EVENT ID: 267625
Event Name: Getting Started with IBM ILOG CPLEX Optimization Studio
Event Date: 2010-12-07

Moderator: Welcome to today's webcast, "Getting Started with IBM ILOG CPLEX Optimization Studio." You may submit your questions during the presentation using the Q&A tab at the bottom of your screen. The presenter will answer the questions during the presentation if possible. Otherwise, all questions will be answered at the end.

Our presenter today is Dr. Jeremy Bloom, Senior Product Marketing Manager, ILOG Optimization. Dr. Bloom has more than 25 years of business experience applying operations research and optimization. He joined IBM ILOG in 2007, serving as a technical account manager and later as Product Marketing Manager for the optimization product line. Before that, Dr. Bloom was a member of the research staff at the Electric Power Research Institute, a nonprofit consortium of U.S. and international utilities conducting advanced research for the power industry. While at the EPRI Dr. Bloom both managed and served as a technical contributor on a variety of programs, including planning, market assessment and asset management. Dr. Bloom taught operations research at Cornell University. He was educated in electrical engineering at Carnegie Mellon University and in operations research at the Massachusetts Institute of Technology.

At this time I'd like to hand the presentation over to Dr. Bloom so we can get started.

Jeremy?

Jeremy Bloom: Hello, everyone, and thank you for joining us today.

My subject today is to give you a brief introduction to the IBM ILOG CPLEX Optimization Studio, and these are things I'm going to cover today. We'll talk about the Optimization Programming Language, or OPL, which is a declarative language for specifying mathematical models for optimization purposes, and it's very similar to the mathematical notation that you would normally see for mathematical programs.

We'll also talk about scripting in OPL, and scripting is used for data processing and for flow control. It's procedural, a complement to the declarative OPL language. We'll talk of somewhat about the OPL interfaces that you can use to deploy applications. We'll speak briefly about the optimization engines that we have, the CPLEX Optimizer for mathematical programming and the CPLEX CP Optimizer for constraint programming and detailed scheduling. I'll talk at some length about how you develop OPL models, particularly how you interface OPL models with data sets. We'll talk briefly about testing and tuning OPL models, and we'll talk briefly about deployment.

And, as Wendy indicated, if you have questions during the session, please submit them through the Q&A and I'll try to answer them as we go along.

This diagram shows a broad overview of the IBM ILOG CPLEX Optimization Studio. There are basically two layers. There are the model development tools, which consist of an integrated development environment, and the OPL programming language. And then there are two optimization engines. The mathematical programming optimization engine is called the CPLEX Optimizer and implements the Simplex method, the Barrier method and Mixed Integer programming, and constraint programming engine is called CPLEX CP Optimizer. For those of you who are not familiar with constraint programming, it's a technology, a different technology for doing optimization that's built on artificial intelligence principles. And we find that constraint programming and mathematical programming are often complementary, since the strengths of one complement the weaknesses of the other.

Now, the basic interface between the model development tools and the optimization engines is the ILOG Concert Technology, which is a set of APIs in C++, .NET and Java, and the OPL programming language is built on top of the Concert Technology. You also have the ability to use the optimization engines through their callable libraries separately from OPL. And, in addition to OPL, the CPLEX Optimization Studio provides connectors to a number of other platforms, including Microsoft Excel, MATLAB, AMPL and Python. And, although we won't talk about it today, we also have an enterprise deployment platform for decision support applications built on optimization called IBM ILOG ODM Enterprise. And I'll talk briefly at the end about some of the features of IBM ILOG ODM Enterprise, but that really deserves a separate webinar in itself.

Now, the major purpose of using CPLEX Optimization Studio is to simplify and speed up the development and deployment of optimization-based applications. Typically we find two kinds of people involved in development of these applications. There are optimization experts, who are trained in operations research and know about mathematical modeling and how to formulate mathematical models that represent business problems, and there are frequently software developers involved, who are responsible for the data interfaces, the user interfaces and any other computer science-type components that are needed in order to develop an optimization-based decision support application.

By using CPLEX Optimization Studio, especially the integrated development environment, which is built on OPL and on the ILOG Concert Technology, you can dramatically reduce the time to market for developing and deploying optimization applications, because you can easily integrate the work of both the software developers and the optimization experts.

Let me continue with a brief overview of the components of the CPLEX Optimization Studio.

We have an integrated development environment. For those of you who have done programming perhaps are familiar with the Eclipse platform, and the integrated development environment in CPLEX Optimization Studio is built on Eclipse, so it would be familiar to you. It has a graphical editor for entering and editing OPL models. It allows you review data and solutions in tabular form. It allows you to use menus and buttons to run and debug your optimization models, and it has online help for the OPL syntax.

