Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

THE EXTEND SIMULATION ENVIRONMENT

David Krahl

Imagine That, Inc.
6830 Via Del Oro, Suite 230
San Jose, CA 95119, U.S.A.

ABSTRACT

The Extend modeling environment provides an integrated structure for building simulation models and developing new simulation tools. This environment supports simulation modelers on a wide range of levels. Model builders can use Extend's pre-built modeling components to quickly build and analyze systems without programming. Simulation tool developers can use Extend's built-in, compiled language, ModL to develop new modeling components. All of this is done within a single, self-contained software program that does not require external interfaces, compilers, or code generators.

1 INTRODUCTION

Over the last decade, there has been a convergence in the simulation industry. Simulation languages have become easier to use, often adding a user interface layer similar to that traditionally found in simulators. Simulators have added functionality to the point where their power and flexibility rival that of traditional languages. Because of this, it has become difficult for an individual to determine the advantages of one product over the others based strictly on a feature comparison.

In this confusing marketplace, Extend stands out as a product whose basic design provides a combination of unparalleled ease of use, power, and extensibility (Krahl 1999). It exists as:

- A stand-alone simulation tool which can be used to create complex discrete event and continuous models without programming.
- A simulation authoring package where model interfaces can be easily created, without programming, to enhance productivity and ease of use.
- A development environment for building sets of custom components. This programming environment allows the modeler to create a simulator for a specific industry.

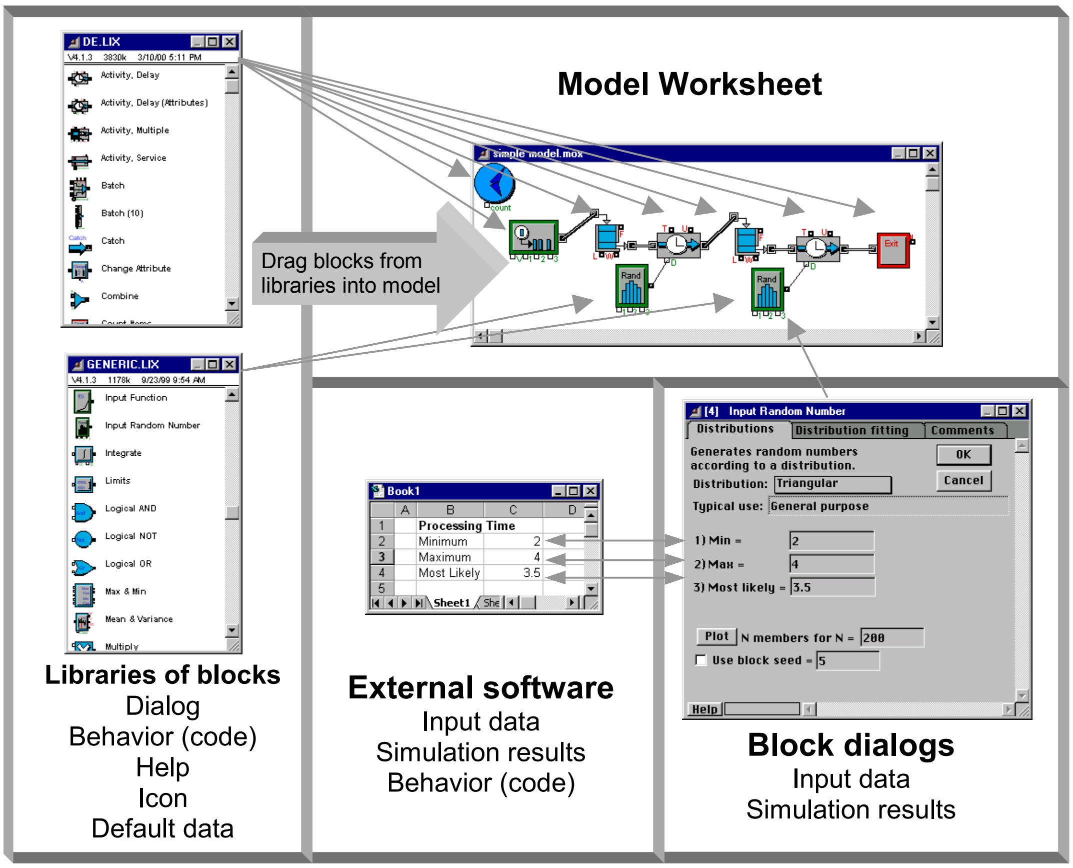
2 EXTEND'S MODELING ENVIRONMENT

Before looking into how Extend can be used to build models, it is helpful to understand the Extend modeling environment (Imagine That, Inc. 1998)

Figure 1 illustrates the overall structure of an Extend model. Blocks are pulled from libraries into the model. The user then fills out the dialog of the block, and, if desired, creates links to external applications. Blocks supply behavior (or code), help, icon, dialog, and default data for each step in a process. Each block in the Extend model references this information from the library. Blocks on model worksheets contain any data entered by the modeler and links to external programs.

Extend models are constructed with library-based iconic blocks. Each block describes a calculation or a step in a process. Block dialogs are the mechanism for entering model data and reporting block results. Blocks reside in libraries. Each library represents a grouping of blocks with similar characteristics such as Discrete Event, Plotter, Electronics, or Business Process Reengineering. Blocks are placed on the model worksheet by dragging them from the library window onto the worksheet. The flow is then established between the blocks.

