A Guide to the iPlanet UDS Workshops

iPlanet™ Unified Development Server

Version 5.0

August 2001

Contents

List of Figures

List of Procedures

List of Code Examples

Preface

A Guide to the iPlanet UDS Workshops provides complete information on the iPlanet UDS interactive development environment, the iPlanet UDS Workshops. The manual also provides background information on object-oriented programming and basic iPlanet UDS concepts.

This preface contains the following sections:

Product Name Change

Forte 4GL has been renamed the iPlanet Unified Development Server. You will see full references to this name, as well as the abbreviations iPlanet UDS and UDS.

Audience for This Guide

A Guide to the iPlanet UDS Workshops is intended for application developers. We assume that you:

- have programming experience
- are familiar with your particular window system
- are familiar with SQL and your particular database management system

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter	**Description**
Chapter 1, "Overview"	Describes the application development process and introduces the workshops you use for each phase of development.
Chapter 2, "Using the iPlanet UDS Workshops"	Provides general information about how to use the iPlanet UDS Workshops including how to use the Control Panel to set common environment variables and set workshop preferences.
Chapter 3, "Using the Repository Workshop"	Provides conceptual information about iPlanet UDS repositories and workspaces, and describes how to use the Repository Workshop.
Chapter 4, "Using the Project Workshop"	Provides conceptual information about TOOL projects and their components, and describes how to use the Project Workshop.
Chapter 5, "Using the Class Workshop"	Provides conceptual information about classes and their elements, and describes how to use the Class Workshop.
Chapter 6, "Using the Interface Workshop"	Provides conceptual information about interfaces and their elements, and describes how to use the Interface Workshop.

Chapter	Description
Chapter 7, "Using the Window Workshop"	Provides conceptual information about windows and their components, and describes how to use the Window Workshop
Chapter 8, "Working with Widgets"	Provides general information about working with widgets in the Window Workshop.
Chapter 9, "Using the Menu Workshop"	Provides background information about menu bars and popup menus, and describes how to use the Menu Workshop.
Chapter 10, "Using the Method Workshop"	Provides background information about methods and describes how to use the Method Workshop.
Chapter 11, "Using the Event Handler Workshop"	Provides background information about named event handlers and describes how to use the Event Handler Workshop.
Chapter 12, "Using the Cursor Workshop"	Provides background information about cursors and describes how to use the Cursor Workshop.
Chapter 13, "Using the Debugger"	Describes how to use the Debugger to debug your application.
Chapter 14, "Using the Partition Workshop"	Provides conceptual information about distributed applications, libraries, environments, and configurations, and describes how to use the Partition Workshop.
Appendix A, "iPlanet UDS Example Applications"	Provides instructions on how to install the examples, a brief overview of the applications to help you locate relevant examples, and a section describing each example in detail.
Appendix B, "Memory and Logger Flags"	Contains a detailed description of how to use the memory (**-fm**) and logger (**-fl**) flags.

Text Conventions

This section provides information about the conventions used in this document.

Format	Description
italics	Italicized text is used to designate a document title, for emphasis, or for a word or phrase being introduced.
`monospace`	Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs.
ALL CAPS	Text in all capitals represents environment variables (FORTE_ROOT) or acronyms (UDS, JSP, iMQ). Uppercase text can also represent a constant. Type uppercase text exactly as shown.
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-S means press the Esc key, release it, then press the S key.

Other Documentation Resources

In addition to this guide, there are additional documentation resources, which are listed in the following sections. The documentation for all iPlanet UDS products (including Express, WebEnterprise, and WebEnterprise Designer) can be found on the iPlanet UDS Documentation CD. Be sure to read "Viewing and Searching PDF Files" on page 50 to learn how to view and search the documentation on the iPlanet UDS Documentation CD.

iPlanet UDS documentation can also be found online at http://docs.iplanet.com/docs/manuals/uds.html.

The titles of the iPlanet UDS documentation are listed in the following sections.

iPlanet UDS Documentation

- *A Guide to the iPlanet UDS Workshops*
- *Accessing Databases*
- *Building International Applications*
- *Escript and System Agent Reference Guide*
- *Fscript Reference Guide*
- *Getting Started With iPlanet UDS*
- *Integrating with External Systems*
- *iPlanet UDS Java Interoperability Guide*
- *iPlanet UDS Programming Guide*
- *iPlanet UDS System Installation Guide*
- *iPlanet UDS System Management Guide*
- *Programming with System Agents*
- *TOOL Reference Guide*
- *Using iPlanet UDS for OS/390*

Express Documentation

- *A Guide to Express*
- *Customizing Express Applications*
- *Express Installation Guide*

WebEnterprise and WebEnterprise Designer Documentation

- *A Guide to WebEnterprise*
- *Customizing WebEnterprise Designer Applications*
- *Getting Started with WebEnterprise Designer*
- *WebEnterprise Installation Guide*

Online Help

When you are using an iPlanet UDS development application, press the F1 key or use the Help menu to display online help. The help files are also available at the following location in your iPlanet UDS distribution: `FORTE_ROOT/userapp/forte/cln/*.hlp`.

When you are using a script utility, such as Fscript or Escript, type help from the script shell for a description of all commands, or help *<command>* for help on a specific command.

iPlanet UDS Example Programs

A set of example programs is shipped with the iPlanet UDS product. The examples are located in subdirectories under `$FORTE_ROOT/install/examples`. The files containing the examples have a `.pex` suffix. You can search for TOOL commands or anything of special interest with operating system commands. The `.pex` files are text files, so it is safe to edit them, though you should only change private copies of the files.

Viewing and Searching PDF Files

You can view and search iPlanet UDS documentation PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

NOTE You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from http://www.adobe.com. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

➤ To copy the documentation to a client or server

1. Copy the `doc` directory and its contents from the CD-ROM to the client or server hard disk.

   You can specify any convenient location for the `doc` directory; the location is not dependent on the iPlanet UDS distribution.

2. Set up a directory structure that keeps the `udsdoc.pdf` and the `uds` directory in the same relative location.

   The directory structure must be preserved to use the Acrobat search feature.

NOTE To uninstall the documentation, delete the `doc` directory.

➤ To view and search the documentation

1. Open the file `udsdoc.pdf`, located in the `doc` directory.

2. Click the Search button at the bottom of the page or select Edit > Search > Query.

3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click Search.

   A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

   NOTE For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

   All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

   To return to the `udsdoc.pdf` file, click the Homepage bookmark at the top of the bookmarks list.

6. To revisit the query results, click the Results button at the bottom of the `udsdoc.pdf` home page or select Edit > Search > Results.

Chapter 1

Overview

This chapter briefly describes the application development process and introduces the workshops you use for each phase of development. This product overview is intended to provide you with the background information necessary for designing and building iPlanet UDS applications.

First, this chapter presents general information about object-oriented programming and distributed application development. This general information is followed by a quick overview of the iPlanet UDS Workshops.

To illustrate some of the concepts introduced in this chapter, we use the iPlanet UDS example program application called Auction. The Auction application is a distributed application that allows multiple users to view information about paintings and to make bids on them. As each user makes a bid, all other users are immediately notified. Thus, the application functions like a real art auction, where a room full of people are all interacting with each other.

About iPlanet UDS

iPlanet UDS provides development tools for building advanced, distributed applications. These tools enable developers to build integrated applications that are optimized for a distributed environment. iPlanet UDS supports quick prototyping, collaboration by multiple developers, and graphical user interface development.

iPlanet UDS has the following basic components:

Component	Definition
Repository	A central storage facility that supports every aspect of developer collaboration, including code sharing and versioning.
Workshops	Visual tools for structuring the application and building a graphical user interface, and for partitioning the application definition and mapping it to physical resources and machines.
Express	The iPlanet UDS application generation tool. Using Express workshops—the Business Model Workshop and the Application Model Workshop—you develop graphical models of objects in your business system and the window flow of your application. Express automatically generates an application based on these models. Subsequently, you can use the standard iPlanet UDS workshops to further refine and customize your final application.
iPlanet Integration Server	A suite of business integration tools for integrating and coordinating heterogeneous applications. iPlanet Integration Server has two subsystems: the iPlanet Integration Server Process Management System and the iPlanet Integration Server Backbone System. The iPlanet Integration Server Process Management System provides workshops for developing business processes and a process engine to manage and execute those processes. You can optionally develop process clients to directly interact with the iPlanet Integration Server process engine; standard APIs are provided for this purpose. The iPlanet Integration Server Backbone System lets you integrate any application, either using a native XML interface or an iPlanet UDS XML adapter. An iPlanet Integration Server backbone enables interapplication message brokering and data transformation using XSLT. The applications can optionally participate in a business process managed by the iPlanet Integration Server process engine.
Environment Console	Visual tool for defining the deployment environments in which your iPlanet UDS applications will run, and for monitoring and maximizing the performance of installed applications.
TOOL and TOOL Debugger	An object-oriented, fourth generation programming language for writing the program logic and a debugger designed especially for the language.
iPlanet UDS libraries	Prefabricated classes that provide the framework for building applications. iPlanet UDS classes define basic application components, such as windows, menus, and fields, as well as external services, such as database management systems and transaction management.

Modular Construction

The metaphor of designing and constructing a building is an accurate way to describe the process of designing and building an application. This metaphor is useful because it gives us a concrete and familiar way to explain abstract programming concepts. But most of all, it helps to clarify the importance of iPlanet UDS's object-oriented design. With iPlanet UDS, you can use this same simple, effective approach in constructing applications.

Modular construction in iPlanet UDS means building an *object-oriented* application. In iPlanet UDS, objects are the basic modular building blocks. When you build an application, you specify which objects are created and how they interact with each other.

Classes are the "templates" you use for creating the objects. Prefabricated classes provide standard, off-the-shelf objects that you can use in a wide range of applications. Custom classes provide objects designed especially for your application or your organization. (Those of you who are not familiar with object-oriented programming concepts should read "Object-Oriented Programming" on page 56 for a detailed explanation of these terms.)

Everything that you build using iPlanet UDS is stored in the repository. This application "warehouse" is a special storage facility that allows multiple developers to collaborate on a single application and to share and reuse code. Special browsers enable you to "shop" for the components you need to build your application.

Construction Crew

The development team for an application works together like a construction crew. Like subcontractors at a building site, multiple developers can collaborate on application development by taking separate responsibility for individual modules. Because classes are modular by nature, a single developer can take responsibility for an individual class or a set of classes.

The repository coordinates this team work by tracking who is working on which module and by ensuring that only one person can modify a particular module at a time.

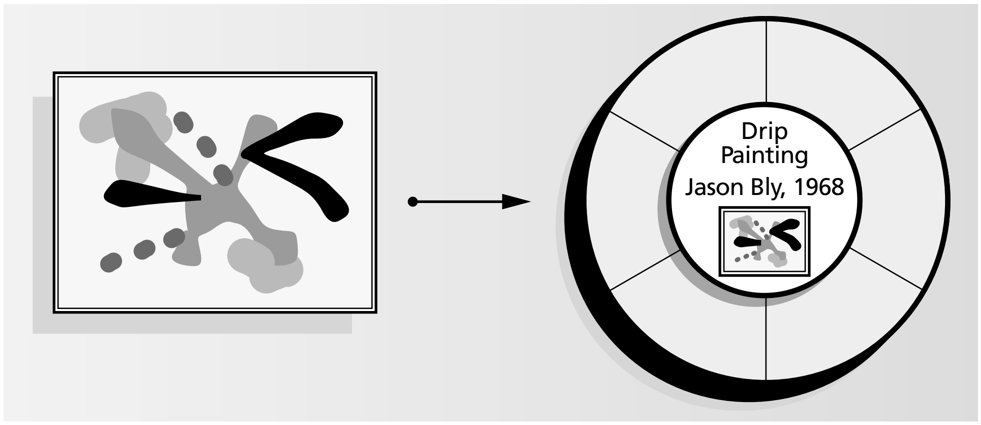
Connecting to Services

In iPlanet UDS, an application is not an independent program, but a collaborative system that interacts with external services and other applications. Connecting your application to these outside services is as natural as connecting a building to telephone, electricity, and water utilities. Standardized, built-in modules let you plug into external services the same way you add any new modules to your application.

Object-Oriented Programming

For those of you who are not familiar with object-oriented programming, an object is a self-contained module that consists of data (called attributes) and operations (called methods) that act on the data. For example, one of the most important objects in the Auction application is the Painting object. This object contains information about the painting in the form of attributes, including the title, artist, and year of completion of a specific painting, and defines an operation to update the data. Figure 1-1 is an abstract representation of a Painting object:

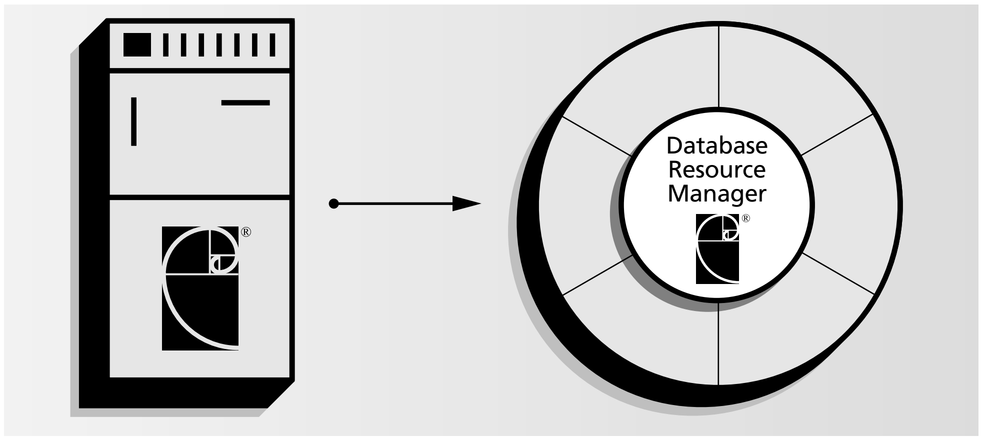
Figure 1-1 Painting Object



Much of the time, an iPlanet UDS object corresponds directly to an object or person in the “real world.” For example, in the Auction application, there are objects for paintings, artists, and bids.

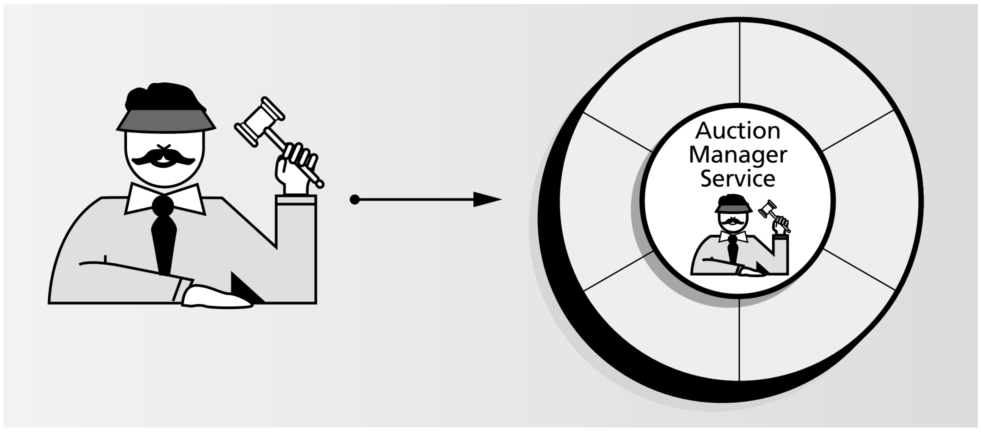
However, at other times an iPlanet UDS object represents a more abstract system component, such as an external service with which you want your application to interact. These objects include database management systems and other existing applications. For example, if you want to start a session with Sybase, you would use a DBResourceMgr object that represents the Sybase installation.

Figure 1-2 Object for Sybase Installation



Finally, an iPlanet UDS object may also represent a processing facility within your system used to control other objects. For example, the Auction application has an AuctionMgr object, which manages the current list of bids that are being offered for the paintings. The Auction application also has an ImageMgr object that manages the images of the art work that is being stored on a server.

Figure 1-3 Object for Auction Manager



Objects provide the basis for your entire application. When you construct an iPlanet UDS application, you specify which objects are included and how they interact with each other.

The structure of an object is determined by its *class*. The class defines the data and operations for all the objects of the same class. A class is similar to a user-defined data type. All objects of the same class have the same types of data, but they do not contain the same "information." For example, objects of the Artist class all have a Country attribute, but an object for Rembrandt has "Holland" for the value of the Country attribute, while an object for Degas has "France" for the value of the Country attribute.