In addition to the integrated development environment, there's the OPL language itself. This is a compact language that represents optimization problems. It has advanced data types that are specifically designed to implement optimization models, and it supports all kinds of mathematical constraint programming models, including linear and quadratic programs with both quadratic and linear objectives and constraints. You can use real or integer variables. For scheduling problems we have special syntax that enables you to specify scheduling problems in terms of interval decision variables with precedence and resource constraints, which I'll talk briefly about later. You can connect OPL models to external data sources, including relational databases and Excel spreadsheets, and we provide OPL Script for procedural data processing and for flow control for things like iterative problem solving, and I'll show you some examples of that, as well.

Let me talk a little bit about OPL Script. It's an implementation of JavaScript, and it's a full-featured scripting language that can be called inside OPL model files. And, as I said, it's a procedural complement to the declarative OPL syntax. For data preprocessing and for solution postprocessing and for flow control you can develop procedural OPL scripts that will walk your model through the various steps. Some of the applications that are for OPL Script include computing model data from raw input data. I'll show you an example of that a little bit later. You can compute results data from the solution values of the optimization problem. You can solve a sequence of related problems. And you can implement decomposition strategies such as column generation or custom cutting planes for mixed integer programming.

OPL has a number of interfaces that you can use. They're built on the ILOG Concert Technology, and they link with the CPLEX Optimizers for mathematical programming and constraint programming. And you can use these interfaces to embed OPL models and scripts into applications. So you can build an application in one of the standard programming languages like C++ or Microsoft .NET or Java and you can call OPL to specify your model. So you don't have to generate the matrix separately. You can use the OPL language to actually create the mathematical model and to link it to data sources. So this is going to be a very efficient way of doing your development in a declarative language, a high-level declarative language, and implementing in a standard programming language. And this is another example of the relationship between the optimization experts and the software developers.

Now, the CPLEX Optimization Studio allows you easy access to all of the mathematical programming and constraint programming engines that are available in the CPLEX Optimization Studio. For mathematical programming you can solve linear and quadratic programs with the CPLEX Simplex or Barrier Optimizers. You can solve mixed integer linear and quadratic programs with the CPLEX Mixed Integer Optimizer. And you can solve quadratically constrained problems with CPLEX Barrier and Mixed Integer Optimizers. For constraint programming you can solve detailed scheduling problems or hard combinatorial optimization problems with the CPLEX CP Optimizer. And the CPLEX Optimization Studio provides access to all of the CPLEX Simplex, Barrier, Mixed Integer and CP Optimizer algorithm settings, so anything that you could do through the command line or the APIs you can do through CPLEX Optimization Studio.

Let's talk a little bit about how we develop models using OPL. A model is a data-independent abstraction of a business problem, and it specifies mathematically the key relationships in the problem. And typically those relationships involve choices or decisions that have to be made. There are goals or key performance indicators that indicate the quality of the decision. What's the best set of decisions among all the decisions I have possible? And not all decisions or combinations of decisions are allowed, so there are constraints that represent limits or requirements or recipes that indicate how the decisions are linked to resources.

OPL allows you to write a mathematical representation of your problem that's separate from your data. So in OPL a project is defined as one or more model files and one or more data files, and within a project a run configuration represents a problem instance with a model and a data file, and you could have multiple run configurations within an OPL project. So, for example, you might want to have a test data set, a small instance of the problem you want to solve for debugging purposes, and then a production data set, where the data source is from a relational database. And all of those could be handled within the CPLEX Optimization Studio, as I'll explain in a few minutes.

Let's start with an example, and this is a very simple production example. For those of you have either the trial or the full version of CPLEX Optimization Studio, you can load this example from the Examples tab on the File menu, and it's called production.prj. And in this particular instance a manufacturer has some products he wants to sell. And the products could be made in either of two fashions. They can be made inside the factory, where production consumes scare resources and there's a cost per unit to manufacture the products, or they can be ordered from outside the factory. In that case there is no resource usage but there's a higher cost per unit to purchase the products. And there's a demand constraint that all the demand for the products must be satisfied, and the goal is to minimize cost. Now, this is a simple, single period implementation of the production problem. Within the examples in CPLEX Optimization Studio there are multiple period versions of this problem, as well, and there are problems that include more complex production processes, but this is a very simple example.

So the first step in specifying an OPL model is to declare the data, and in this particular instance the first things we declare are the index sets, which represent the set of products and the set of resources. And these are defined as sets of string. That is, the strings are the names of the products and the resources. Now, on the right-hand side of the equal sign you see three dots. That's the OPL notation that says you read this data from a data file. And so you don't have to specify the data within the model. The data is read in from a data file.