There are two types of logical flows between the Extend blocks. The first type of flow is that of "items", which represent the objects that move through the system. Items can have attributes and priorities associated with them. Examples of items include parts, patients, or a packet of information. The second type of logical flow is "values", which will change over time during the simulation run. Values represent a single number. Examples of values include the number of items in queue, the result of a random sample, and the level of fluid in a tank.



	A	B	C	D
1		Processing Time		
2		Minimum	2	
3		Maximum	4	
4		Most Likely	3.5	
5				

Generates random numbers according to a distribution.
Distribution: Triangular
Typical use: General purpose
1) Min = 2
2) Max = 4
3) Most likely = 3.5
N members for N = 200
Use block seed = 5

Figure 1: Extend Modeling Structure

Each block has connectors that are the interface points of the block. Figure 2 shows the connector symbols for the value and item connectors.

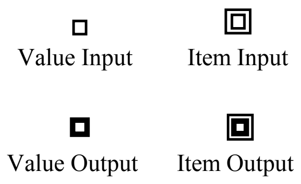


Figure 2: Value and Item Connectors

Connections are lines used to specify the logical flow from one connector to another. Double lines represent item connections and single lines represent value connections. The concept of value connections in addition to item connections is unique to Extend. Contemporary simulation software requires that a function be written whenever a simulation input is based on a value from another point in the simulation. In Extend, this type of logic is performed without programming of any type. More importantly, the logic of the model is visible to anyone examining the model structure.

Figure 1 illustrates the relationship between the libraries, blocks, worksheet and any external programs (such as Excel or a DLL) which may be linked to Extend. It also shows the visual nature of an Extend model. Note that the Input Random Number blocks can be clearly identified as providing the delay (D connector) for the activities.

3 SINGLE SERVER, SINGLE QUEUE EXAMPLE

The following example is of a single server, single queue system. For the purpose of illustration, the model of a car wash will include one wash bay and one waiting line. The model for this car wash is shown in Figure 3.

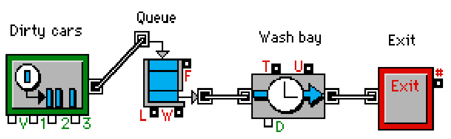


Figure 3: A Single Server Single Queue Model

The block on the far left is a Generator block that periodically creates items (in this case dirty cars). Following this is a Queue, FIFO block that holds the cars until requested by the next block. The wash bay is represented by the Activity Delay which has a limited capacity of one processing unit and delays the car for a fixed amount of time. The last block in the model is an Exit block that removes the cars from the system.

Suppose that the processing time for the wash bay is better represented by a specific random distribution. This can be modeled by connecting the output of an Input Random Number block to the delay connector (labeled "D") on the Activity Delay block as in Figure 4. Every time a car enters the wash bay, a new processing time is requested from the Input Random Number block. For each request, the Input Random Number block generates a new processing time from the specific random distribution defined in the block's dialog.

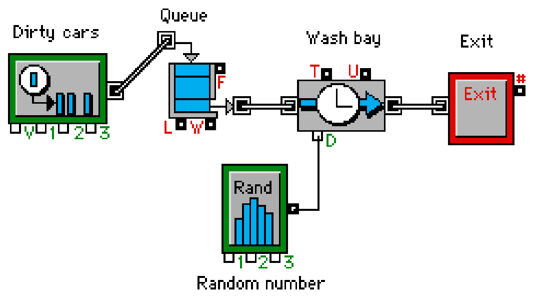


Figure 4: A Model with Random Process Times

3.1 Graphical Output

A Discrete Event Plotter graphically displays model metrics (values). In this example (Figure 5), the Plotter will graph the contents of the Queue (the number of dirty cars waiting in line) over time. Here the length connector (L) on the Queue FIFO is connected to an input on the Plotter.

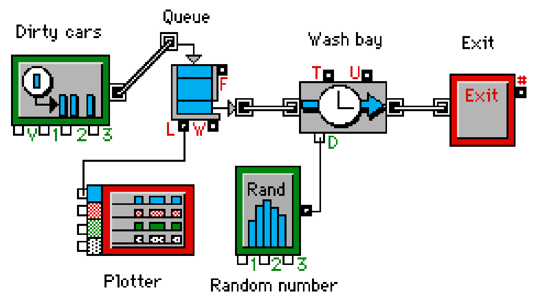


Figure 5: Discrete Event Plotter Added to Model

3.2 Attributes

Assume that the car wash offers two types of washes, basic and deluxe, and that the processing time is dependent upon the type of wash requested. To differentiate between the two different types of wash requests, attributes are used. The Set Attribute Block adds an attribute called "type" to each car. It randomly sets the value of this attribute to 0 (basic) or 1 (deluxe) using another Input Random Number Block as shown in Figure 6. As the dirty cars leave the queue and enter the wash bay, the Get Attribute block reads the "type". A Conversion Table block converts this number to a value representing the mean processing time for washes of that type. To generate the sample from a normal distribution, the mean value can then be fed into the Input Random Number block (Figure 6).

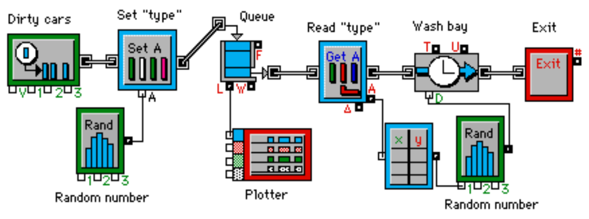


Figure 6: Setting "Type" Attribute

3.3 Resources

Because the activities in Extend already include an implied resource (they have a capacity and items will wait for the activity if it is occupied), resources are required only if the same resource is used at multiple places in the model. In order to effectively illustrate the use of resources in this model, it is necessary to add the additional detail of customers paying for the car wash and the process of drying the cars. In this case, an attendant is required to collect the money for the wash and to dry the car. Additional blocks are added to represent these two additional activities.