When you build your application, you use classes to specify the kinds of objects you want the application to contain. The actual objects are created when the application is executing and they contain data provided by the run-time system. In fact, every object is really just an "instance" of a class. See "Using Classes" on page 63 for more information about classes.

Using objects provides two important advantages. First, once you have defined a particular class, you can use it any number of times within your application or store it in a library for use within other applications. Classes provide a simple mechanism for reusing and sharing your code. Second, because classes provide a modular application design, there is a natural basis for partitioning the application across a set of machines. In iPlanet UDS, individual objects can have different physical locations. When you request an operation on an object, iPlanet UDS automatically performs the operation on the machine where the object resides. For example, in the Auction application, the AuctionMgr object is located on a server. When an end user makes a bid on a painting from a client, iPlanet UDS performs the necessary operations directly on the AuctionMgr server.

Every object consists of three different components: its attributes, its methods, and its events.

- An *attribute* is simply a data item. All the data for the object is stored in its attributes. For example, the Bid object in the Auction application has `PaintingforBid`, `BidValue`, `LastBidTime`, and `LastBidder` attributes, all of which contain information about the bid being made on a particular painting.
- A *method* is a function that you can use to manipulate an object. Each method is written specifically for a class, and you can use the method only to operate on objects of that class. For example, the Bid class in the Auction application has a `CompleteBid` method, which updates the Bid object (and notifies the rest of the application) whenever a bid is completed. When only the class methods can operate on the object, it is much easier to maintain your code. Only the methods need to know how the object is structured. If you change the structure of the object, you need to update only the methods, not the rest of the program. The requirement that a method must modify an object by using one of the object's methods is called *encapsulation*.
- An *event* is a notification that something significant has happened An object can post an event to notify the rest of the application of a change in status. For example, in the Auction application, a BidCompleted event is used to notify all clients that a bid was made on a particular painting. The ability to define an event associated with an object is a special feature of iPlanet UDS. Events are described in further detail under "Events" on page 61.

How Objects Interact

In iPlanet UDS, objects interact in two different ways: through methods and through events.

Methods

One object can cause another object to "react" by invoking one of its methods. For example, in the Auction application, when the ViewWindow object (the end user's window on the client) invokes the `GetImage` method on the ImageMgr object (the image server), the ImageMgr object reacts by displaying the requested painting on the end user's workstation.

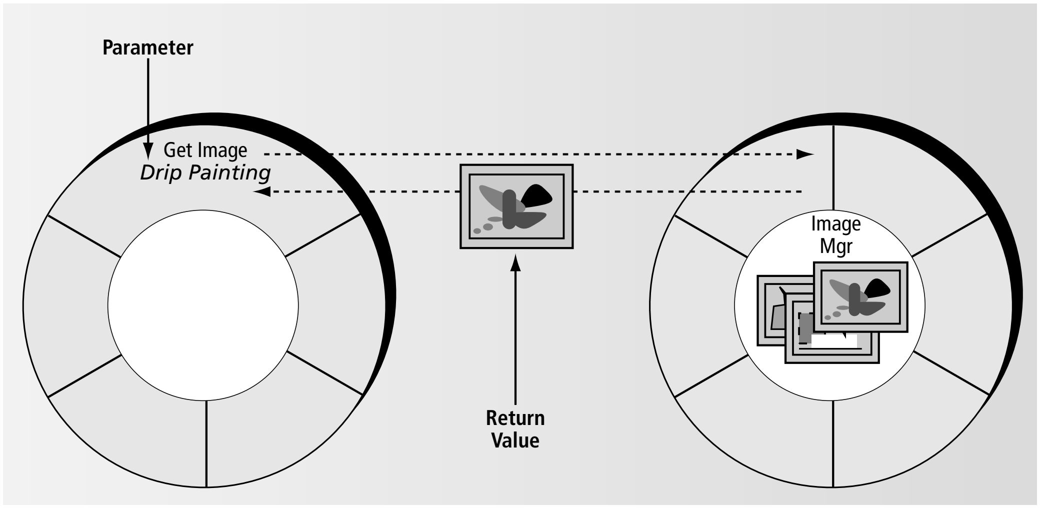
Many methods have *parameters*. Parameters allow the object invoking the method to pass information to the object that it is manipulating. The `GetImage` method for ImageMgr has one parameter that specifies the name of the painting to be displayed. The parameters for methods can be for input, output, or both input and output.

Input parameters provide information to the object on which the method is being invoked. For example, the `GetImage` method has an input parameter that specifies the name of the painting that is being requested. Output parameters allow the object invoking the method to receive information from the object it is manipulating. For example, the `GetBidForPainting` method has output parameters that provide the invoking method with the bid value, the bidder name, and so on.

A method can also have a *return value*, which allows the object being manipulated to pass information back to the object that invoked the method. A return value provides a two-way form of communication between the two objects. For example, the `GetImage` method for ImageMgr returns an ImageData object, which is the bitmap image of the requested painting.

The following figure illustrates invoking the GetImage method:

Figure 1-4 Invoking a Method



Invoking a method is a direct instruction from one object to another to perform a specific action. Invoking a method in your code is like calling a procedure. iPlanet UDS executes the method and then returns to the point where the method was invoked.

Methods provide the programming logic of your application. Every iPlanet UDS application begins execution by invoking a startup method on a startup object. The startup method in the application begins execution by operating on the startup object. From the startup method, you also invoke other methods. Each time you invoke a method, control passes to the invoked method. When each method completes, control returns to the invoking method. The flow of control is the standard processing model that you are already familiar with.

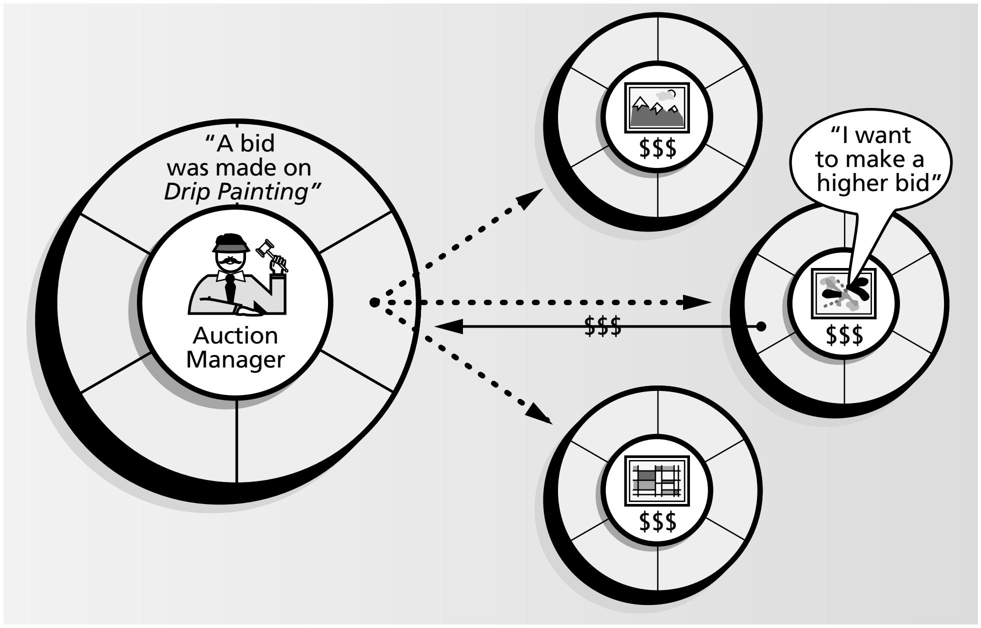
Events

One object can also cause other objects to "react" by triggering an event. In iPlanet UDS, an event is simply a notification that something significant has happened. For example, the Auction application uses the BidCompleted event to notify all clients displaying a particular painting that a bid was made on that painting. When one object triggers an event, other parts of the application that are watching for that event can react to it. In the Auction application, the clients that are currently displaying the painting respond to the event by notifying the end user that the bid was made.

iPlanet UDS is an event-based system. End user actions, such as button clicks, menu item selection, and data entry, trigger events to which the objects can respond. However, a method that is operating on an object can also trigger an event.

The following figure illustrates the relationship between objects and events:

Figure 1-5 Relationship between Events and Objects


"A bid was made on *Drip Painting*"
Auction Manager
$$$
"I want to make a higher bid"
$$$
$$$
$$$

Triggering an event is like "broadcasting" a signal to the rest of the system that a certain condition or action has occurred. Any number of objects may be "tuned in," waiting for that particular signal. If so, when the objects receive the event, they will respond to it. If no object is "tuned in," there will be no response. The object that triggers the event has no way of knowing which object will respond to it or even if there will be a response. Therefore, when you post an event, you cannot rely on that event causing a result. On the other hand, invoking a method on an object ensures that the method will take effect.

Like methods, events can have *parameters* that allow the object that is triggering the event to provide information to any objects that may respond to the event. For example, the BidCompleted event has parameters that specify the value of the bid, the time the bid was made, and the name of the bidder.

Using events is a one-way form of interaction. One object produces an event and another responds to it. No communication takes place after the event is triggered.

iPlanet UDS provides you with the ability to write *event handlers* as part of a class. An event handler is a named block of code that provides programming to be executed in response to one or more events. The event handler provides reusable, modular event handling code.

Using Classes

You use classes to specify the kinds of objects you want your application to contain. To develop an iPlanet UDS application, you must create a project. A *project* is a named collection of classes. The classes in the project define the objects that make up the application. For example, each window in the user interface is defined by a class. The classes also provide the methods that control the flow of the application.

Every class in the project is a template that defines the attributes, methods, events, and event handlers for all the objects of that class. For example, the `Artist` class in the Auction application defines the attributes and methods for artists. All Artist objects have `Name`, `Country`, `Born`, `Died`, and `School` attributes. The following example code illustrates using the `Artist` class to create a variable and assign to that variable a new Artist object:

```
a: Artist;
a = new(name='Pollock', born = 1912, died = 1956);
```

The first statement declares a variable whose type is the `Artist` class. The second statement creates a new object of the `Artist` class, with the specified values for its attributes.

There are two types of classes in iPlanet UDS: prefabricated classes and custom classes.

Prefabricated Classes

Prefabricated classes include classes that are provided in the iPlanet UDS libraries, classes stored in your organization's libraries, and classes in supplier projects.

iPlanet UDS system libraries The iPlanet UDS system libraries provide the basic classes you need to create an application. The classes in the iPlanet UDS system libraries define such things as windows, menus, fields, arrays, files, and text, as well as external services, such as database management systems.

Class libraries Your organization may also have one or more shared libraries of prefabricated classes that define objects appropriate for your particular environment. Using classes from libraries not only saves you the work of having to write basic methods over and over again, but also creates consistency throughout your organization.

Supplier projects and libraries Different developers working in the same repository can also share the same classes by using supplier projects and libraries (called *supplier plans*). Supplier plans are projects or libraries that contain classes that you want to include in the current project. To use supplier classes in your project, you designate the projects or libraries they belong to as "suppliers" for your project. Any classes in your supplier plans are available to you for use in writing your methods.

Custom Classes

Custom classes are classes that you create to meet the special needs of your particular application. When you create a class, you specify its attributes, define its events, write the methods that provide the basic operations, and write event handlers, if desired. Besides the classes mentioned earlier, another important custom class in the Auction application is the ViewWindow class, which defines the window displayed on the end user's workstation and determines its behavior.

The most important part of building your project is creating its custom classes. The methods for your custom classes determine the logical structure of your application. You use these methods to open and close windows, to respond to end user actions, to interact with your databases, and so on. In the Auction application, the `Display` method for the `ViewWindow` class provides the event-handling code that determines how the end user interacts with the Auction Manager.

Inheritance

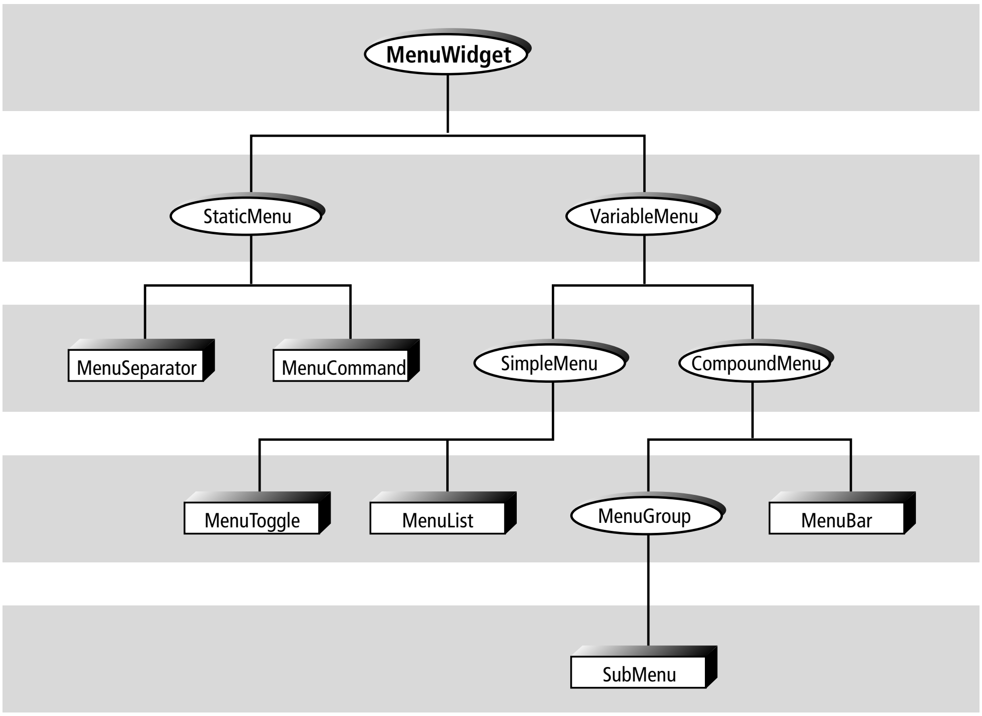
When you create a class, you must declare it as a subclass of an existing class. This means that the new class will "inherit" all the attributes, methods, events, and event handlers defined for the existing class. You can then add new attributes, methods, and events to your new class, or replace any of its inherited methods.

This ability to define one class as an extension of another is called *inheritance*. By starting with one class, defining a second class as a subclass of the first, defining a third class as a subclass of the second, and so on, you can create an inheritance "hierarchy."

The iPlanet UDS Display library makes good use of inheritance (see Display Library online Help for complete information on the Display library classes). For example, all the widgets (or controls) that you can place on a window are subclasses of the FieldWidget class. The FieldWidget class defines all the sizing and coloring attributes needed by all the specific widgets, such as push buttons and data fields. The individual widgets then inherit these basic attributes from FieldWidget, but add their own specialized attributes.

Figure 1-6 illustrates an inheritance hierarchy from the Display library:

Figure 1-6 Inheritance Hierarchy



A class that is above another in the hierarchy is called the *superclass*. A class that is below another in the hierarchy is called the *subclass*. A subclass always inherits the attributes, methods, and events defined for all its superclasses.

Inheritance is a very useful mechanism for sharing common attributes, events, methods, and event handlers among several different classes. Using inheritance saves you the work of writing multiple versions of the same definitions, and provides a consistent way for you to interact with different objects in your system.

For example, the `FieldWidget` class defines a generic `MoveAbove` method, which is inherited by all its widget subclasses. Then, each time you invoke the `MoveAbove` method for any of the subclasses, you will always use the same method name, the same parameters, and so on. In addition, you can easily modify an inherited method so that it performs special processing for the subclass.

Abstract Classes

When creating an inheritance hierarchy, it is very useful to centralize the definitions you want to share by creating abstract classes. An abstract class is not used for creating objects that you want to manipulate in your system. Instead, an abstract class simply defines the common attributes, methods, events, and event handlers for a set of subclasses. For example, the `FieldWidget` class in the Display library is an abstract class that defines the attributes, methods, and events needed for all the individual widgets. Because they provide definitions for a set of subclasses, abstract classes are near the top of the inheritance hierarchy. Concrete classes, the classes you actually use for creating objects, are generally the end points of the hierarchy.

Planning Your Class Hierarchy

The methods, attributes, events, and event handlers for each class come from two sources:

- the class inherits them from its superclasses
- you define them especially for the class

As described above, every custom class is a subclass of another class (the iPlanet UDS Object class is always at the top of the hierarchy). Therefore, any new class that you create automatically inherits the attributes, events, methods, and event handlers defined for its superclasses. The first thing you do when you create a custom class is to identify its immediate superclass. The superclass determines its place in the inheritance hierarchy for the project.

