td>
</tr>
<tr>
<td>IloSubsetEq</td>
<td>387</td>
</tr>
<tr>
<td>IloSum</td>
<td>388</td>
</tr>
<tr>
<td>IloTableConstraint</td>
<td>389</td>
</tr>
<tr>
<td>IloTableNotFoundException</td>
<td>487</td>
</tr>
<tr>
<td>IloThreeHalfPi</td>
<td>391</td>
</tr>
<tr>
<td>IloTimer</td>
<td>392</td>
</tr>
<tr>
<td>IloTwoPi</td>
<td>394</td>
</tr>
<tr>
<td>IloTypedList</td>
<td>395</td>
</tr>
<tr>
<td>IloTypedListManager</td>
<td>396</td>
</tr>
<tr>
<td>IloUnion</td>
<td>488, 489</td>
</tr>
<tr>
<td>IloXmlContext</td>
<td>497</td>
</tr>
<tr>
<td>IloXmlInfo</td>
<td>503</td>
</tr>
<tr>
<td>IloXmlReader</td>
<td>513</td>
</tr>
<tr>
<td>IloXmlWriter</td>
<td>519</td>
</tr>
<tr>
<td>Iterator</td>
<td>79, 213, 252, 380</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinearIterator</td>
</tr>
<tr>
<td>LineIterator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonLinearExpression</td>
</tr>
<tr>
<td>NotExtractedException</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator</td>
</tr>
<tr>
<td>operator-</td>
</tr>
<tr>
<td>operator &amp;&amp;</td>
</tr>
<tr>
<td>operator *</td>
</tr>
<tr>
<td>operator new</td>
</tr>
<tr>
<td>operator!</td>
</tr>
<tr>
<td>operator!=</td>
</tr>
<tr>
<td>operator+</td>
</tr>
<tr>
<td>operator/</td>
</tr>
<tr>
<td>operator==</td>
</tr>
<tr>
<td>operator&gt;</td>
</tr>
<tr>
<td>operator&gt;=</td>
</tr>
<tr>
<td>operator&gt;&gt;</td>
</tr>
<tr>
<td>optim.concert</td>
</tr>
<tr>
<td>optim.concert.extensions</td>
</tr>
<tr>
<td>optim.concert.xml</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sense</td>
</tr>
<tr>
<td>Status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableIterator</td>
</tr>
<tr>
<td>Type</td>
</tr>
</tbody>
</table>