As shown in Figure 7, a Resource Pool block represents the available attendants. The resource is allocated to the item in the Queue, Resource Pool and released in the Release Resource Pool. Another new block in this example is the Activity Multiple. This block allows multiple items to be delayed at the same time. Here, it represents the drying of the cars because multiple cars can be dried at a time.

Another new feature illustrated by this example is the use of "named connections". The labels "Wash" and "Dry" are used to make connections rather than having long lines appear across the model (Figure 7). This feature is essential in large models where a large number of connected lines would quickly become cumbersome.

3.4 Activity Based Costing

Now that there is a basic model of the car wash, the model can be enhanced to calculate the average cost of washing each car. The following information is available:

- Attendants are paid $8.50 an hour.
- Cars use $1.25 in soap
- Electricity and water used by the wash bay cost $1.50 per minute.

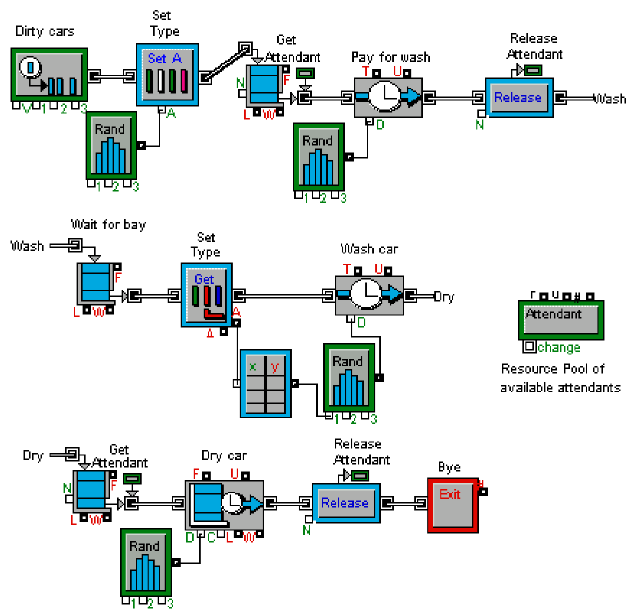
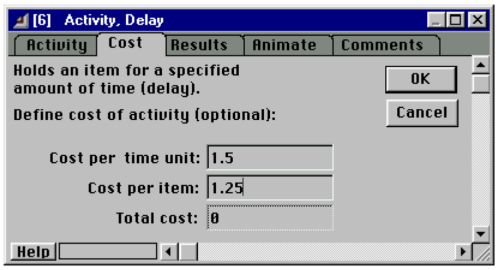


Figure 7: Modeling Resources

The cost of the attendant is defined within the Resource Pool block and the cost of the soap, water, and electricity in the Activity Delay block (Figure 8). As the model is run, the accumulated cost of each vehicle is automatically calculated and stored in an attribute. The Cost By Item Block can be added to read the cost attribute, sort the items by an attribute (such as the "type" attribute) and report on the throughput, total cost, and average cost by type of wash requested. The Cost Stats Block be added to report the total cost generated in each of the blocks.


[6] Activity, Delay

Activity | Cost | Results | Animate | Comments

Holds an item for a specified amount of time (delay).

Define cost of activity (optional):

OK
Cancel

Cost per time unit:	1.5
Cost per item:	1.25
Total cost:	0

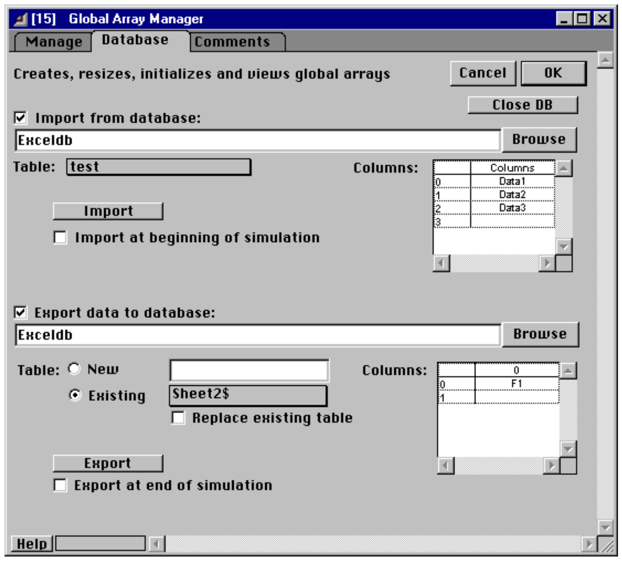
Help

Figure 8: Cost Tab of Activity Delay Block

3.5 Communication with Other Applications

The term interprocess communication (IPC) describes the act of two applications communicating and sharing data with one another. This feature allows the integration of external data and applications into and out of Extend models. Automatic communication between Extend and other applications can take one of three forms:

- "Paste-Link" where the information is automatically updated between Extend and Excel. Setting up this type of communication only requires copying the value in one application (Extend or Excel) and selecting paste-link in the other application. This produces a "live" link that updates whenever the value in the host application changes. These updates even occur when the model is running and can be used to display data or graphs in Excel that "animate" while the simulation model is running.
- Blocks that utilize the IPC functions to communicate directly with other applications. Imagine That's IPC library allows models to send data to, get data from, and execute macros within other applications, including Excel spreadsheets. These blocks can respond to simulation events and traverse the spreadsheet.
- ODBC (Open DataBase Connectivity). Extend can access database information through ODBC. As with all of Extend's interprocess communication features, this is available both on a block level (accessible with no programming required) and on an API level within Extend's ModL programming environment. Figure 9 illustrates the Database import and export functionality of the Global Array Manager block.
- Embedded objects (ActiveX or OLE). These retain their native user interface, but reside with the Extend model worksheet or blocks. All of the features and the interface of the embedded application are directly available within Extend. Figure 10 shows a bar chart object from the GraphicsServer toolkit embedded in an Extend block. This is part of the Bar Chart block that graphs the level of an input value during the simulation run.