The next thing you would specify is the recipe. How many units of each resource are required for each product? This is an array with two indexes. One is the set of products and the other is the set of resources. And it's a real number. In this case we use the notation "float" to indicate a real number. So each entry in this array represents the amount of resource that's required to produce one unit of each product. And there's a capacity available for each resource, or an amount of each resources that's available. This is, again, a real number array capacity that's indexed over the resources. And there's amount of product that's required, which is an array indexed by the products called "Demand," and again it's a real number. And then there are the costs per unit for production inside and outside of each product. So this is the data that is required to solve this problem.

Now, the product in this case could be anything. It could be jewelry. And so what we have here is a representation of the data file that goes with this particular model when I'm producing jewelry. So in this case there are two products, rings and earrings, and there are two resources, gold and diamonds. And the recipe calls for three units of gold and one diamond to produce each ring or two units of gold and two diamonds for each earring. The amount of resources available, there's 130 units of gold and 180 diamonds available. The demand is 100 rings and 150 pairs of earrings. And the inside cost to produce the products, it's $250 to produce rings and $200 to produce earrings. Outside cost is $260 for rings and $270 for earrings. Now, these are the way the data would be specified in a data file that would be associated with the model to create a problem instance.

But I could also have an alternative data file where the products are pasta. So in this case there are three products -- kluski, capellini and fettuccine -- and there are two resources, flour and eggs. And the consumption here is that producing a unit of kluski requires half a unit of flour and 0.2 units of eggs, and so forth for each of the other products. There's a capacity of 20 units of flour and 40 units of eggs. There's a demand of 100, 200, 300 for each of the three products -- kluski, capellini and fettuccine. And there's an inside and an outside cost to produce each product.

Now, the important point here is that the model is the same. The model does not depend on the data. So I specify the model in generic form over these abstract index sets with parameters for consumption, capacity, demand, inside cost and outside cost. And then I can specify two decision variables, which unfortunately don't show on this slide. But the two decision variables are the inside production of each product and the outside production of each product, and they are specified as nonnegative real numbers, so we have a simple linear program here.

The next slide shows the actual specification of the model. The first thing we have is the objective function. The objective function is to minimize the total cost of meeting the demand. So it's the sum over each product in the product set of the inside cost of producing that product times the inside production plus the outside cost of producing that product or acquiring that product times the outside production. And it's subject to two constraints. The first constraint is a capacity constraint on the resources. It says for each resource in the set of resources the sum over all the products of the consumption of the resource for that product times the inside production has to be less than or equal to the capacity available for that resource. The second constraint is the demand constraint. It says for each product in the set of products the inside production plus the outside production has to be at least as great as the demand. So it's a simple linear programming problem. And, as I said, if you want to see this in CPLEX Optimization Studio you can find it as production.prj in the Examples library in CPLEX Optimization Studio.

Now, this slide shows you what the IDE, the integrated development environment, in CPLEX Optimization Studio, looks like. So let's walk through this rather complicated picture and you'll see how each component is used. Again, this is an implementation that's based on Eclipse, which is a well-known integrated development environment for a number of programming languages, including Java, and we've adapted it for OPL and for CPLEX Optimization Studio.

So, in the upper left-hand corner, I have the first box is the Project Navigator. And in the Project Navigator I have a list of all the projects that are present in this work space. In this case there's only one. It's the production project. And then for the production project I have a list of the models, the .mod files. I have a list of the data files, the .dat files. And I have a set of run configurations, and the run configurations represent the pairing of a model with a data file, and I can run each of the run configurations separately.

The next box below that is the Problem Browser. The Problem Browser lists all of the data objects and the solution objects that are present in this model. And I can use that to find out information about each object. I can inspect each object. And I can show them, as I'll show you in a minute, I can show them in a tabular form.

On the right side I have an outline. The Outline shows the structure of this particular project, and it shows all of the files that are associated with the project.

In the center I have an editing area, and I'm showing in this particular window the text of the OPL .mod file, the model file, for the production example that we just talked about. And in this window I can enter my OPL statements, and there are context-sensitive help and syntax checking as you enter the OPL code. I can also display the data files. And if you look at the top tab very carefully you'll see that there's also a data file present, but because this is not an interactive presentation I can't actually show you what the data file looks like, but it's very simple, as I indicated before. And we'll talk a little bit more about data files in a minute.

Finally, at the bottom there's an Output area where there are a number of panes, so you can, for example, see the solution that's produced by the Optimization Engine. This particular picture shows some of the statistics of the solution. On the left-hand side there is a diagram showing the upper and lower bounds. I'll show you that in more detail in a minute. I'm sorry, on the right, and on the left are some of the statistics of the solution process. You can also see error messages in this area, and you can see, as I said, the output from scripts.