When you plan the new classes for your projects, you need to consider how they will fit into the inheritance hierarchy. When planning your class hierarchy you should know that you can override methods as well as control their visibility.

Overriding Methods

Overriding means creating a new method to replace an inherited method. If you use the name of an inherited method name and specify the same parameters used in that method, iPlanet UDS “overrides” the inherited method for the current class. This allows you to create a different “version” of the method that will be invoked with the same method name. When you invoke a method on the object, iPlanet UDS uses the method you defined specifically for the class rather than the inherited method.

Using Private Methods

Another thing to consider is your ability to control the visibility of methods, attributes, events, and event handlers for the subclasses, You can define any method, attribute, or event as “private.” A private method, attribute, event, or event handler can be accessed only by the class that defines it, not by any other classes using the object, including any of its subclasses. This enables you to essentially prevent an attribute, event, method, or event handler from being inherited. By default, attributes, events, methods, and event handlers are public, which means that any other classes in the project can access them.

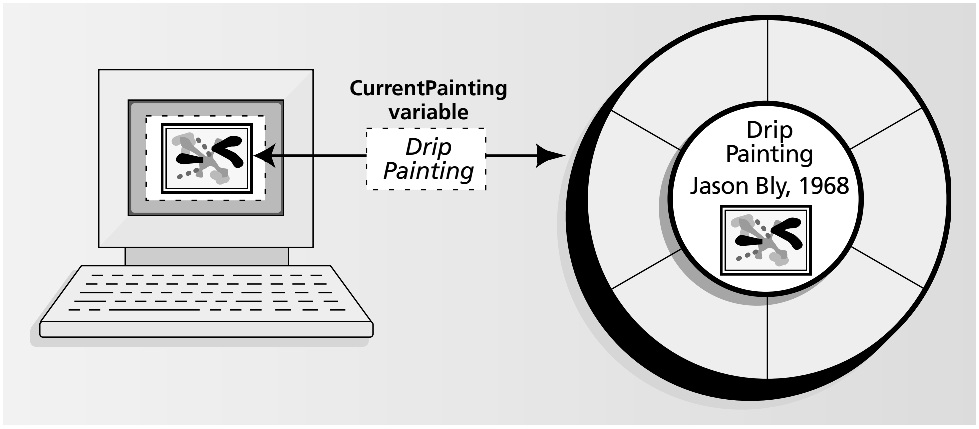
Working with Objects

In iPlanet UDS you never work with objects directly. Every object is associated with at least one data item, either a variable, an attribute, or a parameter. This data item serves as a reference to the object. The data item itself does not have a value, but is simply a name that you can use to reference an object of a particular class.

When you declare a variable or define an attribute or parameter, you specify the class of the object that it points to. After that, you create the actual object. You cannot construct an object without assigning it to a variable, attribute, or parameter.

The following figure illustrates the relationship between a variable and an object:

Figure 1-7 Relationship between Variable and Object



When you want to work with the object, for example, to invoke a method on it or set one of its attributes, you must reference it by giving the name of the variable, attribute, or parameter that points to it.

See the *TOOL Reference Guide* for details about creating and referencing objects.

About Distributed Applications

A distributed application provides access to distributed machines and services through a single, integrated system. The typical client-server application runs on only two machines, a desktop computer and a database server. However, an iPlanet UDS application can run on any number of different machines. A single application can access any number of distributed services, including:

- any number of databases
- 3GL services, such as an API to the NY stock exchange, a statistical analysis package, or a 3GL application developed by your own organization
- services written with iPlanet UDS, such as an image server or coordination facility

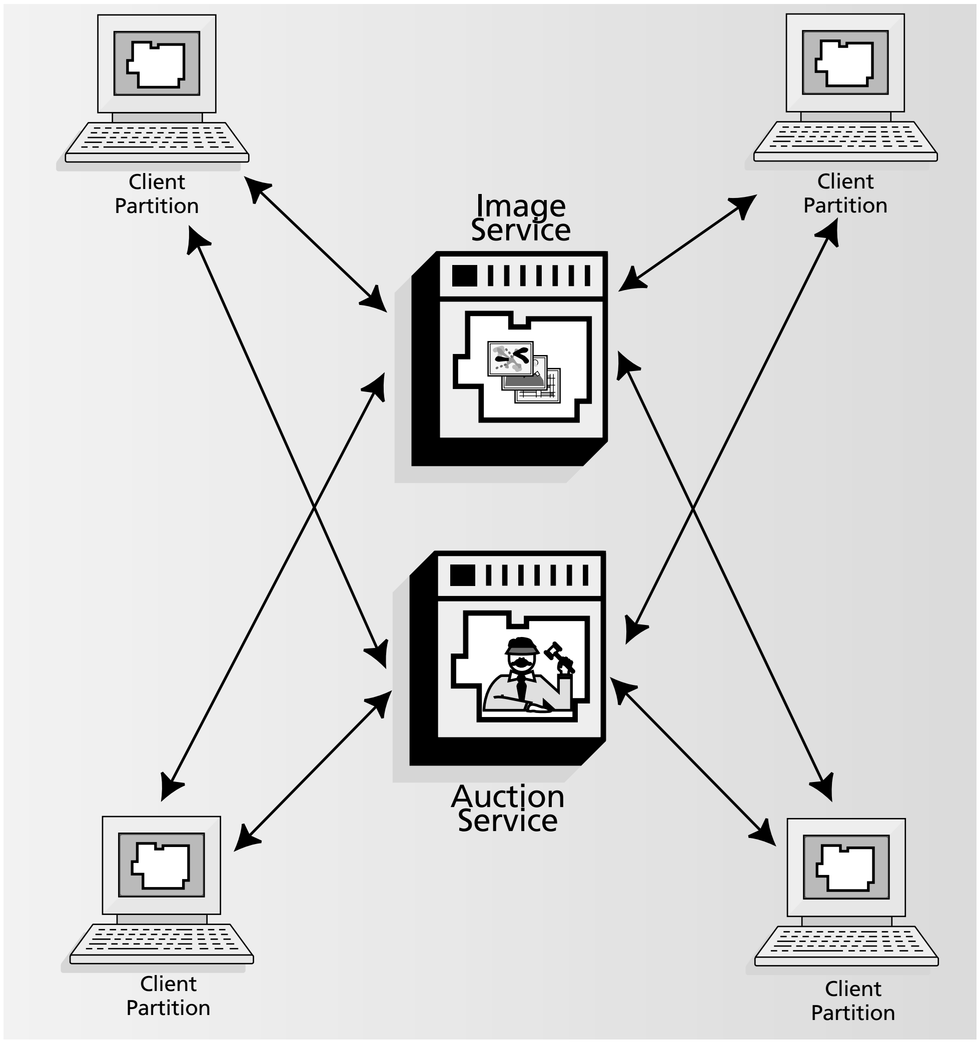
An iPlanet UDS application can also provide fault tolerance and parallel processing by automatically accessing backup servers and load balancing across servers and machines.

About Partitions

To distribute an application, iPlanet UDS divides it into several logical sections, called partitions. Each partition is an independent component, which can run on its own machine. For example, almost every end user application has a client partition on the desktop that provides the graphical user interface. Other partitions could include a DBMS server that runs on a server machine, an image server that runs on a specialized machine, a 3GL service, and so on. iPlanet UDS automatically coordinates all communication between the partitions.

The following figure illustrates the use of partitions in a distributed application.

Figure 1-8 A Distributed Application



A partition is made up of one or more service objects. To create a distributed application, you must define service objects as part of your project.

About Service Objects

As described earlier, an iPlanet UDS application is made up of objects. All of the distributed services with which your application interacts are objects. For example, a database that you wish to access from your application is an object. Likewise, an existing 3GL application that you wish to use is an object. To interact with these objects, you invoke methods on them, just as you would on any other object in your application.

In a distributed application, every object has a single, fixed location. When methods are invoked on an object, they are executed on the machine on which the object is located. Normally, the object is located on the machine on which it was created.

Although your application may consist of thousands of objects, only a few of these need to be in particular locations. Objects that need to be in a particular location include:

- an object that represents an existing external resource, such as a database management system or a 3GL service, that is already present on a particular machine
- an object that defines a shared business service, such as the iPlanet UDS image server and auction manager in the Auction application
- an object that defines a service that you wish to replicate to provide failover or load balancing

When you are ready to divide your application into partitions, these central services are the only objects that you need to work with.

When you create your project, you must specify which objects in the application need to be located on specific partitions. To do this, you simply create and name *service objects*. For example, in the Auction application, we have defined a service object named AuctionService with a type of `AuctionMgr` class. We need this service object because the auction manager service is a shared service that must run on a central server. You do not need to worry about the locations of the rest of the objects in your application. If a service object creates other objects, these will be located on the same partition as their creator. And iPlanet UDS provides you with completely transparent access to all the objects. For example, the AuctionMgr service object (called AuctionService) creates and maintains the Bid objects that contain all the information about the end user's bids. These Bid objects always stay with the service object that creates them, which provides the most efficient communication.

A service object is simply a named object that you can reference from any method in your application. You can think of a service object as being like a global variable, except that you specify its value at compile time. When you create a service object, you give it a name, a class, and values for its attributes.

After creating a service object, you can work with it like any other object. If you want to invoke a method on a service object that is located on a remote partition, you can invoke the method as you would on any other object. The method executes on the machine where the object is located, and iPlanet UDS returns the return value and output parameters to the partition where you invoked the method. Although you cannot reassign the value of the service object itself, you can change the values of its individual attributes just as you would set the value of any object's attributes.

DBMS Resource Manager Service Objects

A special kind of service object is a DBMS resource manager. A DBMS resource manager service object provides you with access to your database management system. By defining a DBMS resource manager service object, you can interact with your database like any other object. You simply invoke methods on the service object to start a database session, retrieve data, update data, and so on.

You create the service object as part of your project. Later, you can assign the service objects to specific partitions. Assigning service objects to partitions is described under "Partitioning an Application" on page 75.

Load Balancing and Failover

One of the most important advantages of using service objects is the ability to make multiple copies of them (called *replicating*) for load balancing and/or failover. The following two sections briefly describe the iPlanet UDS load balancing and failover features.

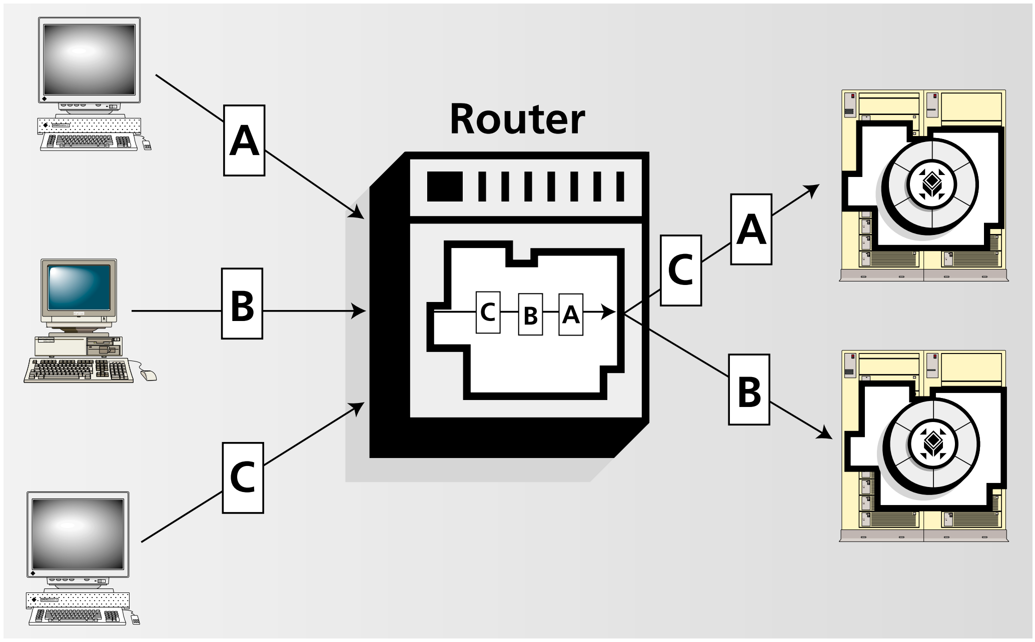
Using Load Balancing For Performance

You can replicate a service object any number of times for load balancing. Load balancing means using multiple copies of shared service to provide simultaneous access for several clients at once. iPlanet UDS automatically coordinates the connections to all the copies.

To provide load balancing, you replicate your service object and install the replicates on different nodes in the environment or on the same node in the environment. When a service object is defined with load balancing, iPlanet UDS automatically provides a router that coordinates the parallel processing.

The following figure illustrates load balancing:

Figure 1-9 Load Balancing



Using Failover for Reliability

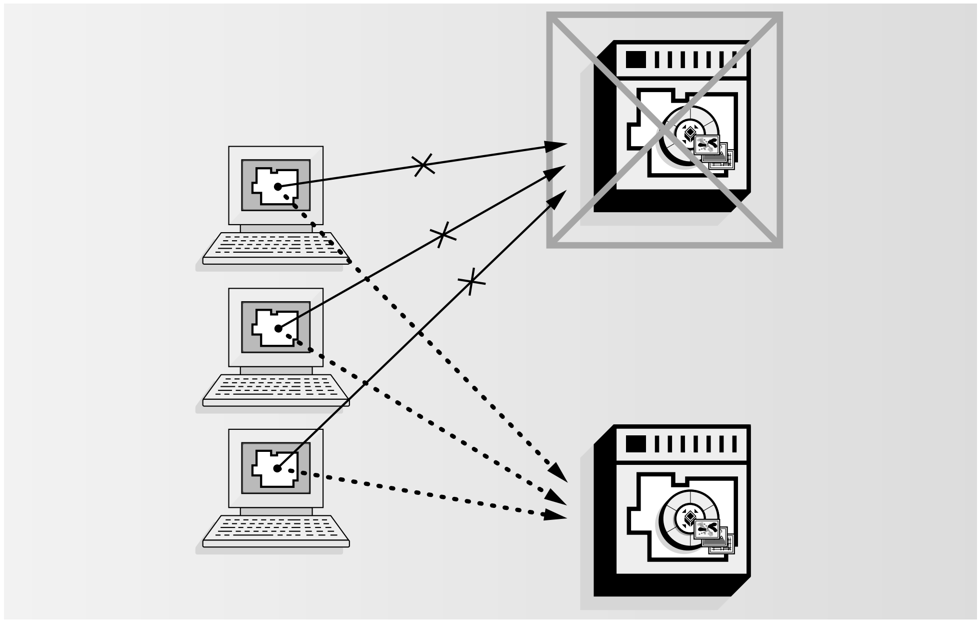
You can also replicate a service object any number of times for failover. Failover means providing backup service objects to be used if the primary service fails. Having backup service objects provides built-in fault tolerance for the application.

To provide failover, you replicate your service object and install the replicates on different nodes in the environment (for hardware failover). It is also possible to install failover replicates on the same node in the environment (for software failover). When a service object is defined with failover, iPlanet UDS automatically switches over to the secondary replicate when the primary service object fails. If the secondary replicate fails, iPlanet UDS uses the next replicate if there is one. You can provide any number of replicates that you wish.

Note that this is not the same thing as load balancing. When a service object is replicated for failover, iPlanet UDS maintains a connection to only one of the replicates of the service object. The others are merely in place in case of failure.

The following figure illustrates failover:

Figure 1-10 Failover



About Environments

To create a distributed application, your system manager must describe the environments in which the application will be deployed.

An environment is a named description of the hardware and software at a particular site, such as the hardware and software installed in your London office or your personnel department. In iPlanet UDS, you can create any number of environments.

A single iPlanet UDS application can run in any number of different environments. Likewise, a single environment can run any number of iPlanet UDS applications. However, before you can install any application in an environment, you must partition the application specifically for the environment. This means assigning each of the application's service objects to specific partitions and specifying which partitions run on which nodes.

Partitioning an Application

An application is a project that has been distributed within a specific environment. Partitioning your application for a specific environment customizes your application for the particular hardware and software on which the application will be deployed.

iPlanet UDS automatically partitions your application for each of your environments. Most of your service objects can run only on a certain node. For example, your database is probably already installed on a particular machine. Therefore, iPlanet UDS automatically assigns the service objects to specific partitions and assigns the partitions to the nodes in the environment that have the appropriate resources and capabilities.