[15] Global Array Manager

Manage | Database | Comments

Creates, resizes, initializes and views global arrays

Cancel OK

Close DB

[✓] Import from database:

Exceldb Browse

Table: test

Columns:

	Columns
0	Data1
1	Data2
2	Data3
3	

Import

☐ Import at beginning of simulation

[✓] Export data to database:

Exceldb Browse

Table: ○ New [✓ ● Existing] Sheet2$

☐ Replace existing table

Columns:

	0
0	F1
1	

Export

☐ Export at end of simulation

Help

Figure 9: ODBC Import/Export

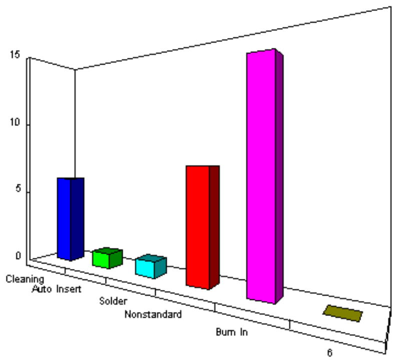
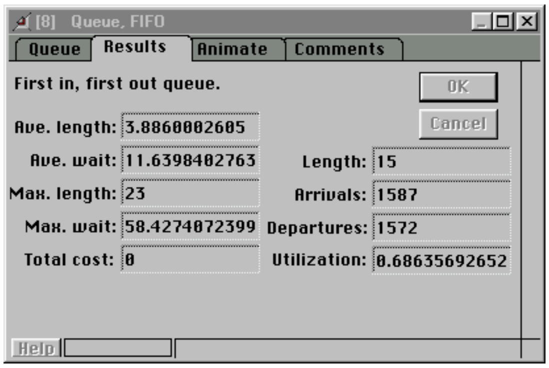


Figure 10: Embedded Bar Chart Object

3.6 Model Results

Once the simulation run has completed, the results of the simulation are reported within the blocks, displayed on plotters, sent to reports, and exported to other applications. Double clicking on each block reveals the information collected from the simulation run. For example, double clicking on the Queue, FIFO block opens a dialog showing the following information about the state of the block:

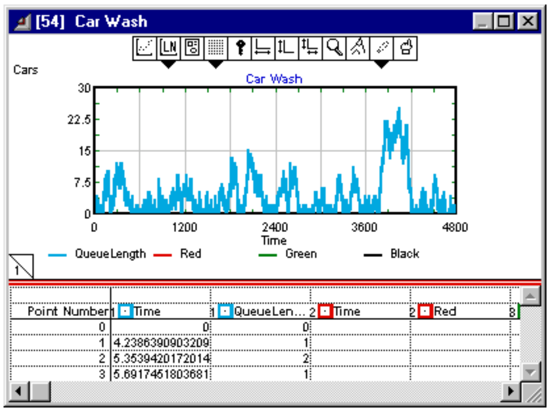


First in, first out queue.

Ave. length: 3.8860082605
Ave. wait: 11.6398402763
Max. length: 23
Max. wait: 58.4274072399
Total cost: 0
Length: 15
Arrivals: 1587
Departures: 1572
Utilization: 0.68635692652

Figure 11: Dialog of Queue FIFO

The Plotter block shows the number of items stored in the Queue, FIFO over time in both graphical and tabular format:



Point Number	1 Time	1 QueueLen...	2 Time	2 Red	8
0	0	0			
1	4.2386390903209	1			
2	5.3539420172014	2			
3	5.6917451803681	1			

Figure 12: Plot of Queue Length

Simulation results may be stored in a table, plotted, cloned to a different area of the worksheet, exported to another program such as a spreadsheet or database, displayed in an animation, or even used to control some aspect of the outside world through external device drivers.

3.7 Data Analysis

Extend offers a number of methods for analyzing both input and output data. These range from internal analysis features to built-in interfaces with other applications.

An interface to distribution-fitting programs is provided to aid users in selecting the appropriate statistical distributions based on empirical data collected in the field.

In addition, sensitivity analysis can be performed to determine how sensitive a system is to changes in specific input parameters. For example: to determine how sensitive the car wash is to changes in the inter-arrival time of dirty cars, sensitivity analysis can be performed on the inter-arrival mean parameter of the Generator Block. By selecting the inter-arrival time dialog item and choosing Sensitize Parameter from the Edit menu, the change in the parameter value from one run to the next is defined. Simulation parameters such as the number of runs and simulation end time can be specified in the Simulation Setup dialog under the Run menu. By cycling through different inter-arrival times for the dirty cars and comparing the results from the different runs, an understanding of how sensitive the car wash is to the arrival rate of dirty cars can be obtained.