Let's talk in a little bit more detail about the OPL programming language and some of its capabilities. OPL basically has three primitive data types: integers; real numbers, which are called floats; and strings. But there are also the ability to create a number of complex data structures, as well. Most importantly are what are called "tuples," and tuples are data structures with multiple fields in it. So in this particular example I'm defining a tuple called a Route, and a Route represents an origin, which is a string, represented by a string; a destination, which is represented by a string; and a product. So this might arise in a transshipment model where I have products that I need to ship between various cities.

Now, I can also create sets in OPL, and these are typically used for indexing decision variables and parameters. So, there are two ways to specify a set. The first one is to use the keyword "setof," and in parentheses is the type, the data type of the set, so this is a set of strings. And in this case this set is called "cities." And, as I said before, the equal sign with the three dots on the right represents a command to read data from a data file. The other way to specify a set is to use the braces, and so what I'm saying here is I have a set of routes, a set of tuples that are of type Route, and I'm reading those routes, that set of routes, from a data file, as well. So, both of these are equivalent. You can use either syntax in either context.

I can also specify arrays, and I can have one-dimensional arrays or multidimensional arrays. So in this case I'm representing an array of real numbers, of floats, called "cost," and it's specified over the set of routes. And, again, I'm reading the data from a data file.

Now, I can also create, in addition to parameters, I can also create decision variables. These are created, or these are solved, in the optimization engine. There are three types of decision variables. There are integers, there are real numbers and there are booleans, that is, 0/1 variables used for indicator variables. And you can index the decision variables on either a range or index set. So in the first example here I have a decision variable called shipments. It's a real number, and it's nonnegative. That's what the "float+" means. And it's indexed over all of the tuples in the set of routes. And it's restricted to the range of 0 to the capacity of the route. In other words, the decision variable has both an upper and a lower bound in this case.

The second example is an integer decision variable called "nTrucks." It's indexed on a range from one to the maximum number of truck types. And it's, as I said, it's an integer variable.

And the final example is a boolean variable, a 0/1 variable, called "open" which is indexed on a set of cities. So it might be used in a facility location problem to indicate whether a facility is open or not in a particular city.

I can also create what are called decision expressions, which are combinations of decision variables and parameters. In this case a decision expression is "shippingCost," which is the total cost of shipments. It's the sum over all the tuples in the route set of the cost of the route times the shipment amount along that route. Decision expressions are used in the objective function of the model, the goals of the model, and we define decision expressions primarily for readability, but also if you use ODM Enterprise you can define multiple decision expressions and use goal programming in ODM Enterprise to optimize over a set of decision expressions simultaneously.

Now, one of the most important features of the OPL language is the ability to specify sparse data structures, data structures where there's a lot of empty or zero cells. So let's take an example to see how this works.

I'm going to create a set of cities, which are strings, and there are four cities: PIT, FRA, WIN and LIF, which I suppose represent airport codes or something like that. So there are four cities. So if I wanted to specify a set of routes among these cities and I didn't have a sparse data structure I'd have to specify 12 routes. Assuming that there's no route from a city back to itself, each city is connected to three other cities, so a total of 12. And you can see that if I have more cities the number of elements in the route set specified that way would grow quadratically. It would grow as the square of the number of the cities. So it could get very large if you had, for example, 50 state capitals.

But frequently you find in optimization problems the route set is fairly sparse. That is, not all cities are -- not all pairs of cities are connected by routes. So another way to deal with this is to explicitly specify the routes. And I do that by defining a tuple called "root," which has an origin and a destination and a product. And then I can read in from a data file or I can construct in data preprocessing the set of routes that are present in the network. In this case, there are three routes. One goes from FRA to WIN and ships rings. Another one goes from FRA to LIF and it ships rings. And the third one goes from PIT to WIN and it ships rings, as well. So, in this case, I only have three routes in my network, and so I only have to read in three data elements to represent the routes, rather than 12, if I don't use a sparse representation.

I can also use set operations in OPL to calculate subsets or slices of the sets. So if, for example, I want to know what all the origins are, after I've read in the routes I can then take the origins out of each route table and collect them together into a string, a set of strings, or a set of origins.

And I could also define decision variables over the routes, so I can define the shipments over the routes. And this particularly helps me reduce the size of my matrix that I'm going to be trying to solve, because I only have entries in the decision variables when there is a route present in the set of routes. So if I had a non-sparse representation I'd have to have 12 decision variables in this case, whereas since there are only three real routes in this system I only need three decision variables. And, again, you can see that as I get a very large model this ability to take advantage of sparsity can significantly reduce both the amount of data processing and also the amount of computation in the solver.

Let me go back again and discuss this notion of data and model separation. As I said, in CPLEX Optimization Studio and in OPL generally data access is separate from the model. So, I have a model file. I specify the model file in terms of parameters that re not known at the time that the model is created. So in the model file I have a set of routes, a set of route tuples that indicate the routes, and, again, the equal sign with the three dots on it indicates that I'm going to be reading that data from a data file.