After iPlanet UDS partitions the application, you can examine it and make adjustments. If necessary, you can assign service objects to different partitions or assign partitions to different nodes. After you have finished partitioning your project, you have a *configuration*. From each configuration, you can generate an *application distribution*. An application distribution contains all the files necessary for installing an application into a particular deployment environment.

The Development and Deployment Process

With iPlanet UDS, there are four phases to developing and deploying a distributed application.

➤ **To develop and deploy an application**

1. Specify the various *environments* in which the application will run. An environment describes the hardware and software on which you plan to deploy your iPlanet UDS applications, such as the hardware and software installed in your London office or your personnel department. Your system manager specifies environments using the Environment Console.

2. Define the application by constructing a *project*. The project determines the basic structure of your application, including the design of the user interface and the functionality of the application. This phase is where you do the majority of your development work, creating classes, defining service objects, designing windows, writing methods, and testing and debugging the application. You construct projects using the iPlanet UDS Workshops.

3. Create the *configurations* for the project by mapping the project to each environment in which it will run. Creating configurations optimizes the application for each particular environment. You can customize a single project for any number of different environments. After you have created the configurations, you can generate the application distribution for each environment. You create configurations and make application distributions using the Partition Workshop.
4. Install each *application distribution* in the appropriate environment. Then, after the application is installed and is being used by your end users, your system manager can monitor the performance of the applications within each environment and make any necessary adjustments. Your system manager deployed applications using the Environment Console.

You can think of the project as a blueprint for the system. The project specifies everything that is included in the application, but it cannot actually run on physical machines. A configuration applies the blueprint to a particular set of machines and resources. The result is an application distribution that is customized for a specific environment. A single project can have any number of configurations, providing customized versions of the application for each different environment.

The ability to have multiple configurations is extremely useful when you need to run the same application in different departments or in different physical locations. For example, imagine the firm using the Auction application has a London office that uses VAX computers while the New York office uses Sun workstations. Using the same project, we can create a London configuration and a New York configuration, each one tailored for the specific hardware and resources used by the particular office.

Repository Workshop

The Repository Workshop provides the support you need to collaborate with other developers on developing an application. By allowing you to create your own "workspaces," the Repository Workshop ensures that you can work independently of other developers who are collaborating with you on the same projects. Any changes you make in your workspace are not visible to other developers until you integrate them into the repository. And changes made by other developers are not visible to you until you update your workspace to synchronize the changes in your workspace with the system baseline.

Source code control commands allow you to "check out" any project component, which gives you exclusive write access to the component until you integrate your changes into the repository.

The Repository Workshop also functions as the control center of the iPlanet UDS Workshops. From the Repository Workshop, you can access any of the other workshops you need to examine or modify project components or configurations.

Project Workshop

A project is either an application definition, a service definition, or a library of reusable components. When a project is an application definition, it defines the application's user interface and specifies the application's programming logic. When a project is a service definition, it specifies the programming logic that creates a service that can be shared by any number of applications. When a project is a library, it simply provides definitions for use by other projects.

The Project Workshop lets you examine and modify existing projects, and create new ones. The Project Workshop provides tools for creating all the components of a project, including:

- classes
- interfaces
- service objects
- database cursors
- project constants

Creating Projects

Creating a new project consists mainly of defining its classes (see "Class Workshop" on page 78). However, you also need to identify the libraries that contain definitions you wish to include in your project and to set other project properties. In addition, the project's service objects provide iPlanet UDS with information about the external resources and existing 3GL applications with which your application will be interacting.

Testing Projects

A simple command lets you test the application by running the project in the development environment. As the application runs, you can interact with it exactly as the end user will. To examine the code while the application is executing, you can use the Debugger (described under "Debugger" on page 81).

Class Workshop

The Class Workshop allows you to create custom classes specifically for your project. The Class Workshop provides tools for creating all the elements of a class, including:

- attributes
- methods
- events
- event handlers
- class constants

Although iPlanet UDS provides automatic compilation, you can compile your class at any point while you are writing it to check for errors.

Interface Workshop

The Interface Workshop allows you to create interfaces for your project. An interface defines a set of class elements, without providing the code that implements them. The interface provides the method and event handler signatures that define a standard "interface" to an object. The code for the methods and event handlers in the interface is provided by the classes that *implement* the interface.

The Interface Workshop provides tools for creating all the elements of an interface, including:

- virtual attributes
- methods
- events
- event handlers
- interface constants

Window Workshop

The Window Workshop provides a visual editor for creating the windows for your user interface. In the workshop, you can build and test windows, although you need to use a separate workshop, the Menu Workshop described below, to create the menu bar for your window. The workshop itself consists of two separate windows: the tool palette window and the user window.

Tool Palette

The tool palette in the Window Workshop provides a wide range of fields that you can use to design your form. The fields on the tool palette include standard controls, such as buttons, lists, radio buttons, text fields, and images, as well as special compound fields, such as array fields (which display tabular information) and grid fields (which let you create portable windows).

To create a window, you simply select an item from the palette and place it directly on the window's form. Just like a drawing program, the workshop lets you arrange the fields on the form by moving them and resizing them as desired. Special commands let you align, format, and color the items on the form.

Menu Workshop

The Menu Workshop provides a visual editor for creating the menu bar for a window. To create or modify a menu bar, you build a hierarchy that represents the pull-down menus on the menu bar. Each menu can include any of the following:

- menu buttons—simple controls that display commands
- menu toggles —simple controls that displays a toggle which the end user can switch on or off
- menu lists—simple controls that display a set of options from which the end user can make one selection
- slide-off menus

Method Workshop

The Method Workshop provides tools for writing and editing the methods associated with a class. Methods are procedures defined specifically for operating on objects of a given class.

In the Method Workshop, you specify the name, return value, and parameters for the method, and then write the method itself using TOOL, the iPlanet UDS object-oriented programming language. With TOOL, you can write code that responds to the end user's interactions with the user interface, manipulates the display, accesses and updates databases, interacts with existing 3GL applications, uses transactions, and invokes other methods. See the *TOOL Reference Guide* for an overview of TOOL.

Although iPlanet UDS provides automatic compilation, you can compile your method at any point while you are writing it to check for errors. When the method is complete, you can use the Debugger (described below) to monitor the code while the method is executing.

Event Handler Workshop

The Event Handler Workshop provides tools for writing and editing the event handlers associated with a class. An event handler is a named block of TOOL code that provides programming to be executed in response to one or more events. The event handler provides reusable, modular event handling code that you can include in any number of event statements.

In the Event Handler Workshop, you specify the name and parameters for the event handler, and then write the event handler itself using TOOL. See *TOOL Reference Guide* for an overview of TOOL.

Cursor Workshop

The Cursor Workshop provides tools for writing and editing the cursor associated with a project. A cursor is a row marker that you can use for selecting and working with a set of rows from a database. The cursor definition consists of a name and a select statement that selects a set of rows from the database.

In the Cursor Workshop, you specify the name and place holders for the cursor, and then write the cursor source code itself using TOOL. See *TOOL Reference Guide* for an overview of TOOL.

Debugger

The iPlanet UDS Debugger was designed specifically for debugging TOOL code. Special features include the ability to set breakpoints for events and exceptions, and to debug multiple tasks in parallel.

The Debugger consists of the following windows:

Window	**Purpose**
Application	Lists the tasks currently executing for the application
Task	Displays code for the method in the task that is currently being executed, and allows you to set breakpoints on the code.
Variable	Displays the current values for the local variables in the method in the Task Window.

To help you monitor the flow of the application's execution, iPlanet UDS lets you set breakpoints on the following: statements, method entry and exit, event posting and handling, and exception raising.

As the application runs, the Debugger displays the code for each task in separate task windows. In each task window, you can set breakpoints, step through the code, and view the values of local variables. Because an application can execute any number of tasks concurrently, you may not wish to view them all simultaneously. Therefore, iPlanet UDS lets you decide which tasks to display and which tasks to hide.

Partition Workshop

The Partition Workshop lets you examine and modify configurations and generate the code to be installed in the environment. A configuration is a project that has been partitioned for a particular environment. When a project is divided into partitions, individual partitions can be assigned to run on particular nodes in the environment. This creates a distributed application. You create a separate configuration for each environment in which the application will run.

The Partition Workshop provides a visual editor that displays the configuration on the environment map created when the environment was originally defined.

Examining Partitions

To examine a configuration, you display the environment map. The environment map shows the installed partitions on each of the nodes in the environment. To view detail information on a particular partition, simply double-click on the partition to open its property sheet.

Creating Configurations

When you are ready to create a new configuration, you select an environment and iPlanet UDS automatically creates a default configuration for your project. Because many partitions can run only on a certain node (for example, your database is probably already installed on a particular node), iPlanet UDS automatically partitions your project and places the partitions on appropriate nodes. The Partition Workshop displays the default configuration, which you can then modify as necessary. You can make adjustments by dragging partitions into place on the environment map. You can also change settings for individual partitions by editing their properties.

Once you create a configurations, you can make an application distribution for configuration. The application distribution is a representation of the application outside of the repository that is used to install the application in an environment.

After the application distributions have been created, your system manager can install them in the appropriate deployment environments.

Environment Console

The Environment Console lets your system manager examine, modify, and create environments. An environment is a description of a particular arrangement of hardware and software. The Environment Console provides a visual editor that displays an environment as a network map. In the Partition Workshop, you use this same map to partition the application

Examining Environments

To examine an environment, the system manager displays the network map, which shows all the individual nodes included in the environment. For detailed information on a particular node, the system manager can simply double-click on that node to view its property sheet. For complex environments, the system manager also has the option of viewing the environment information in a condensed, textual format.

Creating Environments

To define an environment, the system manager adds nodes to a network map by dragging individual nodes into place. The system manager can then provide detailed information about the software, external resources, and existing 3GL applications on a particular node by filling in its property sheet.

The Environment Console also provides tools for managing the runtime environment. These tools enable your system manager to manage the entire environment from a single node.

Managing the iPlanet UDS Runtime Environment

To let the system manager monitor the runtime environment, the Environment Console displays the environment map, showing all the partitions running on each node. The system manager can monitor partition performance, monitor alerts from applications, and start and stop services. Summary information about the environment as a whole is provided automatically. For detail information on individual partitions, the system manager can simply double-click on the partition.

This manual does not describe the Environment Console. For information about examining, creating, and managing environments, see *iPlanet UDS System Management Guide*.

Express

Express, an add-on product for iPlanet UDS, is an application generation tool, which allows you to develop graphical models of objects in your business system and the window flow of your application. Express automatically generates an application based on these models. Subsequently, you can use the standard iPlanet UDS workshops to further refine and customize your final application. In iPlanet UDS installations that do not include Express, the Application Model and Business Model Workshops, described next, are not available.

Application Model Workshop

You use the Application Model Workshop to develop the windows that form the client side of your application. You can create one or more windows for each business object in your data model. You can retrieve data from multiple business objects into one window, or display data from each business object in separate windows. The Application Model Workshop provides a set of property dialogs in which you define window types, data interfaces, and links between windows.

Business Model Workshop

The Business Model Workshop provides tools for drawing and defining a business model. A business model represents all the objects in your business system and the relationships between them.

iPlanet Integration Server

iPlanet Integration Server is an add-on product for iPlanet UDS that consists of a suite of business integration tools for integrating and coordinating heterogeneous applications. The tools and software components provided with iPlanet Integration Server let you integrate newly developed applications, legacy applications, and off-the-shelf packages into business processes that are automated and controlled by a process engine.

An iPlanet Integration Server system is composed of two subsystems, a process management system and an XML-based backbone system.

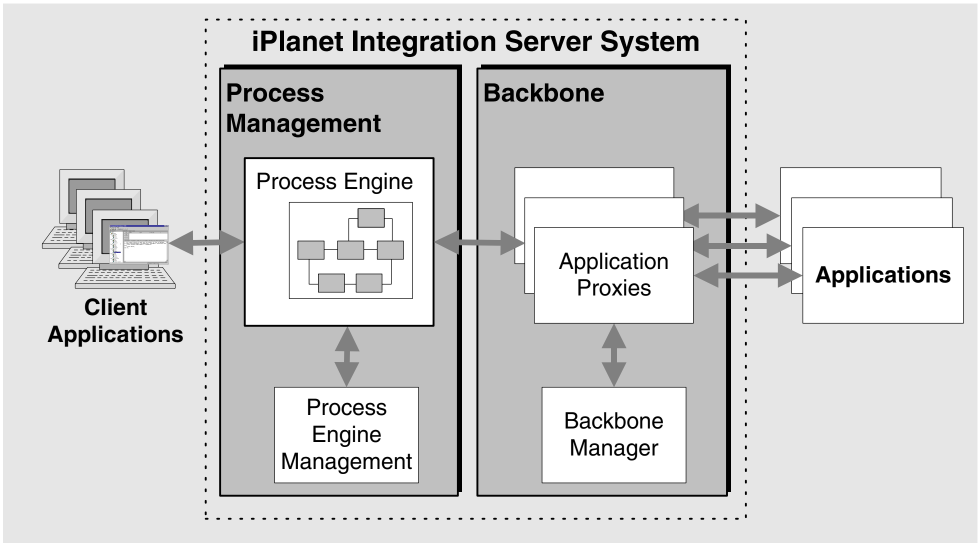
iPlanet Integration Server Process Management System The iPlanet Integration Server process management system (formerly known as Conductor) provides a set of tools and software modules that support the development, execution, and management of business processes. The heart of this system is the iPlanet Integration Server process engine, which controls and manages business processes from beginning to end, coordinating the work of the different resources or applications that participate in the processes.

iPlanet Integration Server customers use the process management system to:

- develop process logic with the graphical process development workshops
- manage sessions and processes, and the engine itself, using the iIS Console and other tools
- build applications, called process clients, that make direct API calls to the process engine, using the process client APIs (iPlanet UDS, CORBA/IIOP, JavaBeans, ActiveX, or C++)

Backbone system The iPlanet Integration Server Backbone provides a set of tools and software modules that use XML messaging over HTTP or JMS to simplify communication and coordination between applications. An iPlanet Integration Server backbone can support different styles of integration, but the backbone is always installed on top of the iPlanet Integration Server process engine runtime. The heart of a backbone system is a set of application proxies that perform message brokering and data transformation on behalf of applications. For business process support, proxies interact with the iPlanet Integration Server process engine on behalf of any applications that participate in a common business process. The main purpose of these interactions is to communicate the initiation and completion of work activities.

Figure 1-11 iPlanet Integration Server System and Subsystems



iPlanet UDS provides adapters as well as an adapter toolkit to integrate packages or custom applications that lack a native XML/HTTP interface into an iPlanet Integration Server backbone.

iPlanet Integration Server customers use the iPlanet Integration Server Backbone primarily to:

- provide an XML/HTTP interface between proxies and applications
- configure application proxies to participate in a managed business process
- develop, test, debug, store, and manage XML documents and the XSL stylesheets used for message transformation between applications

XML/XSL Workshop

The Fusion XML/XSL Workshop is an interactive tool for creating, editing, testing, and debugging the XSL stylesheets that you register with a Fusion backbone. You can create XML and XSL documents in the Workshop, or import Fusion proxy documents from files. The XML documents you use to test your XSL stylesheets can be actual proxy documents or sample documents that you create just for testing purposes.

Process Development Workshops

User Profile Workshop

The User Profile Workshop allows you to customize a default user profile supplied by the Fusion process engine. The workshop is used to specify the important user-information needed by assignment rules to determine who should be permitted to perform process activities.

Assignment Rule Workshop

The Assignment Rule Workshop allows you to create assignment rules and group them into dictionaries. The workshop is used to specify simple role-based rules as well as more sophisticated rules. More sophisticated rules—that depend on user profile attributes, process attribute values, or any condition that might determine user access to one or more activities—are implemented in the workshop by writing an Evaluate method. The assignment rule dictionaries are needed by process developers to specify activity properties.

Application Dictionary Workshop

The Application Dictionary Workshop allows you to create application dictionaries. The workshop is used to create items that define the work associated with activities, including the process attributes needed to perform the work. Application dictionaries are needed by process developers to specify activity properties.

Process Definition Workshop

The Process Definition Workshop allows you to create process definitions. It is a tool in which the developer creates a visual model of a process and specifies the properties of each of the components of the model: activities, routers, timers, and so on. The workshop is also used to write methods that specify process logic, such as trigger methods and router methods. This is also where process attributes, an important design element, are specified.

Validation Workshop

The Validation Workshop allows you to write a ValidateUser method that authenticates users when they open a session with the Fusion process engine, and which may also populate users' profiles by extracting data from an organization database.

Chapter 2

Using the iPlanet UDS Workshops

This chapter provides general information about how to use the iPlanet UDS Workshops.