The Statistics library helps users to collect and analyze output data. Blocks from the Statistics library automatically gather data from the appropriate blocks and calculate confidence intervals.

3.8 Optimization

Until now, optimization of a simulation model has been a process of trial and error. Extend's Evolutionary Optimizer block employs powerful "enhanced evolutionary" algorithms to determine the best possible model configuration. Using a drag and drop interface, parameters that can be varied and pertinent results are entered into the Optimizer block. These parameters are used in an equation that defines the objective function. When the model is run, the Optimizer block generates alternatives and locates the statistically best configuration. Unlike external optimizers, Extend's optimization is well integrated into the program. For example, when the optimization process is complete, model parameters are automatically set to the optimal configuration. In addition, because the optimizer has been implemented in a block, the source code is available for examination and modification.

4 CUSTOMIZING EXTEND

The above discussion illustrates the highly graphical and interactive nature of Extend. However, Extend can also take the shape of the modeled system. Interfaces, components, and graphics can be created which tailor the model to a specific application area.

The most visible aspect of a custom model is the user interface. By modifying an existing interface or creating a new one, the simulation modeler is able to create a model which can be exercised by someone more familiar with the system than with the simulation tool. Models can be built that fit naturally into the conceptual framework of the person using the model. The following sections will describe some of the tools provided in Extend that facilitate customization.

4.1 Animation

Animation is a powerful presentation and debugging tool that can greatly increase model clarity. In Extend, animation icons moving from block to block represent the flow of items through the system. Users can choose from a number of icons provided with Extend or create their own in an external drawing package.

For example, adding animation to show cars traveling from block to block in the car wash model is done by selecting the appropriate icon in the Animate tab of the Generator block. From here, the picture representing all of the items created by the Generator is defined. Each block that the items pass through has the capability of changing the item's animation icon. For example, every item exiting the Generator block can be represented with a picture of a dirty car. As the items pass through the wash bay, the Activity Delay block changes each item's animation picture to a clean car, providing visual cues of how the items are changing as they progress through the model.

In addition, custom animation can be added to display pictures and text, level indicators, and pixel maps.

An interface also exists between Wolverine Software's animation package, Proof Animation™. Activities, Resources, Generators, and Exit blocks each have specific functionality to send information to the Proof animation during simulation execution (Wolverine Software Corporation 1995). Additional animation features in Proof can be accessed in Extend through the Proof library of blocks. This allows Extend modelers to easily utilize the industry's most sophisticated animation package.

4.2 Hierarchical Modeling

Extend provides unlimited layers of hierarchy, created using simple menu command. Hierarchy allows models to be subdivided into logical components or sub-models, represented by a single descriptive icon. Double clicking on the hierarchical block opens a new window displaying the sub-model. This greatly simplifies the representation of a model and allows the user to hide and show model details as appropriate for the target audience.

In the car wash model (Figure 7), as detail was added, the number of blocks increased. As a result, the representation of the model has become slightly encumbered.

Using hierarchy, the model can be represented by the system's most basic elements (Figure 13):

- the arrival of dirty cars
- the queue of dirty cars waiting for availability of the wash bay
- the wash bay
- the departure of clean cars.

By selecting a group of blocks and choosing Make Selection Hierarchical from the Model menu, a section of the model can be encapsulated within a hierarchical block. Extend's hierarchy fully encapsulates the enclosed block and does not require the renaming of variables and connections. All of the connection names within the hierarchical block. This allows multiple instances of identical hierarchical blocks in the same model (Pidd and Castro 1998). The hierarchical blocks can be copied within a model or saved to a library to be used again in other models. The icon for the hierarchical block can be modified by using the built-in icon editor or by importing an existing picture. Figure 13 shows the car wash model with hierarchical blocks representing some of the basic elements of the car wash. While the representation of the model is more intuitive and simple than Figure 7, all of the detail of the model can still be accessed by double clicking on any of the hierarchical blocks to display the underlying sub-model.

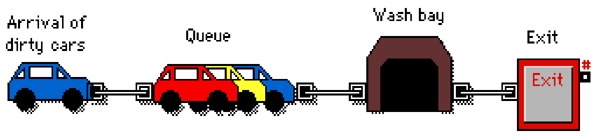
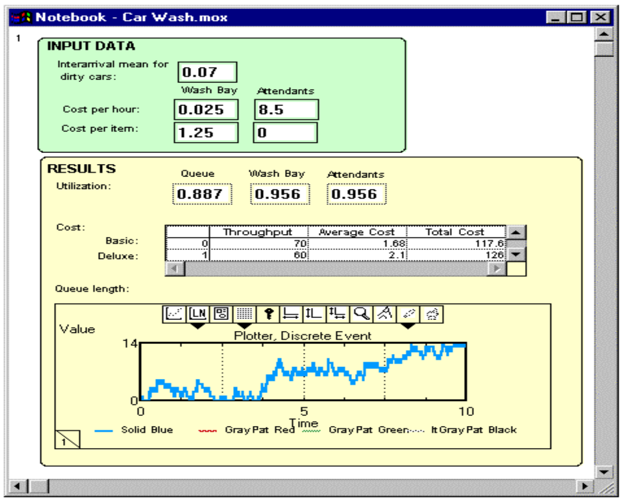


Figure 13: Car Wash Model with Hierarchical Blocks

4.3 Dialog Cloning and the Notebook

As noted in the previous section, input and output parameters associated with the model can be found in the dialogs of the appropriate blocks. While this provides an intuitive association between system metrics and the constructs used to model them, it can make searching for specific data cumbersome. This is especially true when working with large models containing many layers of hierarchy. An effective way of dealing with this is to use the Extend notebook and cloning feature. With the notebook, a single custom interface can be created that consolidates critical parameters, results, and model control to a central location.