And then I have several possible origins for the data. If I'm just doing testing I might have a simple text file, which we call a .dat file in OPL. I'm just reading in the routes, in this case, the three routes that you saw on the previous slide. But when I'm finished with the testing I might want to actually work with a much larger data set, and I might want to, rather than explicitly generating the data as text, I might want to read it from a database file. And so in the .dat file I can have commands which select the data from a data set, and in this case you're actually using SQL to create the data.

And the point here is that I can switch between the test data file and the link to the production database with no changes to the model. The model stays identical. Once I've debugged it with a test data set I can move to production without requiring any changes to the model. And that's done through the run configuration capability of CPLEX Optimization Studio.

In general, there are a couple of different ways you can read data from external sources. One of the most common ones, especially for desktop applications, is to read the data from an Excel spreadsheet. So this illustrates how a data file would be configured in order to read data from a spreadsheet. I use an OPL command called "SheetConnection" to create a data source called "sheet," and I identify the sheet in the file system by its file name. And then I can read the routes from the sheet using a SheetRead command, and I can specify the location of that data either by an explicit range in the database or I can -- I'm sorry, in the spreadsheet, or I can have a named range in the spreadsheet and read the data from a named range.

I can do the same trick with a database, except instead of using SheetConnection I use another OPL command called "DatabaseConnection," or "DBConnection." In this case I'm creating a database connection called "db." It's an Access database. And the file name is specified. And then once I have established that database connection I can then use a DBRead command to issue an SQL command to the database to return the data. So in this case I'm reading the routes from a database -- from a table called "Routes," and I can also use more sophisticated SQL queries in this case to capture all of the cities in the model by reading both the origins and the destinations from the Routes table and creating their union.

This slide illustrates this process in database connectivity in a little bit more detail. I won't repeat what I just did. But I want to point out that you can not only read data from a database but you can write data back to the database. And so on the right side here is the syntax for creating a result from a solution of an optimization model in OPL and then putting a data reference in the .dat file that uses a DBUpdate command with an SQL command to write the data back into the database.

And I have a similar capability with spreadsheets. In this case on the right I illustrate how to write data from a database. I've got a result, in this case the solution to the optimization model. The decision variable is called "shipments," and I can use a SheetWrite command to write the data back to arrange in the spreadsheet.

So, the data connections are independent of the model. They are implemented through a .dat file, which can either create -- either include data itself as text or commands to connect to a database or a spreadsheet and specify the location of the data where the data is to be either read from or written.

Now, I mentioned before that in addition to the ability to solve mathematical programs OPL also allows you to solve constraint programs. And for constraint programs we've developed some specialized syntax in OPL to enable you to specify detailed scheduling problems in a compact fashion.

The first special syntax is to a define a decision variable called an interval. And an interval, in this case the name of the interval is "job," and there's one of them for each tuple in a set of tasks. And I can specify, for example, the earliest start time and the latest end time for that interval, and I can specify the length or the size of that interval. So this is a decision variable that's specified as an interval.

I can also use what are called cumulative functions to model resource usage. So, for example, suppose I have a limited number of workers. I need to make sure that I don't have more people working at any moment in time than I have available. I can create a cumulative function called "workerUsage" which is the sum over all the tasks of a pulse, and a pulse is a time function which is 0 outside the interval and 1 inside the interval. And I have a pulse for each job, for each interval called "job" in the task set. And I calculate the total number of workers by the summing over the pulses for these jobs. And I can write a constraint that says I cannot exceed the maximum number of workers available.

And I have other scheduling-specific constructs. For example, I can create sequences of intervals, and I can specify within those intervals the first interval that must occur, the last interval that must occur, and I can specify ordering among the intervals in that sequence. I can also use various kinds of intensities to determine the usage of resources during an interval. And there are many, many other features that I'm not going to talk about today because, in fact, detailed scheduling deserves a whole webcast of its own. You can find some additional information on our website, including two white papers on how to do scheduling using CPLEX Optimization Studio.

Okay, I've got a few questions. I'm going to take some of those questions right now. Question is can you export the optimization algorithm to Java code? The answer is yes. I'll discuss that in a minute.

How do I handle a problem where the number of combinations is high due to many variables and level for each, especially if they are discrete variables requiring a large design space sampling? That's a very complicated question that I don't think I can answer today, but we can get back to you on that. But OPL has features that enable you to place index sets in various dimensions, and so you can create rather complex index structures that enable you to officially handle the data.

Are there any reference examples of projects for stochastic integer programming in CPLEX? I believe there are. You'll have to look in the advanced set of examples in the CPLEX Optimization Studio example library.