In this chapter, you will learn how to:

- use the UDS Control Panel to set common environment variables
- start the UDS Workshops
- use UDS windows
- use UDS names
- use UDS data types
- use UDS Help
- set workshop preferences
- leave the iPlanet UDS Workshops

The chapter also provides information about international support.

Before Using the iPlanet UDS Workshops

Before you can use the iPlanet UDS Workshops, you must have access to a *repository*. A repository is a database that stores the iPlanet UDS libraries as well as all the projects that you create with iPlanet UDS. When your system manager sets up the distributed iPlanet UDS development environment, she creates at least one central repository. See your system manager for guidance about which repository you should use. See Chapter 3, "Using the Repository Workshop," for further information about repositories and how to create and use them.

The following two sections, "Using the iPlanet UDS Control Panel" and "Setting Environment Variables Without the Control Panel" on page 100 describe how to specify your default repository as well as several other settings you may need to specify before starting the iPlanet UDS Workshops.

Using the iPlanet UDS Control Panel

The iPlanet UDS Control Panel provides a simple user interface that allows you to view and/or set the most commonly used iPlanet UDS environment variables. For Windows, the Control Panel allows you to view and change the current settings. For other platforms, the Control Panel only provides read access to the settings. For these platforms, you must set the environment variables as described under "Setting Environment Variables Without the Control Panel" on page 100.

Opening the Control Panel

To start the Control Panel from Windows, double-click the iPlanet UDS Control Panel icon, or choose the Start > Programs > iPlanet UDS > iPlanet UDS Control Panel command.

To start the Control Panel from UNIX or VMS, use the `fcontrol` command. The portable syntax is:

```
fcontrol
```

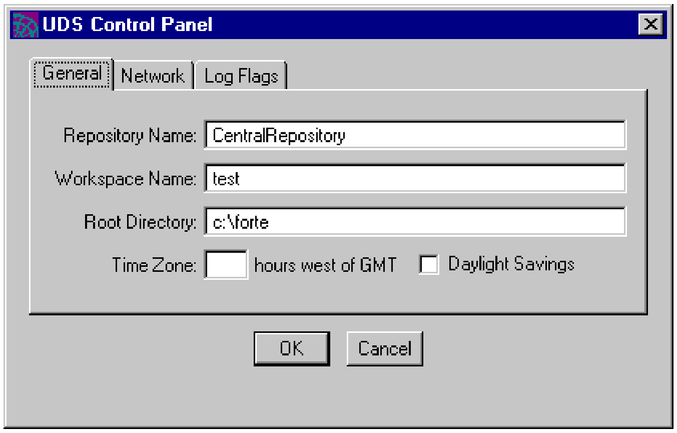
The OpenVMS syntax is:

```
VFORTE FCONTROL
```

The Control Panel Window

The Control Panel window is a dialog box, shown below, where you can view or change the iPlanet UDS environment variable settings.

Figure 2-1 Control Panel



Repository Name: CentralRepository

Workspace Name: test

Root Directory: c:\forte

Time Zone: hours west of GMT ☐ Daylight Savings

If you use the utility on Windows, the window displays OK and Cancel buttons. Any changes you make to settings take effect when you click the OK button. Clicking the OK button also closes the dialog.

If you use the utility on any platform other than the Windows, you cannot change any of the settings, so the Control Panel simply displays a Close button. Click the Close button to close the dialog when you have finished viewing the settings.

When you open the Control Panel, the Control Panel window displays a tab folder with the following tab pages:

Tab Page	Available Settings
General	Repository name, workspace name, root directory, and time zone.
Network	Model node, node name, name server address, and communication provider.
Log flags	Default logger settings for the iPlanet UDS session.

The General tab page is displayed first. To display or set the network settings, click the Network tab. To display or set the log flags setting, click the Log Flags tab. Do not click the OK on the Control Panel until you are ready to close the Control Panel.

The following sections provide detailed information about the settings you can view or change on each of the tab pages.

Closing the Control Panel

On Windows, you can use either the OK or Cancel button to close the Control Panel. The OK button changes the settings as specified and then closes the Control Panel. The Cancel button discards the new settings and then closes the Control Panel.

On all other platforms, simply click the Close button to close the Control Panel.

General Tab Page

The following table lists general settings you can control with the Control Panel and shows the environment variable that corresponds to each setting:

General Setting	Environment Variable
Repository Name	FORTE_REPOSNAME
Workspace Name	FORTE_WORKSPACE
Root Directory	FORTE_ROOT
Time Zone	FORTE_TIMEZONE
Daylight Savings	FORTE_TIMEZONEDST

The following sections provide information on the individual settings.

Repository Name

The repository name setting specifies the name of the repository to be used as the default repository for all iPlanet UDS commands that use a repository.

To specify the repository name, enter one of the following options:

- "Central Repository" for the default central repository.
- A repository service name.
- The name of a private B-tree or shadow repository using the following format: `bt:` *private_repository_name*.

  Specify the private repository name as the root file name of the B-tree repository (that is, the full path name of the file without the trailer).

By default, the repository name has the value “CentralRepository,” which represents the default central repository in the distributed environment.

The following examples illustrate how to specify a B-tree repository name:

Platform	Command Syntax
UNIX	`bt:$FORTE_ROOT/repos/myShadow`
OpenVMS	`BT:FORTE_ROOT:[REPOS]MYSHADOW`
Windows	`bt:\forte\repos\myshadow`

On Windows platforms, if you choose the `New Shadow` command in the Repository Workshop, iPlanet UDS automatically resets the value of the Repository Name setting (and the FORTE_REPOSNAME environment variable) to the shadow name. As a result, if you leave and then reenter iPlanet UDS, the same shadow repository will automatically be open.

The `-fr` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Repository Name setting. For information about the `-fr` flag, see “Flags for ftcmd, forte, ftexec, and ftclntws Commands” on page 118.

Workspace Name

The Workspace Name setting specifies the name of the workspace to use for the default workspace when you start the iPlanet UDS Workshops or when the you give an `Open` command in Fscript.

The `-fw` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Workspace Name setting. For information about the `-fw` flag, see “Flags for ftcmd, forte, ftexec, and ftclntws Commands” on page 118.

If you do not specify a value for Workspace Name and do not specify a value with the `-fw` flag on the command line, iPlanet UDS opens the Repository Workshop without a workspace and starts Fscript with the prefabricated workspace called “FirstWorkspace.”

On Windows, the value of the Workspace Name setting (and the corresponding FORTE_WORKSPACE environment variable) is automatically set by the iPlanet UDS workshops to the last workspace that was opened. As a result, if you leave and then reenter iPlanet UDS, the same workspace you left will automatically be open.

You can override the Workspace Name setting by using the `-fw` flag on the commands that start the iPlanet UDS Workshops. In the Repository Workshop, you can open any workspace in the repository by choosing the File > Open Workspace command (see "Using Workspaces" on page 168).

Root Directory

The Root Directory setting specifies the root directory path for iPlanet UDS. All the files that make up the iPlanet UDS system as well as the default locations for files that iPlanet UDS writes to and reads are stored in the root directory. The value of this setting is usually specified during the installation process using the FORTE_ROOT environment variable. The Control Panel displays the current setting mainly for your information only. You should not specify a new root directory unless you are moving your entire iPlanet UDS installation.

Time Zone and Daylight Savings

The Time Zone setting specifies the time zone for the machine as the number of hours west of GMT. On some systems (on UNIX and on PCs running PC/NFS), this is not needed, as the system provides an accurate value for the time zone setting. However, if you do specify a value for this setting, note that the setting will override any other value it gets from the operating system.

You must set the Time Zone setting to an integer value between +11 and -13. For Pacific time, set the value to 8, and for Eastern time, set the value to 5. For time zones east of GMT, set the Time Zone setting to a negative number. If daylight savings time is in effect, set the Daylight Savings toggle to on.

The Daylight Savings toggle specifies whether or not daylight savings is in effect. If the toggle is set to on, one hour is added to the current time setting.

Network Tab Page

The following table lists the network settings you can control with the Control Panel and shows the environment variable that corresponds to each setting:

Network Setting	**Environment Variable**
Model Node	FORTE_MODELNODE
Node Name	FORTE_NODENAME
Name Server Address	FORTE_NS_ADDRESS
Communication Provider	FORTE_PROVIDERS

The following sections provide information on the individual settings.

Model Node

The Model Node setting specifies the name of the model node, in the active environment, which this node uses for its definition. The Model Node is usually set during installation. If you do not specify a model node name, the node is not treated as a model node.

The `-fmn` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Model Node setting. For information about the `-fmn` flag, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

Node Name

The Node Name setting specifies the name of the node on which you are running. The Node Name setting in the Control Panel is used only on Windows 3.1; on UNIX, OpenVMS, and NT servers, the host name is taken from the hosts file. The Node Name is usually set during installation, but you can change the value from the Control Panel if necessary.

The `-fn` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Node Name setting. For information about the `-fn` flag, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

Name Server Address

The Name Server Address setting specifies the address of the iPlanet UDS name server process. The Name Server Address is usually set during installation, but you can change the value from the Control Panel if necessary.

The syntax for the Name Server Address setting is:

address [::*protocol_name*]

The syntax of *address* is protocol dependent, as shown in the following table.

Protocol	Address Syntax
TCP/IP	*machine_name*:*port_number*
DECnet	*machine_name*:*object_name*
UNIX Domain	*path_name*

The optional *protocol_name* allows you to use a different protocol than the default for the platform. The default protocol for OpenVMS is DECnet and for all other platforms TCP/IP.

The *protocol_name* is one of the following values:

TCP/IP
DECnet
UNIX Domain

CAUTION If you change your name server address, the next iPlanet UDS application that you start, including the iPlanet UDS Workshops, will use the new value. This effectively changes you to a different iPlanet UDS environment. You should discuss this with your system manager.

For further information about the name server, see the *iPlanet UDS System Management Guide*.

Communication Provider

The Communication Provider setting specifies the communication transport providers to use for the node. A transport provider is a program that enables iPlanet UDS to access a particular communication package.

In the Control Panel, the only platform for which you can change the communication provider is PC Windows. The default communication provider for PC Windows is Winsock. You can choose PC-NFS if appropriate.

Log Flags Tab Page

As you develop and test applications in the iPlanet UDS Workshops, iPlanet UDS logs messages in the trace window or log file as specified by FORTE_LOGGER_SETUP environment variable (see "Common Environment Variables" on page 105). If you did not specify a log file name with FORTE_LOGGER_SETUP, iPlanet UDS logs the messages in the trace window.

The Log Flags setting in the Control Panel allows you to specify the filter settings used for logging the messages. This is equivalent to the log settings you can specify the FORTE_LOGGER_SETUP environment variable. However, the Log Files setting does not allow you to specify the log file names. Instead, all messages are logged in the default log file, "stdout."

CAUTION If you change any of the filter settings using the Control Panel, the new settings override any file specifications set by the FORTE_LOGGER_SETUP environment variable. All messages will be logged in the default log file, "stdout."

To change the filter settings for an individual message, edit the fields in the array row as follows:

Field	How to Fill It in
Message	Choose the message type from the drop list.
Service	Choose the service type from the drop list.
Group	Enter integers in the two data fields to specify a range. The integers can be from 1 to 63.
Level	Enter an enter from 1 to 255.

See "-Fl Flag (iPlanet UDS Logger)" on page 795 for specific information about each of these settings.

Inserting and Deleting Log Settings

You can request an additional log setting by inserting a new row in the array field or eliminate a log setting by deleting the row from the array field. The Insert button adds a new row above the selected row, using default values for each of the fields. The Delete button removes the currently selected row.

The LogMgr object (described in the Framework Library online Help) examines the Log Flags setting (or the FORTE_LOGGER_SETUP environment variable) and based on the information it finds, opens one or more text files or display windows. If you do not specify the log flags with the LogFlags setting or FORTE_LOGGER_SETUP (and you have not specified in the `-fl` flag in your command line), no specially filtered messages are logged.

The `-fl` flag on the commands that run iPlanet UDS, such as `ftcmd`, `ftexec`, `forte` and `fscript`, overrides the value the Log Flags setting. For information about the `-fl` flag, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

Changing Default Filter Settings

If you wish to change the default filter settings at any point during your development session, you can use the Modify Log Flags command in the Repository Workshop (see "Modifying Log Flags" on page 195). The Modify Log Flags command opens a window, where you view and/or change the filter settings in an array field.

Modifying Log Specifications at Runtime

You can also modify the types and detail level of messages being logged while the program is running. The `ModifyFlags` method defined on the `LogMgr` class provides a runtime interface to the same settings the Log Flags setting (and the `-fl` flag on iPlanet UDS command lines) provides at application start-up. The Environment Console provides an interface to a running server's LogMgr object and can be used to modify the LogMgr settings of the server being monitored.

Setting Environment Variables Without the Control Panel

You may find you need to set environment variables for Windows that are not included in the iPlanet UDS Control Panel. To do so, you must use the operating system's standard format. For platforms other than Windows, you must set all your environment variables using the operating system's standard format.

The following sections provide information about setting environment variables in the operating systems supported by iPlanet UDS.

"Common Environment Variables" on page 105 provides a list of the most commonly used environment variables. For a complete list of the environment variables, see the *iPlanet UDS System Management Guide*.

Setting Environment Variables on NT

On Windows and Alpha NT, the default settings for the iPlanet UDS environment variables are set in the Registry. NT allows you to set environment variables in several different places, all of which are described below. We recommend that you set your environment variables using the Control Panel or the Registry Editor (regedit). In general, you should avoid using the NT Control Panel.

The order of precedence is as follows:

1. individual process window (set at the DOS command prompt)
2. NT Control Panel's User variables
3. NT Control Panel's System variables
4. Registry's Current User Tree (HKEY_CURRENT_USER)
5. Registry's Global Tree (HKEY_LOCAL_MACHINE)

Using the Registry

The environment variables in the Registry are set in two different directories.

Registry's Global Tree iPlanet UDS machine-wide information and default iPlanet UDS environment variables are kept in the iPlanet UDS Global Tree branch:

HKEY_LOCAL_MACHINE\Software\ForteSoftwareInc\Forte\version number

Registry's User Tree All user-specific iPlanet UDS environment variables are kept in the iPlanet UDS User Tree branch:

HKEY_CURRENT_USER\Software\ForteSoftwareInc\Forte

Environment variable settings in the iPlanet UDS User Tree override those in the iPlanet UDS Global Tree. Unless you are a system administrator who is setting up a machine for use by several users, we recommend that you set your environment variables in the *User Tree*.

To change the value of an environment variable or add a new one, use regedit.exe to open the Registry Editor and modify the iPlanet UDS User Tree.

Using the NT Control Panel

The NT Control Panel allows you to set any of the iPlanet UDS environment variables. Like the Registry, the NT Control Panel contains System and User sections. The settings in the User section override the settings in the System variables.

Settings in the NT Control Panel override the settings in the Registry (and therefore settings in the iPlanet UDS Control Panel). Therefore, we recommend that you do *not* use the NT Control Panel to set your environment variables. Setting environment variables in too many places can make your setup confusing and inconsistent.

Using the DOS Command Line

For an individual process window, you can set environment variables using the `set` command in DOS. Using the `set` command for an individual window overrides the environment variable settings for that individual process.

Using the iPlanet UDS Control Panel

You can set the most commonly used environment variables using the iPlanet UDS Control Panel as described under "Using the iPlanet UDS Control Panel" on page 92. Setting environment variables in the iPlanet UDS Control Panel changes the value of the equivalent environment variable in the User Tree of the Registry (*not* in the NT Control Panel).

Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Environment Variables on Windows 95

On Windows 95, the default settings for the iPlanet UDS environment variables are stored in the User Registry.

Using the Registry

The following file in the Registry contains the iPlanet UDS environment variables:

HKEY_CURRENT_USER/Software/ForteSoftwareInc/Forte

To change the value of an environment variable or add a new one, use `regedit.exe` to open the Registry Editor and modify this file.

Using autoexec.bat

You can also set iPlanet UDS environment variables in the `autoexec.bat` file. Setting environment variables in the `autoexec.bat` file overrides the settings in the User Registry. For consistency, we recommend that you set your environment variables in the Registry as described above, and *not* in the `autoexec.bat` file.

Using the iPlanet UDS Control Panel

You can set the most commonly used environment variables using the iPlanet UDS Control Panel as described under "Using the iPlanet UDS Control Panel" on page 92. Setting an environment variable in the iPlanet UDS Control Panel changes the value of the corresponding environment variable in the Registry. Therefore, environment variables set in autoexec.bat file will override those set by the iPlanet UDS Control Panel.