The notebook is a separate window associated with each model. Initially, the notebook is a blank worksheet to which text, pictures, and clones can be added. Clones are direct links to dialog parameters and are created by selecting the Cloning Tool from the tool bar and using it to drag a dialog parameter from a block dialog to the notebook or model worksheet. Once a clone is created, any changes to the clone are immediately reflected in the block and vice-versa. Therefore, it is no longer necessary to access the block's dialog to change an input parameter or view updated results. Creative use of the notebook can result in a simple yet effective interface for a large, complex model. As an illustration of how the notebook can be used to consolidate important parameters into one location, Figure 14 shows the notebook for the car wash model.


Notebook - Car Wash.mox

INPUT DATA

Interarrival mean for dirty cars:	0.07	
	Wash Bay	Attendants
Cost per hour:	0.025	8.5
Cost per item:	1.25	0

RESULTS

	Queue	Wash Bay	Attendants
Utilization:	0.887	0.956	0.956

Cost:

		Throughput	Average Cost	Total Cost
Basic:	0	70	1.68	117.6
Deluxe:	1	60	2.1	126

Queue length:

Figure 14: Notebook for Car Wash Model

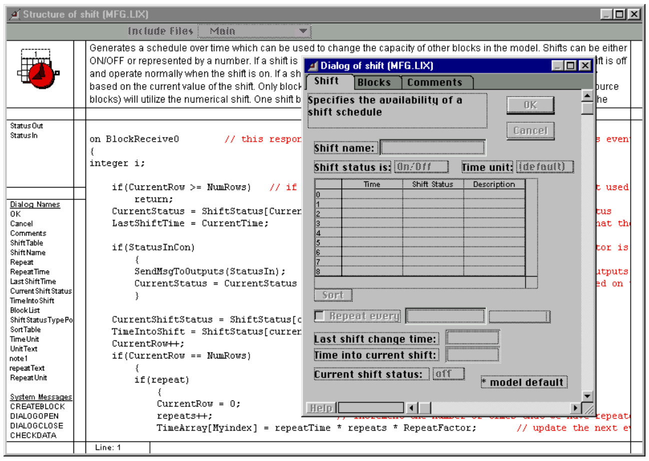
4.4 Block Development

The block development environment is one of Extend's most powerful features. While the majority of Extend users find the pre-built constructs sufficient for their needs, the block development environment provides a way for users to expand the modeling capabilities to perform unusual or highly specialized tasks. Extend's environment is easily accessible to even novice programmers. It typically takes only minutes for a programmer to learn the basics of building modeling components in Extend.

Extend's open source architecture allows access to the structure of any block that is shipped with Extend. By opening the structure, the icon, dialog, help text, and programming code of the block can be edited. The interface and functionality of any block can be modified or a new block created from scratch.

ModL is the powerful and flexible language used to define the behavior of each block. This language provides high-level functions and features while having a familiar look and feel for users with experience programming in C. In addition, external XCMDs and DLLs can be called from within ModL, giving the option of programming in any language which supports this feature (such as C or Pascal).

The ModL development environment with its interface for editing the dialog, help, icons, connectors, and code, is illustrated in Figure 15. Other tools include block performance profiling, included program files, and an interactive debugger.


Structure of shift (MFG.LIX)

Include Files: Main

Generates a schedule over time which can be used to change the capacity of other blocks in the model. Shifts can be either ON/OFF or represented by a number. If a shift is … ift is off and operate normally when the shift is on. If a sh … based on the current value of the shift. Only block … ource blocks) will utilize the numerical shift. One shift b … he

Status Out
Status In

Dialog Names
OK
Cancel
Comments
ShiftTable
ShiftName
Repeat
RepeatTime
LastShiftTime
CurrentShiftStatus
TimeIntoShift
BlockList
ShiftStatusTypePo
SortTable
TimeUnit
UnitText
note1
repeatText
RepeatUnit

System Messages
CREATEBLOCK
DIALOGOPEN
DIALOGCLOSE
CHECKDATA

```
on BlockReceive0        // this respor
{
integer i;

    if(CurrentRow >= NumRows)    // if
        return;
    CurrentStatus = ShiftStatus[Curren
    LastShiftTime = CurrentTime;

    if(StatusInCon)
        {
        SendMsgToOutputs(StatusIn);
        CurrentStatus = CurrentStatus
        }

    CurrentShiftStatus = ShiftStatus[c
    TimeIntoShift = ShiftStatus[currer
    CurrentRow++;
    if(CurrentRow == NumRows)
        {
        if(repeat)
            {
            CurrentRow = 0;
            repeats++;
            TimeArray[Myindex] = repeatTime * repeats * RepeatFactor;       // update the next e
```

Line: 1

Dialog of shift (MFG.LIX)

Shift | Blocks | Comments

Specifies the availability of a shift schedule

OK
Cancel

Shift name:

Shift status is: On/Off   Time unit: (default)

	Time	Shift Status	Description
0			
1			
2			
3			
4			
5			
6			
7			
8			

Sort

Repeat every

Last shift change time:

Time into current shift:

Current shift status: off

* model default

Help

Figure 15: ModL Block Development Environment

This level of extensibility has prompted many users to develop libraries of custom blocks for specific industries. Users and third-party developers have created libraries for modeling many systems including neural networks, control systems, high-speed systems, chemical processes, silicon wafer fabrication, pulp and paper mills, and radio and microwave communication systems. Some blocks coded by customers can be found on Imagine That's web site (<`http://www.imaginethatinc.com`>).