Is it possible to create a dynamic set during iteration in which the elements in a set can change based on some conditions? You can do that using OPL script. Again, this is really a question that's beyond the scope of this presentation, but I'll try to get you an answer on that.

What about goal programming? Goal programming is primarily a feature of ODM Enterprise, which has the ability to create multiple objectives and to either set minimum levels for each objective or to weight the objectives dynamically based on user inputs. Within CPLEX Optimization Studio we don't generally support goal programming, although you could potentially use a script to create it.

Does it mean that OPL scripts can be used to substitute programming languages like C++ or C#? The answer is generally yes. OPL scripts are somewhat more restricted in terms of their syntax and their functionality than C++ or C#, so in many cases we find people moving to C++ or C# when they have a complex set of data processing or complex flow controls that they want to do. But for many instances OPL Script is sufficient.

How to retrieve objective values at different times along optimization [inaudible] in OPL. Again, this question is really beyond the scope of our discussion today, but it is possible using scripts to interrogate the solution as the solution process proceeds.

Are connections to Excel only for data interchange, or can Excel formulas be used to perform calculations alongside ILOG as part of an optimization application? The access to Excel typically is static which means that if you're going to be doing calculations in Excel you want those calculations to be done either before the data is read or after the data is written to the Excel spreadsheet. It is possible using .NET to construct more complicated kinds of interactions with Excel spreadsheets, but, again, that’s a fairly advanced topic that I'm not going to discuss about.

Does OPT -- I assume that means OPL -- support integer second order cones? The answer is yes.

Is it possible to solve nonlinear optimization problems using ILOG CPLEX? Right now CPLEX solves linear and quadratic programs. You could build an algorithm based on something like sequential linear programming to solve linear programs, but there is no native support in the CPLEX Optimizers for CPLEX for nonlinear optimization.

And the final question I have is is the IDE available on Linux? Not today. The IDE is only available on Windows. However, all the other features of CPLEX Optimization Studio, including OPL, the Concert APIs and the solvers are available on Linux.

Okay, let me continue on with my presentation, and I just wanted to show you a couple of examples of using scripts. This is the OPL script language. So, one of the examples you might want to deal with is data preprocessing. You have data. In this case the data is the longitude and latitude of various cities and the freight rate for shipping products and you want to calculate the cost of shipping a longer route.

So, the way you do that is to use a script. In this case the script is indicated by the keyword "execute" followed by a block of code enclosed in braces. And in this case the name of the script is INITIALIZE. And I define some local variables, the difference in longitude and latitude. And then I loop through the routes using a for loop, and this is very similar to anybody who's used Java or JavaScript would recognize this one. I calculate the differences between the longitude and the latitude of the organization and the destination city in that route, and then I calculate the distance between the two cities as the sum of the squares of the differences in longitude and latitude, take the square root, multiply it by the freight rate for that route, and I calculate the cost. So this is an example where the data is not exactly in the format or the units that you need, and you can use an OPL script to calculate the data.

The next example is a flow control example. So, in this particular case I've got a model and I want to solve it repeatedly for different freight rates for shipping a product. So, the flow control script is indicated by the keyword "main," and the block of code is enclosed in braces after that. There are several steps to this. The first is I initialize the project by identifying the model, the model definition and the data elements. And I also read the base value of the freight rate from the data file.

Then I have a for loop, again familiar to those you who program in C++ or Java. I loop over, in this case, 10 times through this example. I reset the model. I change the freight rate in the data file. I generate the model and solving it using CPLEX, and then I write out the objective function value. And then I increment the freight rate by multiplying it by 1.1, and I do this 10 times. So this might be an example where I'd want to do a sensitivity analysis of a different set of ranges for the -- or a range of different values for the freight rate.

Let me move on now to the next part of the presentation, where I'm going to talk a little bit about tuning and testing of OPL models. And I'll cover a number of topics. The first one is I'll talk about how you can use the debugging features of CPLEX Optimization Studio in model development. Then I'll talk about some execution tracing. And finally I'll talk about the ability to export data for testing.

So, let me show you how you would see a tabular view of your data or your decision variables in the integrated development environment. So in this particular case I've selected the decision variable Inside from the Problem Browser, which is on the lower left, and you can see it's highlighted. And in this particular example this is a multi-time period version of the production problem we talked about earlier.

So the Inside production has two indices. It's the product and the period. And you can see in the table on the right that I've got a row for each combination of the product and the time period. And then the first column after the indexes shows you the value of the decision variable. The second one shows you its reduced cost. And the third one shows you its range -- that is, the range over which this variable can change without changing the optimal basis. So this is the kind of sensitivity analysis you might want to do. And it lays out as a table, so that you can actually see the individual values and identify where they are in the solution matrix.