In summary, the following environment variable settings take precedence:

1. `autoexec.bat` file
2. User Registry

Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Environment Variables on UNIX

On UNIX, the default settings for the iPlanet UDS environment variables are stored in the following files:

FORTE_ROOT/fortedef.csh

FORTE_ROOT/fortedef.sh

These two files are created by the iPlanet UDS Installer, and save your input for the iPlanet UDS system information.

Using fortedef

To change the default settings or add new environment variable settings, simply edit the appropriate fortedef file.

Using the Command Line

You can also enter settings for iPlanet UDS environment variables on the command line. Following the UNIX standard, environment variable settings specified at the command line override those in the fortedef file.

Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Setting Logical Names on OpenVMS

On OpenVMS, the default settings for the iPlanet UDS logical names and DCL foreign symbols are stored in the following file:

```
FORTE_ROOT:[INSTALL.SCRIPTS]FORTE_LOGIN.COM
```

Using the FORTE_LOGIN.COM File

To change the default settings or add new logical names or symbols, simply edit the `FORTE_LOGIN.COM` file or create a new file:

```
FORTE_ROOT:[INSTALL.SCRIPTS]SITE_LOGIN.COM
```

In order to define the FORTE_ROOT logical name you can simply execute:

`@SYS$LIBRARY:FORTE_LOGIN_V`*version_number*

Using Your Personal login.com File

If you use iPlanet UDS frequently, you may wish to include the following command within your personal `LOGIN.COM` file:

`$ @SYS$LIBRARY:FORTE_LOGIN_V`*version_number*

Using the Command Line

You can also define settings for iPlanet UDS logical names on the command line. Logical name settings specified at the command line override those in the `FORTE_LOGIN.COM` file. The logicals defined in `FORTE_LOGIN.COM` are stored in two logical name tables:

`FORTE_PRCTABLE_V`*version_number*

`FORTE_GBLTABLE_V`*version_number*

Therefore, if you use the DEFINE command to set your own logicals in either the LNM$PROCESS table (with the default /PROCESS qualifier to DEFINE) or the LNM$JOB table (with the /JOB qualifier to DEFINE) you can leave the FORTE_PRCTABLE_V*version_number* and FORTE_GBLTABLE_V*version_number* tables undisturbed for other users.

Note that iPlanet UDS command-line flags that specify the same settings as environment variables, such as those that set the repository or workspace, always override the environment variable settings.

Common Environment Variables

FORTE_FTLAUNCH_PORT

Specifies the port number for the launch server.

`FORTE_FTLAUNCH_PORT` *tcp_port_number*

If this variable is set, then ftcmd uses the specified port, instead of the default (port 3783), to contact the launch server.The port number must be a numeric value which is a legal and unused TCP port for the underlying platform and operating system.

Because this variable should be set individually for each iPlanet UDS user, it should not be set in a centralized fortedef file which is accessed by multiple iPlanet UDS users.

FORTE_GC_SPECIAL

Sets the memory allocated for iPlanet UDS partitions. The syntax for this variable is identical to the syntax for the -fm startup flag (also used to set memory).

`FORTE_GC_SPECIAL` (*memory_option* {: | =} *number* [, *memory_option* {: | =} *number*])

The syntax and default values for the memory flags are described in "-Fm Flag (Memory Manager)" on page 799. An example follows:

FORTE_GC_SPECIAL (n:5000, x:10000)

Memory flags are set according to the following rules:

- If the `-fm startup` flag is set, then FORTE_GC_SPECIAL is ignored.
- If the `-fm startup` flag is not set, then the setting for FORTE_GC_SPECIAL is used.
- Any memory flags that are not explicitly set use the default values.

FORTE_LOGGER_SETUP

Specifies the default logger settings for the iPlanet UDS session.

`FORTE_LOGGER_SETUP` *file_name*(*file_filter*)[*file_name*(*file_filter*)...]

You can override the settings for FORTE_LOGGER_SETUP by using the `-fl` flag on any of the iPlanet UDS command lines. In fact, we recommend that you use the `-fl` flag rather than resetting FORTE_LOGGER_SETUP when you want to change message filtering.

The `file_name` argument is any valid file name where you want to log certain messages. The special file names "`%stdout`" and "`%stderr`" log the messages to standard output or standard error, respectively.

On Windows only, you can use the name "`%stdwin`" to create a simple, scrollable output window for textual output. "`%stdwin`" is particularly useful when using the FORTE_LOGGER_SETUP variable to specify an alternative file for the output from Fscript or the development environment.

Each file name is associated with a *file_filter*. The `file_filter` argument may include one or more filter options separated by a space. The syntax for the `file_filter` parameter is the following:

message_type[:*service_type*[:*group_number*[:*level_number*]]]

For information on the file filters, see the description of the LogMgr class in the Framework Library online Help. The LogMgr object examines the FORTE_LOGGER_SETUP environment variable and opens one or more text files or display windows based on the settings it finds there. If FORTE_LOGGER_SETUP is not set (and you have not specified in the `-fl` flag in your command line), no specially filtered messages are logged.

Modifying Logging At Runtime

You can also modify the types and detail level of messages being logged while a program is running. The ModifyFlags methods defined on LogMgr provides a runtime interface to the same settings that FORTE_LOGGER_SETUP (and the `-fl` flag on iPlanet UDS command lines) provides at application start-up. The Environment Console provides an interface to a running server's LogMgr object and can be used to modify the LogMgr settings of the server being monitored. Finally, the Repository Workshop allows you to change LogMgr setting while developing iPlanet UDS applications.

FORTE_MODELNODE

Specifies the name of a model node in the active environment, which the current node uses for definition.

`FORTE_MODELNODE` *model_node_name*

This variable is normally set during the installation process. If the environment variable is not set, the node is not treated as a model node. See also FORTE_NODENAME.

FORTE_NODENAME

(For PCs only) Specifies the name of the current node. (On UNIX and OpenVMS, the host is taken from the hosts file.)

`FORTE_NODENAME` *node_name*

This variable is normally set during the installation process.

FORTE_NS_ADDRESS

The FORTE_NS_ADDRESS environment variable is used one way for the Environment Manager process and another way for all other processes.

- For the Environment Manager, FORTE_NS_ADDRESS specifies one or more network addresses at which the Environment Manager listens for incoming requests.
- For *all other* processes, FORTE_NS_ADDRESS specifies one or more addresses to use when contacting the Environment Manager. Addresses are tried in the order they are specified. If a connection fails while in use, the next address is automatically tried.

In both cases, however, the syntax is the same:

`FORTE_NS_ADDRESS` *address* [::*protocol_name*][;*address* [::*protocol_name*]...]

For further information, see the *iPlanet UDS System Management Guide*.

FORTE_PROVIDERS

Specifies the transport provider for a given platform. Normally you need not set this variable as the installation does so for you.

`FORTE_PROVIDERS` *protocol_name*

However, you can use this variable to specify a different transport provider for your platform. For example, for Windows the default communication protocol is Winsock, but you can specify the use of the native interface to the PC-NFS product. For Windows the valid values for this environment variable are:

Communication Provider	Communication Protocol
W3TPSUN	PC-NFS
W3TPWSS	Winsock

FORTE_REPOSNAME

Specifies the name of the repository that will be used as the default repository for all iPlanet UDS commands that use a repository.

`FORTE_REPOSNAME` *repository_name*

To specify the repository name, enter:

- "Central Repository" for the default central repository
- A repository service name.
- The name of a private C-tree or shadow repository using the following format: `ct:` *private_repository_name.*

  Specify the private repository name as the root file name of the C-tree repository (that is, the full path name of the .dat or .idx file without the trailer).

By default, this has the value "CentralRepository" which represents the default central repository in the distributed environment.

On Windows, if you give a `New Shadow` command in the Repository Workshop, this automatically resets the value of FORTE_REPOSNAME to the shadow name. This way, if you leave and then reenter iPlanet UDS, the same shadow repository will automatically be open.

Note that the `-fr` flag on several iPlanet UDS commands, such as `forte` and `fscript`, overrides the value of FORTE_REPOSNAME.

FORTE_ROOT

Specifies the root directory path for iPlanet UDS.

`FORTE_ROOT` *file_specification*

All the files that make up the iPlanet UDS system as well as the default locations for files that iPlanet UDS writes and reads are stored in FORTE_ROOT. The value of this environment variable is set during the installation process; you should not reset it.

FORTE_STACK_SIZE

Sets the thread stack size in bytes iPlanet UDS and Posix threads.

`FORTE_STACK_SIZE` *integer*

On NT, the system default is 1MB and is not adjustable. On other servers the default is 28K to 40K. Motif clients have a minimum of 100K. You cannot set a size below the system default.

Generally, you should not increase this value. Instead, avoid programming practices that require an increased stack; for example, avoid using deeply nested or recursive method invocations.

You can increase the stack size if necessary. Because the new stack size you specify is used for every thread, it will increase memory usage by the stack size times the number of concurrent active threads. iPlanet UDS rounds the value up to the nearest system MMU (Memory Management Unit) pagesize (8K on many machines, but sometimes more or less). iPlanet UDS adds an MMU pagesize guardword to the size; this page is memory protected to try and catch stack overflow cases.

FORTE_TIMEZONE

Specifies the time zone for the machine as the number of hours east or west of GMT.

`FORTE_TIMEZONE` *integer*

This variable is only needed on OpenVMS and in the southern hemisphere on Win3.1. On UNIX systems and PCs running PC/NFS), the system provides the time zone setting. However, if you set this environment variable, its value overrides the operating system timezone.

Acceptable values are integers between -23 and 23. For example, use 8 for Pacific time and 5 for Eastern time. For time zones east of GMT, set this to a negative number. If daylight savings time is in effect, use the FORTE_TIMEZONEDST environment variable in addition.

FORTE_TIMEZONEDST

Specifies whether or not daylight savings is in effect. If it is set to TRUE, one hour is added to the current time setting.

`FORTE_TIMEZONESDT` {`TRUE | FALSE`}

This variable is only needed on OpenVMS.

FORTE_WORKSPACE

Specifies the name of the iPlanet UDS workspace to use for the default workspace for the `forte` command or when the user gives an `Open` command in Fscript.

`FORTE_WORKSPACE` *workspace_name*

If you do not set this environment variable and do not specify a value with the `-fw` flag on the `forte` or `fscript` command, iPlanet UDS opens the Repository Workshop without a workspace and starts Fscript with the prefabricated workspace called “FirstWorkspace.”

On Windows, the value of this environment variable is automatically set by the iPlanet UDS workshops to the last workspace that was opened. This way, if you leave and then reenter iPlanet UDS, the same workspace you left will automatically be open.

You can override the setting of FORTE_WORKSPACE by using the `-fw` flag on the `forte` and `fscript` commands. In the Repository Workshop, you can open any workspace in the repository with the Open Workspace command on the File menu.

Starting the iPlanet UDS Workshops

You can use the iPlanet UDS Workshops in one of two modes: standalone mode or distributed mode.

Standalone Mode

In standalone mode, the iPlanet UDS Workshops run only on your client workstation and you do not have access to the server nodes within your development environment. You can access all the workshops except the Partition Workshop (the Partition Workshop needs access to your development environment). In addition, in standalone mode, you *cannot* test your application by running it as a distributed application within the development environment.

In standalone mode, the repository you use must be a local repository; you *cannot* use a distributed repository. If you try to run in standalone using a central repository, you will get an error.

Distributed Mode

In distributed mode, the iPlanet UDS Workshops run on your client workstation and on the server nodes in your development environment. You have access to all the workshops, including the Partition Workshop, and you can test your application by running it as a distributed application within the development environment.

In distributed mode, the repository you use can either be local or distributed.

The following sections provide detailed information about starting the workshops in both standalone and distributed mode on the following operating systems:

- Windows 95, Windows NT, Alpha NT
- UNIX
- OpensVMS

iPlanet UDS Launch Server

On all platforms except OpenVMS, iPlanet UDS provides commands and icons for starting the iPlanet UDS Workshops that use the iPlanet UDS Launch Server. The Launch Server is an iPlanet UDS service that runs on client nodes and starts iPlanet UDS applets and applications. The Launch Server can run several iPlanet UDS applications under the same operating system process, which enables the applications to start faster and use less memory. It can also ensure that the most recent copy of an application is installed on the client; if it is not, the Launch Server automatically downloads the newer copy.

The Launch Server is started automatically when you invoke the command to start the Workshops. However, on UNIX you must set your FORTE_FTLAUNCH_PORT environment variable to a user port before the Launch Server starts. See "Starting the Workshops on UNIX" on page 114 for information about the FORTE_FT_LAUCH_PORT environment variable.

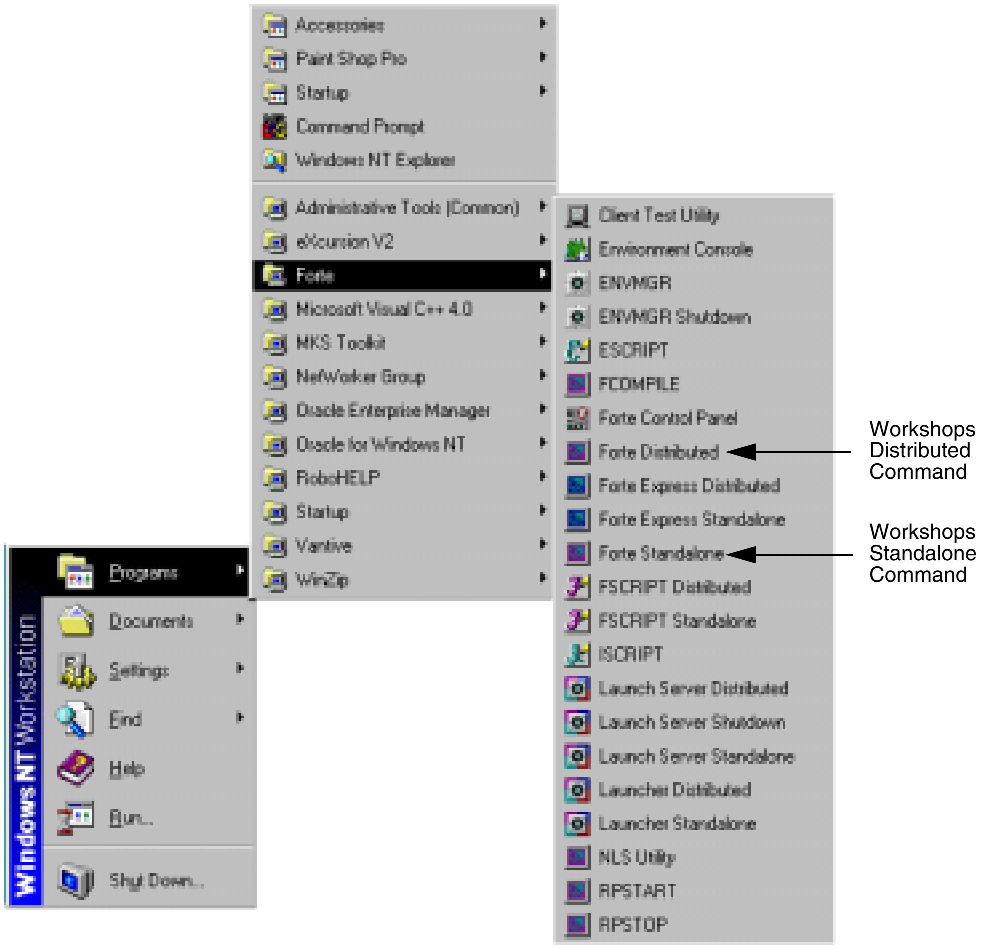
iPlanet UDS Launcher Application

For iPlanet UDS developers, we generally recommend starting the iPlanet UDS Workshops using the icons and commands set by up the iPlanet UDS installer. However, you can also start the Workshops using the iPlanet UDS Launcher Application. The iPlanet UDS Launcher Application is an iPlanet UDS application that allows end users to start and shutdown applications; the iPlanet UDS Workshops are included in the list of applications that can be started with the Launcher Application. See "Using the iPlanet UDS Launcher Application" on page 122 for information on using the iPlanet UDS Launcher Application to start the iPlanet UDS Workshops.

Starting the Workshops on Windows 95, Windows NT, Alpha NT

For Windows 95, Windows NT, and Alpha NT, the iPlanet UDS Installer installs a group of iPlanet UDS shortcuts in your Start menu, under Programs > iPlanet UDS. These shortcuts include the following shortcuts for starting the iPlanet UDS Workshops in standalone and distributed mode In addition, the iPlanet UDS Installer creates a parallel set of icons.