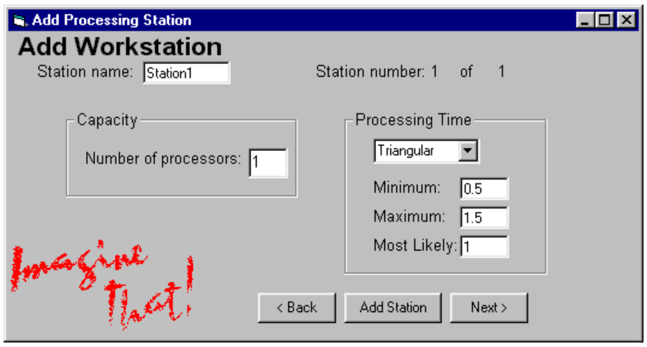
4.5 Scripting

Since Extend was created from the ground up as a graphical simulation tool, much of the process of defining a model was originally dependent on user interaction. For example, the user places blocks on the model worksheet, connects blocks together by drawing a connection between them, defines the block's behavior by double-clicking the block to open its dialog and filling the appropriate parameters, etc.

Scripting is a feature that allows models to be created and/or modified through a suite of ModL functions. With this functionality, users can create objects that automatically build and modify models. With scripting, users can develop their own model building "wizards" or self-modifying models. Without having to rely on general-purpose "wizards" provided by the software vendor, users can develop "wizards" specific to their needs and can have complete control over the level of detail and accuracy resulting from automated model building.

Coupled with Extend's ability to communicate with other applications using interprocess communication (IPC), scripting provides an easy way to allow other applications to control every aspect of Extend, including building the model, importing/exporting data, and running the simulation.

Figure 16 shows a Visual Basic application or "wizard" which builds an Extend model based on the information entered in a series of forms.


Add Processing Station

Add Workstation

Station name: Station1

Station number: 1 of 1

Capacity

Number of processors: 1

Processing Time

Triangular

Minimum: 0.5

Maximum: 1.5

Most Likely: 1

Figure 16: Visual Basic "Wizard" Building a Model

5 DISCRETE EVENT ARCHITECTURE

Extend utilizes a message-based architecture which allows for more natural model building than is possible in other simulation tools. Messages are used to pass information to connected blocks about the state and actions of the block sending the messages. For example, as soon as a queue receives an item it will send a "wants" message to the downstream blocks to see if any of them can accept an item. The messaging system is applied to the item as well as the value connectors. Because of this, complex models and logic can be built without resorting to "dummy" resources, "logical" workstations, or programming that is reusable.

5.1 Modeling Enhancements

A more advanced architecture makes modeling easier. In using a modern, message-based system Extend allows the modeler to focus on the modeling task rather than the simulation tool.

- Complex model segments can be built from simple, elemental blocks. These segments can then be saved in a library for use in other models. This type of model construction eliminates the need for "kitchen sink" modeling components in which every possible permutation must be programmed by the developer (making the interface unnecessarily complex) or requiring programming to enhance the capabilities of the modeling component.
- Easier rescheduling of events. Because blocks, not items (entities), are entered into the event calendar, changing an event time is a simple assignment. In other simulation tools, the event calendar must be searched for a specific item before the change can be made.
- Events do not have to be item based. Blocks can post themselves on the event calendar even if they do not handle items. This reduces the overhead in the model because items do not have to be generated or processed when an event occurs.
- Blocking through decisions. Extend automatically determines which path an item takes before it arrives at the decision point. The alternative to this would be adding "dummy" resources to prevent the item from moving forward if space was not available.
- Queues can be separated from activities. Any number of blocks that do not hold items (passing blocks) can be between a queue and the next activity.
- Conditions do not need to be "time checked". Messages are sent to connected blocks whenever a condition changes and the condition is evaluated immediately.
- Model logic is represented graphically and is visible as part of the model structure.

6 WHAT MAKES EXTEND UNIQUE

Extend provides features and capabilities not found in other simulation software. This allows the modeler to concentrate on the modeling process and quickly produce a model that is easy to manipulate and communicate to others. These features include:

- An integrated development environment for building modeling components which fit naturally into the user interface.
- Graphical logic making the model easier to understand and communicate.
- An unparalleled level of interactivity. Model parameters can be changed and results viewed during simulation execution. This is done through the graphical user interface; there is no need to enter a debugging mode or enter cryptic debugging commands.
- Superior hierarchy. Extend's hierarchy allows for animation and reuse, and can be any number of levels deep. This gives modelers an excellent tool for organizing large models and reusing model segments in other models.
- An innovative discrete event architecture which makes model building more intuitive.

7 APPLICATIONS

Since Extend is a general purpose simulation program, it has been applied in a wide range of areas. The sample of application examples included here are supply chain, high speed manufacturing, and chemical processing.

7.1 Supply Chain

The US Marine Corps is undergoing a revolution in the way they conduct combat operations. There are many ideas regarding how the tactical supply chain must change to provide the necessary logistical support.