Another function that I can do is I can expand constraints. So if I have complex index structures on my constraints it's sometimes hard to see how OPL is creating the actual constraint structure. So what this tabular view allows you to do is it allows you to expand the constraints, and in this case I have a constraint on resources, which is the constraint we talked about before, except it has a multi-period version. So the first two columns are the indices, the resource and the period. And then the third column is the actual constraint that you -- that OPL generates for that resource and that time period. And the next column shows the slack, and the final column shows the dual value for that constraint. And I can expand or collapse the levels in the constraint so that I can see generically what they look like. At the lowest level I can see each individual product or each individual period, but I could also just look at each individual product across all periods.

I have another question here. Could I please elaborate on how CPLEX can be used for modeling rapid deployment for distributed real-time computation? Again, that's a fairly complex topic that I'm not going to deal with in detail today. However, just want to point out that with the CPLEX Optimization Studio I have the ability to use the IDE to do both detailed debugging and also looking at the individual processes which are involved in generating and solving the model, and so some of those features can be used to deal with distributed real-time computations. And I'll defer to get you an answer from one of our experts to discuss that in more detail.

The next slide shows my ability to trace execution. So, this is a situation in which I'm solving a mixed integer program. The graph on the right shows the upper bound and the lower bound, and it shows them converging together. The table on the left shows some of the statistics of the actual solution process. I can pause execution in the middle of the solve process. I can examine the current solution. And I can determine whether I want to continue or stop where I am, so I can actually look at the model and the solve process and determine whether I'm close enough and how much more time I'm willing to expend to solve the model.

Let me talk briefly about some of the deployment options you have. There are two key deployment options. One is I actually can compile the OPL model and create code that is compatible with C++, Java or Visual Basic. And if I'm not going to change the model, then the application can be deployed as a C++, Java or .NET application. I can also use the OPL model in an interpreted fashion, which takes a little bit more time, but if I'm going to be changing the model frequently then I don't want to compile it, and so I can use the application with interpreted model files. And, again, I can deploy it as a C++, Java or .NET application.

So, the way we recommend doing rapid application deployment is to develop the model using the CPLEX Optimization Studio in the IDE, maintaining model and data separation, and then use the OPL interfaces, that is, the APIs, in C++, Java or the .NET languages, to create an application. I can integrate external data, again, by reading through the OPL data file or by creating objects in the API. And then I can deploy the application and use it wherever I want.

You can link with the solvers. You can use all the mathematical programming solvers in the CPLEX Optimizers or the constraint programming CPLEX CP Optimizer.

Another option is to use IBM ILOG ODM Enterprise. This is a deployment platform that provides much of the functionality that business users need to see in an optimization application, and it allows you through menus and so forth to create business user-friendly applications without a lot of coding. It has flexible tabular and chart views of both data and solutions that are provided out of the box. They require no coding. You can access multiple data sources. You can have multiple objectives and do goal programming.

The business user can actually modify data constraints and business rules. They don't have to understand the underlying optimization model in order to do that. They can relax constraints. It's very useful when there are problems that could be infeasible, especially if the infeasibility is due to data rather than a structural infeasibility, because they can relax constraints and diagnose what's causing the infeasibility, so you never end up with a situation where I can't get a solution.

You can create and compare what-if scenarios, which is usually very important in establishing the credibility of the model for the business user. You can have multiple users collaborating on the same decision support application. And you can support decision process with operations research and information technology experts or planners or reviewers. And when you have a very large computationally intensive solve process you can actually put the solve process on a server separately from the database servers or the application server so that you can segregate the computational burden to a specific server. So these are some of the advantages of using IBM ILOG ODM Enterprise.

Just finally I'd like to review some of the features of CPLEX Optimization Studio, and it's a powerful, graphic, integrated development environment that allows you to enter models and data; organize projects with multiple models and multiple data sources; allows you to debug these applications; you can visualize data and solutions; and you can control the optimization using the parameters, the settings for the optimization engines. You have an expressive OPL modeling language and scripting language that can specify linear, quadratic and constraint programming problems with continuous or integer decision variables, and specialized syntax for specifying detailed scheduling problems.

You have access to the best-in-class solver technologies, the IBM ILOG CPLEX Optimizers and the IBM ILOG CPLEX CP Optimizer. You have access to multiple data sources, including databases and spreadsheets. And you have OPL interfaces that can accelerate the application deployment.

If you're interested in downloading a trial, I've given you the address on our website where you can download the trial. You can look at the model library, which ships with the product, and you can try out the various examples. We also offer product training, and we have a couple of ways of getting additional assistance, including our website and email or telephone. So this information you can use to get access to what we have.

Thank you for your attention.