Figure 2-2 Workshops Standalone and Distributed Shortcut Commands



Using Shortcuts

The iPlanet UDS Standalone shortcut invokes the following `ftexec` command, starting the iPlanet UDS Workshops in standalone mode and using the default repository:

FTEXEC.EXE -fs -fss -fi bt:%FORTE_ROOT%\userapp\forte\c10\forte

The iPlanet UDS Distributed shortcut invokes the following `ftcmd` command, starting the iPlanet UDS Workshops in distributed mode and using the default repository:

FTCMD.EXE run forte

If you wish to use these shortcuts as they are, you can start the iPlanet UDS Workshops as follows.

➤ **To start the iPlanet UDS Workshops on Windows**

1. Using the iPlanet UDS Control Panel, make sure the Repository setting specifies the name of the repository you wish to use. If not, enter the correct repository name. For standalone mode, the repository must be a local repository. For distributed mode, the repository can be a local or a distributed repository.

   At this time, you can set any other environment variables you wish.

2. Select the Workshops Standalone or Workshops Distributed menu command (or the corresponding icon).

3. The Repository Workshop opens.

If you wish to use flags on the `ftcmd` command that are other than the default, you must edit the shortcuts.

Editing Shortcuts

To change the `ftexec` or `ftcmd` command in a shortcut, you must edit the shortcut using the File > Properties command. The Properties command opens the properties dialog for the shortcut; the Target field contains the `ftcmd` command line. See "Ftexec Command" on page 118 for complete information about the flags you can set for the `ftexec` command. See "Ftcmd Command" on page 115 for complete information about the flags you can set for the `ftcmd` command.

Starting the Workshops on UNIX

On UNIX, you start the iPlanet UDS Workshops by using the `forte` command. By default, the `forte` command starts the workshops in distributed mode. To start the workshops in standalone mode, you can use the `-fs` flag.

The syntax for the `forte` command is:

`forte` [`-fs`] [`-fr` *repository*] [`-fw` *workspace*] [`-fnd` *node_name*] [`-fmn` *model_node_name*] [`-fm` *memory_flags*] [`-fst` *integer*] [`-fl` *logger_flags*] [-fcons]

For information about these options, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

Before using the `forte` command, you must be sure that your FORTE_FTLAUNCH_PORT environment variable is set to a user port (your own individual port) rather than a system port. Please check with your system administrator for a port number that you should use. You should set the FORTE_FTLAUNCH_PORT environment variable on the command line or in your personal environment variable definition file because it is specific for an individual user.

➤ To start the iPlanet UDS Workshops on UNIX

1. Use the `source` command to set the environment variables stored in your `fortedef.csh` or `fortedef.sh` file.
2. Set your FORTE_FTLAUNCH_PORT environment variable to a user port number.
3. Enter the `forte` command.
4. The Repository Workshop opens.

Here are some examples of the `forte` command:

```
forte -fs
forte -fr bt:$FORTE_ROOT/repos/demo30
forte -fl "%stdout(trc:os:1:1 trc:err)" -fm "(n:4000,x:8000)"
```

Starting the Workshops on OpenVMS

On OpenVMS, you start the iPlanet UDS Workshops by using the `FORTE` command. By default, the `FORTE` command starts the workshops in distributed mode. To start the workshops in standalone mode, you can use the `/STANDALONE` qualifier.

The syntax for the FORTE command is:

```
VFORTE FORTE
        [/STANDALONE]
        [/REPOSITORY=repository_name]
        [/WORKSPACE=workspace_name]
        [/NODE=node_name]
        [/MODEL_NODE=model_node_name]
        [/MEMORY=memory_flags]
        [/STACK=integer]
        [/LOGGER=logger_flags]
        [/FCONS]
```

For information about these qualifiers, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

➤ **To start the iPlanet UDS Workshops on OpenVMS**

1. Set the iPlanet UDS logical names by entering the following command at the system command prompt:

   ```
   $ @SYS$LIBRARY:FORTE_LOGIN_Vversion_number.COM
   ```

2. Enter the `FORTE` command with the appropriate flags.
3. The Repository Workshop opens.

Ftcmd Command

The iPlanet UDS `ftcmd` command starts the specified application using the Launch Server. The syntax is:

```
ftcmd [-v] [-nolog] [-port port_number] [-fnd node_name] [-nonode] [-fs]
        [-fm memory_flags] [-fst integer] [-fl logger_flags]
        run application_name [release] [arguments] [update]
```

The following table describes flags that apply only to `ftcmd`. For information about the flags that apply to all commands used to start iPlanet UDS, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" on page 118.

Flag	Description
-v	Displays detailed information about the steps performed by the `Ftcmd` command. This flag works only on the current Ftcmd command. This flag is useful for diagnosing command syntax errors.
`-port` *port_number*	Identifies the port number that this `ftcmd` command will use to locate a Launch Server. If a Launch Server is not already running at this port number, a Launch Server is started at this port number.
`-i` *input_file.*	Run a single `ftcmd` command in the specified file.
-nonode	Tells the Launch Server not to check for a running Node Manager on the client node and not to act as a Node Manager. Normally, the Launch Server checks whether a Node Manager is running on the client node, and if there is none, then the Launch Server behaves like a Node Manager.
`run`	Specifies the application you wish to run.

The `run` flag of the `ftcmd` command specifies the application you wish to run using the following format:

Argument	Description
application_name	Specifies the name of the application that is being launched by the Launch Server.
release	Specifies the compatibility level to be started. By default, the Launch Server starts the release with the highest compatibility level.
arguments	Specifies command line arguments for the application being started.

Argument	Description
update	For installed client partitions only. Indicates whether the Launch Server automatically downloads and installs the most current application distribution, if the application has not yet been installed, or if the installed application is not as current as the distribution available in the environment. By default, the value is TRUE, which means that the more current application distribution is automatically downloaded and installed, if necessary. To prevent this automatic update, set this value to FALSE. This argument is ignored for publicly-available applications. Note that only the client partition of an application is downloaded, not any libraries that the client partition might depend upon.

If you specify just `ftcmd run` *application_name* without the *release* or *update* arguments, the Launch Server runs the application with the highest compatibility level and updates the application distribution if an application distribution that is more current than the installed one is available in the environment.

If you specify `ftcmd run` *application_name release arguments* `FALSE`, the application is started without checking whether the installed image repositories are as current as the image repositories in the environment.

Specifying Arguments

The command line arguments need to be interpreted by Fa string in quotation marks, for example "-fr CentralRepository -fw tempworkspace".

On some platforms, such as UNIX and Windows, you need to use escape characters, such as `\`, to have the quotation marks interpreted correctly. Therefore, to specify arguments, you might need to specify them using something like \"-country canada\", so that the quotation marks are not removed before the Ftcmd utility receives them. To see the actual command string that is being sent to the Ftcmd utility, specify the `-v` flag, as discussed in "Ftcmd Command" on page 115.

The following example demonstrates how to start an application on a Windows NT node called OLMBanking and specify an argument for that application:

```
ftcmd run olmbanking cl0 \"-country canada\"
```

For further information about the `ftcmd` command, see the *iPlanet UDS System Management Guide*.

Ftexec Command

The `ftexec` command starts the standard iPlanet UDS partition in the specified image repository. The syntax is:

```
ftexec -fi image_repository_name [-fs] [-fns name_server_address]
       [-fl logger_flags] [-fm memory_flags] [-fst integer]
       [-fcons] [-fnw] [-fterm] [-fss] [-fnomad]
```

For information about these flags, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands" below.

For further information about the `ftexec` command, see the *iPlanet UDS System Management Guide*.

Ftclntws Command

The `ftclntws` command starts the standard iPlanet UDS partition in the specified image repository. The syntax is:

```
ftclntws -fi image_repository_name [-fs] [-fns name_server_address]
       [-fl logger_flags] [-fm memory_flags] [-fcons] [-fnw] [-fterm]
       [-fss] [-fnomad]
```

For information about these flags, see "Flags for ftcmd, forte, ftexec, and ftclntws Commands."

For further information about the `ftclntws` command, see the *iPlanet UDS System Management Guide*.

Flags for ftcmd, forte, ftexec, and ftclntws Commands

The following table briefly defines the command line flags that can appear on the various commands used to start iPlanet UDS:

This Flag	Specifies
-fr *repository* `/REPOSITORY=`*repository_name*	The repository to be used for the development session. See below for information on specifying a repository.

This Flag	Specifies
-fw *workspace* /WORKSPACE=*workspace_name*	The workspace to be used for the development session. See below for information on specifying a workspace.
-fi *image_repository_name* /IMAGE_REPOSITORY= *image_repository_name*	Specifies the image repository which contains the partition to start executing. There is no default, and this must be specified. The full specification of the image repository name is: bt:FORTE_ROOT/userapp/*distribution_ID*/cl#/*partition_ID* *distribution _ID*: first 8 characters of the application distribution name #: compatibility level number *partition_ID:* first 6 characters in partition name plus partition's number
-fs /STANDALONE	(Clients only) Starts a client application as a standalone application.
-fns *name_server_address* /NAMESERVER= *name_server_address*	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. Generally, you should *not* use this flag when you are starting the iPlanet UDS Workshops.
-fnd *node_name* /NODE=*node_name*	Specifies the node name to use for this session. If you do not specify the node name in the command, the default node name depends on the operating system. On Windows, the default node name is set by the FORTE_NODENAME environment variable. On all other platforms, the actual node name is used. Generally, you should *not* use this flag when you are starting the iPlanet UDS Workshops.
-fm *memory_flags* /MEMORY=*memory_flags*	Specifies the space to use for the memory manager. See "-Fm Flag (Memory Manager)" on page 799 for details.
-fst *integer* /STACK=*integer*	Specifies the thread stack size in bytes for iPlanet UDS and POSIX threads. This specification overrides default stack size allocation. For more information, refer to the *iPlanet UDS System Management Guide*.

This Flag	Specifies
`-fl` *logger_flags*	Specifies the logger flags to use for the session. See "-Fl Flag (iPlanet UDS Logger)" on page 795 for details. If you do not set the logger flags in the command, iPlanet UDS uses the value of the FORTE_LOGGER_SETUP environment variable. Note that you can change the logger settings from the Repository Workshop.
-fcons `/FCONS`	(Clients only) On Windows platforms, this flag specifies that the background trace window also displays. By default, on these platforms, this trace window is not shown for iPlanet UDS client applications that do not use a command-line interface. On UNIX and OpenVMS platforms, this flag specifies that the client partition tries to run even if it cannot make a required connection to an X windowing system. Without this flag, such a partition would fail without the X windowing system connection.
-fnw `/FNW`	(UNIX and OpenVMS clients only) Specifies that the client session begins as a multithreaded partition without opening an X windowing system connection.
-fss	(Clients only) Displays the iPlanet UDS splash screen.
-fterm /FTERM	(UNIX clients only) Specifies that the client session run as always attached to the terminal, so that it always responds to terminal commands, such as Control-c.
-fnomad	(Clients only) Starts a client application without initially connecting to a name service or environment (a nomadic application).

Choosing a Repository

To specify the repository in a command:

Central repository Specify a repository service name. See your system manager for information about your central repository.

Private B-tree repository Specify the repository name using the following format: `bt:` *private_repository_name*.

Shadow repository Specify the repository name using the following format: bt: *shadow_repository_name*.

See "About Repositories" on page 139 for information about the different kinds of repositories.

If you do not specify a repository in the command, the default repository depends on the operating system. For all platforms, iPlanet UDS uses the setting of the FORTE_REPOSNAME environment variable, and if the environment variable is not set, iPlanet UDS uses the distributed repository called "CentralRepository."

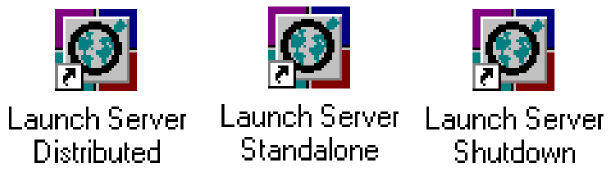
Choosing a Workspace

If you do not specify a workspace in the command, the default workspace depends on the operating system. For all platforms, iPlanet UDS uses the setting of the FORTE_WORKSPACE environment variable, and if the environment variable is not set, iPlanet UDS opens the Repository Workshop without a workspace. For Windows, iPlanet UDS uses the last workspace you opened in the Repository Workshop. If the value of FORTE_WORKSPACE is not set, the default workspace is FirstWorkspace.

Running the iPlanet UDS Launch Server

The iPlanet UDS Installer installs the following icons for starting and stopping the Launch Server.

Figure 2-3 iPlanet UDS Launch Server Icons



If you are starting iPlanet UDS in standalone mode, you can start the Launch Server in either standalone or distributed mode. If you are starting iPlanet UDS in distributed mode, you must start the Launch Server in distributed mode.

➤ **To start the Launch Server**

1. Double-click the Launch Server Standalone or Launch Server Distributed icon.
2. Leave the Launch Server running as long as you are using iPlanet UDS.
3. After ending your iPlanet UDS session, double-click the Launch Server Shutdown icon.

Using the iPlanet UDS Launcher Application

On all platforms except OpenVMS, you can start any of the iPlanet UDS utilities, including the iPlanet UDS Workshops, by using the iPlanet UDS Launcher application.

The following table describes how to start the Launcher application on each platform:

Platform	How to Start the Launcher Application
Windows 95 or Windows NT	Double-click the Launcher Distributed or Launcher Standalone icon.
UNIX	Run the launcher script in `FORTE_ROOT/install/bin/LAUNCHER`.

Using Distributed Mode

If you wish to run the iPlanet UDS Workshops (or any other applications) in distributed mode, you must start the Launcher Application in distributed mode. Otherwise, you can run the Launcher Application in standalone mode.

The Launcher Application displays the following tab pages, which let you start and shutdown applications using the Launch Server:

User Applications available to the end user. You can start applications from this window.

iPlanet UDS iPlanet UDS product applications, such as the Environment Console, the iPlanet UDS Workshops, and so forth, which can be started by the Launch Server. You can start some of the iPlanet UDS product applications from this window.

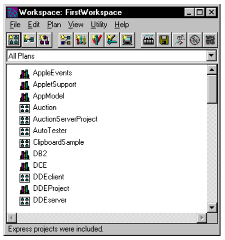
Running Applications that have been started by the Launch Server. You can shutdown applications from this window.

Options Shows the options that the user can set when running the Launcher application.

The online help for the Launcher application (available through the Help button) provides specific information about using the application to start and stop applications.

Using iPlanet UDS Windows

An iPlanet UDS workshop is a graphically-oriented application, itself developed in iPlanet UDS.



A workshop appears in a window with a menu bar. The window may include a variety of buttons and icons on a toolbar for performing specific workshop tasks. In any workshop, you can perform tasks either by using the controls on the toolbar or by choosing commands from the workshop's menu bar. This way, you can reduce the size of a workshop window by closing the toolbar, but still have access to the same functions through the menu bar.

The windows in the iPlanet UDS workshops behave as any application windows in your host window system, but provide a few iPlanet UDS-specific enhancements.

Using the Mouse

The iPlanet UDS workshops behave like any standard applications in your window system—mouse clicks select objects, double-clicks open objects, and click-and-drag operations move or copy objects.

To work in iPlanet UDS you need only one mouse button. If your mouse has more than one button, you use the leftmost button, or, if your mouse is configured specifically for left-handed use, the rightmost button.

To activate some workshop commands, you use mouse clicks in conjunction with qualifier keys. For example, in the Window Workshop, to select multiple widgets, your first click on one widget, then, while depressing the Shift key, click on successive widgets to add to or delete from the selection.

Using the Keyboard

The iPlanet UDS workshops are consistent with their host window systems in the flexibility of keyboard control they provide.

In the Windows and Motif window systems, you can use the keyboard exclusively to complete any iPlanet UDS workshop task (even those designed for mouse interaction) because these window systems have been designed to allow mouse-independent operation.

The iPlanet UDS workshops offer keyboard equivalents for many menu commands on every window system. When available, the keyboard equivalent is displayed on the menu next to the command.

Using Specialized iPlanet UDS Widgets

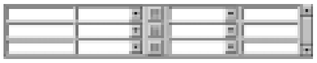
The iPlanet UDS workshops use many of the specialized widgets provided by the iPlanet UDS Display library. These widgets include the following compound fields:

Compound Widget	**Definition**
Panel	A subform that contains any number of widgets.
Compound Graphic	A group of graphic widgets only.
Viewport	A scrollable field displaying a larger widget, such as a picture graphic.
Grid Field	A table to organize child widgets into rows and columns for uniform display.
Array Field	An array of widgets in a tabular format, like a spread dialog, where component widgets are the same from row to row, but their data varies.