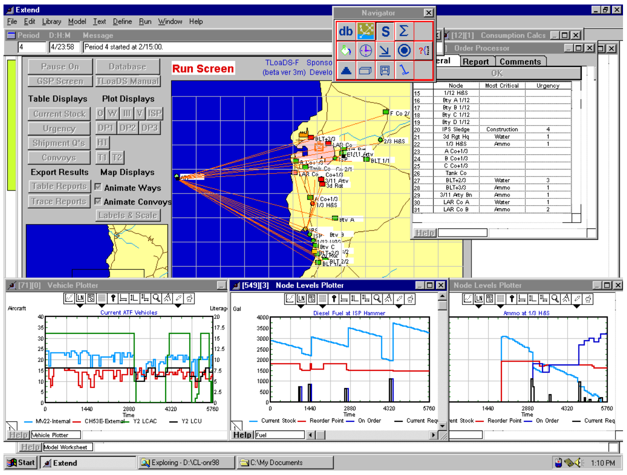


Figure 17: Supply Chain Model

The TLoaDS model (Figure 17) has been developed to explore the ability of existing, evolutionary, or revolutionary methods and equipment for this challenging mission. This application has been described as "warehouses that move", referring to the changing location of supply ships and depots (Hamber 1999).

7.2 High Speed Processing

This is a model of a packaging line that takes bulk material (cookies) in pounds, packages the material in bags, packages bags into cartons, and then cartons into cases. The model is used to understand the dynamics and capacity of the overall system. This model shows the effect of changing the speed of a piece of equipment, the failure and repair rates for that piece of equipment, the size of in-line storage, or even the mixing of different products.

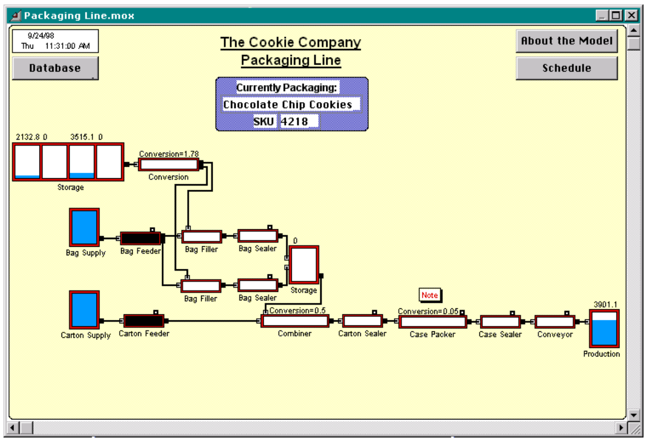


Figure 18: High Speed Processing

This example was built using the Simulation Dynamics' "SDI Industry" product which utilizes the Extend simulation engine.

7.3 Pulp and Paper Processing

This model represents an integrated pulp and paper facility in New Zealand's central North Island.

The model is a steady state representation of an integrated pulp mill, recycle facility, and sack kraft machine. It allows the engineers to perform "what ifs" to determine the optimum mix of products and grades for specific economic conditions. It is also capable of material and energy balances. Without a tool such as a model, these types of highly flexible operations, with hundreds of permutations and combinations, are difficult to optimize.

This mill now boasts bleached softwood kraft production costs in the bottom quartile worldwide.

This application was built in Simons Technologies, Inc. "IDEAS" simulation software which is based on the Extend simulation engine.

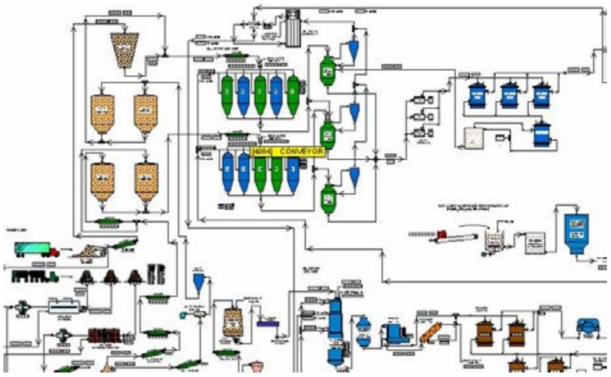


Figure 19: Pulp and Paper Processing

8 SUMMARY

As demonstrated above, Extend's design provides a superior simulation environment. By incorporating an intuitive interface, an extensive authoring and development environment, and a more advanced simulation technology, Extend has succeeded in defining its position as the leader in simulation software.

REFERENCES

Hamber, Robert. 1999. CloaDS & TloaDS. *1999 Simulation Solutions Conference.* Institute of Industrial Engineers, Norcross, GA

Imagine That, Inc. 1998. *Extend Software Manual*. San Jose, CA.

Pidd, M and Castro, R. Bayer 1998. Hierarchical Modeling in Discrete Simulation. In *Proceedings of the 1998 Winter Simulation Conference Proceedings,* ed. D. J Medeiros, E. F. Johnson, J. S. Carson, M. S. Manivannan, 383-389. IEEE, Piscataway, NJ

Krahl, Dave. 1999. Modeling with Extend. In *Proceedings of the1999 Winter Simulation Conference Proceedings,* ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans*,* 188-195. IEEE, Piscataway, NJ.

Wolverine Software Corporation. 1995. *Using Proof Animation.* Annandale, VA

AUTHOR BIOGRAPHY

DAVID KRAHL, Lead Engineer with Imagine That, Inc., is responsible for block development and technical support. He received a MS in Project and Systems Management in 1996 from Golden Gate University and a BS in Industrial Engineering from the Rochester Institute of Technology in 1986. Mr. Krahl has worked extensively with a range of simulation programs and is actively involved in the simulation community. His email and web addresses are `<davek@imaginethatinc.com>` and `<www. imaginethatinc.com>`.