Let me see if there are any additional questions and we'll end up here.

I think I've answered all the questions, so if you have any additional questions please enter them now and I will try to answer them.

Okay, I have a few more questions here.

Okay, hang on, let me get the -- okay, so let me get some of these questions. Can you export the optimization algorithm to Java code? You cannot export the optimization algorithm. You can export the model and the data to Java and you can call the optimizers.

Is there access to cloud computing? Not at present, but it's something we're considering implementing.

Once deployed in Java, do you need a license to run it? You need a license for the Optimization Studio. Once you have the Optimization Studio you can deploy it.

How can we use initial solutions? Well, CPLEX has the ability to specify in advance starting solution, and you can certainly access that through the IDE in the CPLEX Optimization Studio.

Is there an ODBC connection with a 64-bit version? I don't know the answer to that question. We'll try to get the answer for you.

We have a license to use CPLEX callable and Concert library. Do we need an additional license to try CPLEX Optimization Studio? Depends on the license that you have right now, but generally speaking we've been upgrading all of our customers who have previous CPLEX licenses to CPLEX Optimization Studio. But you'd have to talk to your specific rep to determine what the requirements are.

How do you decide that matrices are too sparse, when to use tuples or some [inaudible] construct other than matrices? That's something that comes with experience in working with a model and with OPL. So we can give you some training and some guidance on that. There's actually a white paper on our website called "Efficient Modeling with CPLEX Optimization Studio" which gives you some guidance on that one.

Can you do dynamic programming to optimize the functions for time and optimize the example -- for example, the commodity versus time in the solution? You can do dynamic programming if you can formulate it as a linear program, and there are some standard ways of converting some dynamic programs to linear programs that you can use.

Can CPLEX use multicore hardware? Yes, CPLEX has a multicore version which is a standard part of the distribution package of CPLEX Optimization Studio.

Can Optimization Studio generate an MPS file for the matrix? Yes.

I have an n by n matrix where the indices are -- can I have an n by n matrix where the indices are decision variables? No, that's not a linear problem, and, as I said, the CPLEX Optimization Studio handles linear and quadratic problems.

Can we import a model written in new math programming into OPL? Generally not. There's no automatic way to do that. You'd have to translate it.

How can we store and see each of multiple solutions with the same best objective function value or next best values? I believe you could probably use a script to do that. You would create an array to store each of the solution values. But, again, this is sort of an advanced topic, so we'd have to get some expertise to discuss your specific requirements there.

And I think that's all the questions that I have. Let me hit refresh again to see.

White paper, please. The white papers, you can, if you go to the optimization website on IBM.com, we have a library and there are white papers that you can download there.

For complex modeling and solving with lots of flow controls and sensitivity analyses, should someone who wants to use CPLEX come with a mindset that he or she needs to learn a programming language like C++ or C# since OPL scripts seem to have a limit? It's been my experience that if you want to do complex solves chances are you need to know a programming language like C++ or C#. It is possible to do a great deal with OPL and OPL Script, but OPL and OPL Script have their limits, and at some point it becomes much more effective to use a more standard programming language.

If we have AIMMS development licenses do we also have CPLEX development licenses automatically? That depends on the purchase that you've made from AIMMS, so you have to contact either AIMMS or an ILOG sales rep to determine that.

I have experienced OPL to be slow when I use OPL Script to loop in a set of tuple. Any reason for that? Like any programming languages, there are inefficiencies in the way scripts are formulated, and you have to look very carefully at how the script is being implemented. My recommendation is you can either contact our help, our experts, and maybe they'll take a look at it and let you know.

Let me see if there are any other questions here.

What's the difference between the development and the deployment license? Development licenses are used for developing an application. Once you have the application completed and you want to use it in an environment, in a production environment, you need a deployment license to do that.

Can CP and MP be used together for solving problems? Yes, and there are in fact a number of examples in the example library where they're used together.

Your phone number again, please. Let's see. Can we get back there? There's my email address.

Any expectation when Linux or Mac OSID available? I can't say.

Let's see. I think that's it on the questions.

So, again, thank you very much for your time. We went a little bit over, but hopefully you got something out of it. Again, there's my email address. You can also contact us through our website, which is the optimization website on IBM.com, and we'll be happy to answer your questions.

Wendy, anything from you?

Moderator: No, that's it. Thank you, Jeremy, and thank you to all the attendees, as well. We know you all have busy schedules, and we appreciate you spending the time with us. If you didn't get a direct answer to your question, we'll do our best to follow up with you one on one once the presentation is over.

And that concludes our presentation for today. We did record this presentation, and everybody will receive a URL for an on-demand viewing, and also I believe from the screen that you're on now you can download the presentation.

So, thank you. That concludes our presentation for today. Goodbye.