When you are working with some of these compound fields (panel, grid field, and array field), you may need to edit the data they contain. To move from one field to the next within the compound field, you can either use the mouse or the Control-Tab key combination.

The following sections provide further information about two specialized widgets, array fields and outline fields.

Array Fields

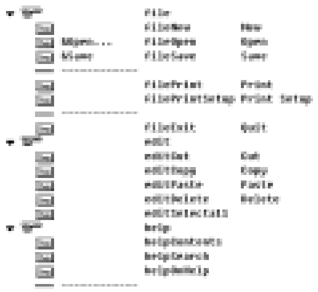


iPlanet UDS workshops use array fields to display rows and columns of data in a scrollable field display that looks something like a spread sheet. Each row in an array field represents a row of data from an underlying array. When you click the scroll bar to scroll through the data, you bring new rows from the data array into view. Frequently, you can edit the data in an array field, add new rows to it, or delete rows from it.

The figure at left shows an array field from an iPlanet UDS window. Here, as in other cases where the workshops use array fields, you use buttons to insert or delete rows from the array field. When you click the Insert button, the workshop adds a row to the array field at your cursor's position. When you click the Delete button, the workshop deletes the active row.

For instructions on how to use array fields in your own applications, see "Creating an Array Field" on page 514 and the Display Library online Help.

Outline Fields



Outline fields are the browsers that the iPlanet UDS workshops use to display hierarchical data. One example of a browser is in the Menu Workshop, shown at left, which provides a hierarchical display of menu components.

You may be familiar with browsers as the file directory display mechanisms in your host window system. iPlanet UDS outline fields behave most like the file directory application in Microsoft Windows, representing containers and the components they contain with icons. These browsers display hierarchical information, successively indented from left to right. To select an item from a browser, you click on it. To open a line in an outline field, you double-click it, press Return while it is selected, or click its expansion arrow.

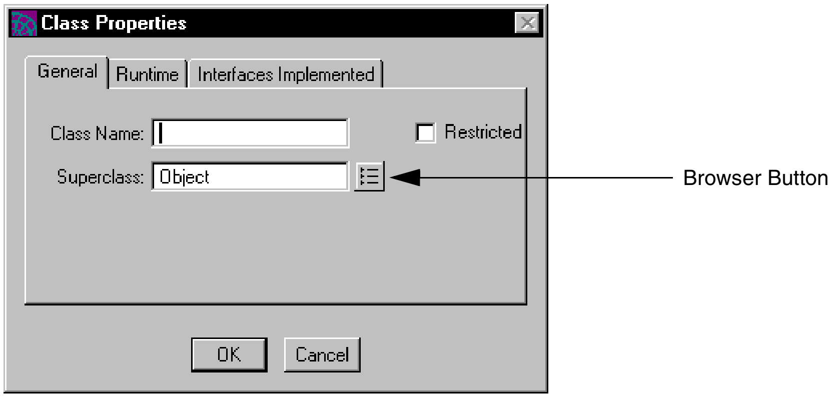
The expansion arrow lets you open or close the list of items "below" the current item in the hierarchy. If the expansion arrow is pointing to the right, click the arrow to open up the list. If the expansion arrow is pointing down, click the arrow to close up the list.

For instructions on how to use outline fields in your own applications, see "Creating an Outline Field" on page 472 and the Display Library online Help.

Browser Buttons

Many iPlanet UDS dialogs contain a browser button (shown at left), which allows you to open a class browser or class elements browser. To use the browser, click on the browser button, which opens an outline field. You can then browse through the information until you find the item you wish to select. The value you select from the browser is then used to specify the value for a field on the dialog. Figure 2-4 illustrates a dialog with a browser button:

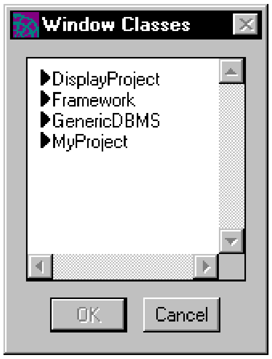
Figure 2-4 Browser Button on Class Properties Dialog



Class Name:

Superclass: Object

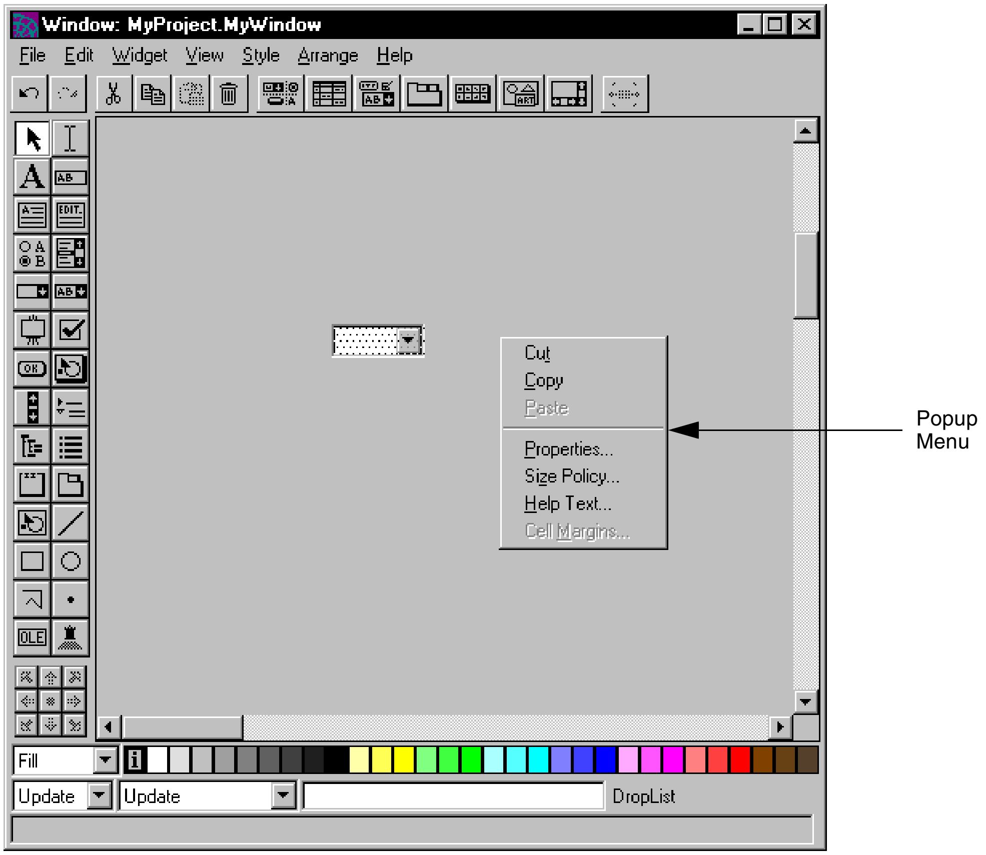
Figure 2-5 Browser Button Display



Using Popup Menus

A popup menu is an independent menu associated with a specific widget on the window. The popup menu is displayed next to the widget when you use the appropriate key combination. Figure 2-6 illustrates a popup menu:

Figure 2-6 Popup Menu in Window Workshop



Activating a Popup Menu

The key combination needed to activate a popup menu depends on the particular window system. To activate a popup menu:

Window System	Key Combination to Activate Popup Menu
Windows	Use mouse outer button, or select the field and use Shift F10. The outer mouse button is the right-most button on a right-handed mouse and the left-most button on a left-handed mouse.
Motif	Use mouse outer button, or select the field and use Shift F10. The outer mouse button is the right-most button on a right-handed mouse and the left-most button on a left-handed mouse.

iPlanet UDS uses the position of the cursor at the time the popup key combination is entered to determine which popup menu to display. If the widget over which the cursor is positioned has a popup menu assigned to it, iPlanet UDS displays the current widget's popup menu. If the current widget does not have a popup menu assigned to, iPlanet UDS displays its parent's popup menu. If the parent widget does not have a popup menu assigned to it, iPlanet UDS displays the popup menu for the parent's parent, and so on. If the top-level container for the widget, the Window object, does not have a popup menu, nothing happens.

Choosing a Menu Item from a Popup

Once the popup menu is displayed, you can choose a menu item from it the same way you chooses a menu item from a menu on the menu bar.

You can close the popup menu any time with the appropriate action. To dismiss the popup menu:

Window System	Key Combination to Dismiss Popup Menu
Windows	Click outside the menu or press Esc key.
Motif	Click outside the menu or press Esc key.

Importing and Exporting Data

You can import and export data in iPlanet UDS through the window system clipboard, and through file saving and file loading mechanisms that the workshops provide.

Using the Clipboard

You can use your host window system clipboard to cut, copy, and paste textual and image information within the iPlanet UDS workshops, and to exchange such information between the iPlanet UDS workshops and other applications in your host window system.

Using the Motif Clipboard

Under the Motif window system, the iPlanet UDS workshops offer limited support for the Motif clipboard. iPlanet UDS workshops use the clipboard only to exchange iPlanet UDS-specific data among themselves (under certain conditions).

In Motif, iPlanet UDS workshops use a clipboard and can exchange data one to another, only when they are running under the same instance of an iPlanet UDS executable. You can, for example, copy and paste data from one instance of the Method Workshop to another, so long as both instances run under the same iPlanet UDS executable. If, however, you copy data to the clipboard from an instance of the Method Workshop and then quit the workshop, the clipboard does not retain the data. iPlanet UDS data pasted or cut to the Motif clipboard is not visible to other, non-iPlanet UDS applications.

Under any circumstance, you can paste text from any Motif application into iPlanet UDS by using the technique you use to select and paste text from one xterm to another: select the text, and press the middle mouse button in the target window. This technique works because it does not employ the Motif clipboard.

Using Multiple Windows

After opening the Repository Workshop, you can open any number of Project Workshops for each project you wish to view or change. iPlanet UDS lets you work with any number of concurrent windows. If your screen ever gets too cluttered or the window you wish to use is not completely visible, you can use the File > Workshops Open… command in any workshop. This command displays a list of all the open workshops, and lets you choose a window to bring from underneath to the front.

Printing Windows

The File menu in every iPlanet UDS workshop has a Print command, which allows you to print the information in the workshop. The Print command prints all information in the browser, source code field, or form (in the case of the Window Workshop). All information contained by the field is included, not just the information currently being displayed. For example, the Print command prints the entire method currently being displayed in the Method Workshop. Multiple pages are used if necessary, with titles and page numbers. For the Window Workshop, the Print command tiles the form onto multiple pages if necessary.

The File > Print Setup or Page Setup command opens the Window System’s Print Setup dialog, where you can specify your setup options.

Using iPlanet UDS Names

All of the components stored in the development repository have names, including:

- projects
- libraries
- named constants
- classes
- interfaces
- service objects
- methods
- attributes
- events
- event handlers
- cursors
- application models
- business models
- process definitions
- application dictionaries
- user profiles
- user validations
- assignment rules

Rules for Naming Components

An iPlanet UDS name can be any series of alphanumeric characters and underscores, with the first character an alphabetic character or an underscore. Case is not significant. No spaces and symbols (except the underscore) are allowed. The name can be any length.

When you are naming a new component, the name must be unique for the current project, class, or method, and must not be a reserved word (see the *TOOL Reference Guide* for information on name resolution and reserved words).

In addition, due to the iPlanet UDS own naming scheme, there are two restrictions on words that you can include in names:

- You cannot start any name with the word "FORTE".
- You cannot end any name with the word "Proxy."

Using iPlanet UDS Data Types

In the iPlanet UDS Workshops, you often set the data type of components you create. You sometimes have a choice of several simple and class data types.

Simple Data Types

The simple data types store boolean, numeric, or character data. All the simple data types have counterparts defined as iPlanet UDS classes, which store data in objects. Although the simple data types are more efficient than their corresponding classes, the classes provide a wide range of methods for manipulating the data. The simple data types are shown in the following table:

Simple Data Type	Definition
boolean	Stores two logical values, TRUE and FALSE.
integer	Stores a positive or negative whole number of two to four bytes.
long	Stores a positive or negative number of at least four bytes.
float	Stores a smaller floating point number (exact size is machine dependent).
double	Stores a larger floating point number (exact size is machine dependent)
string	Stores character data.

Data Type Classes

The iPlanet UDS data type classes define objects for storing boolean, numeric, or character data. Using objects to store this kind of data lets you take advantage of the methods provided for manipulating the data. In addition, the `DateTimeData`, `DecimalData`, `IntervalData`, and `ImageData` classes provide the only way to store date, time, interval, and image data.

The following table lists the data type classes, and shows how they correspond to simple data types:

Class Data Type	**Simple Data Types**	**Class Data Type Definition**
BooleanData	boolean	Stores logical values of TRUE and FALSE.
DateTimeData	none	iPlanet UDS offers no simple type for date data; DateTimeData and IntervalData store this data.
DecimalData	float, double	DecimalData represents floating point data to a specific precision of as many as thirty decimal places.
DoubleData	double	DoubleData stores floating point data.
ImageData	none	ImageData stores images in a standard, portable format. It also reads several popular formats.
IntegerData	integer, long	IntegerData accepts whole numbers of any size.
IntervalData	none	IntervalData defines intervals of time.
TextData	string	TextData stores character data.
BinaryData	pointer	BinaryData provides access to database BLOB types.

For more information about using simple data types, see the *TOOL Reference Guide*. For more information about using the class data types, see the Framework Library online Help.

NULL Class Types

For each of the class data types, there is also a corresponding NULL class data type variation. The NULL data type classes are provided to give iPlanet UDS applications seamless access to relational databases, which often contain NULL data. To use database tables containing NULL data, you must use the following NULL class data types:

Class Data Type	NULL Class Data Type
BooleanData	BooleanNullable
DateTimeData	DateTimeNullable
DecimalData	DecimalNullable
DoubleData	DoubleNullable
ImageData	ImageNullable
IntegerData	IntegerNullable
IntervalData	IntervalNullable
TextData	TextNullable
BinaryData	BinaryNullable

For more information about using the NULL class data types for database access, see *Accessing Databases*.

Using iPlanet UDS Online Help

Online help for iPlanet UDS is available from any iPlanet UDS window. iPlanet UDS Help follows Windows Help standards, so you can refer to the Windows documentation for background on using Help.

"Help on Help"—that is, online Help on how to use the Help system—is available by pressing the F1 key when a Help window is active.

There are two kinds of Help:

- a general Help function, available through the Help menu
- context-sensitive Help, available through pressing the F1 key

Context-sensitive Help is not available for all windows and dialogs. When context-sensitive help is not available, pressing the F1 key displays the main Help contents page.

You close Help windows in the same way you close other windows—using the mechanism provided by your particular window system.

Jumps and Pop-ups

The main feature of hypertext help is the ability to jump to a topic or get an instant pop-up definition of a term.

- On Windows, *jumps* are highlighted and underlined in green (other window systems may use different indicators for jumps). If you click on a jump, you will open a help window on that topic.
- On Windows, *pop-ups* are highlighted in green but with a dotted underline (other window systems may use different indicators for pop-ups). If you click on a pop-up, you will get an instant definition of the highlighted term.

Searching

A useful feature of the Help system is the ability to search for key words. You begin searches by choosing the Help > Search command. Once you find the key words or phrases you seek, choose Show Topics to see a list of topics associated with the search term. Open a topic window by selecting the topic and choosing Go To.

Other Features

Other features of iPlanet UDS Help include the following:

- choose Contents to return to the main Help contents page
- choose History to review and, if desired, go to a recently selected Help topic
- choose Back to return to the previous Help topic
- when available, use the Browse buttons (double-angle brackets) to browse through the topics

Other Help features allow you to copy selected text to the clipboard or to print a Help topic or file. See your Windows manual or “Help-on-Help” for more information.

Setting Workshop Preferences

All the iPlanet UDS workshops, except the Debugger and the Partition Workshop, have a Workshop Preferences... command, which you can use to set preferences that are saved as part of your current workspace. The Workshop Preferences... command opens the Workshop Preferences dialog for the current workshop, where you can set preferences that are saved as part of your current workspace.

This section describes the preferences common to all or most workshops. For information on preferences that apply only to a specific workshop, see the chapter on that workshop.

The general preferences that apply to all or most workshops are:

- workshop window size and positions
- filter
- viewing preferences
- font preference

Workshop Size and Position

There are three different ways you can set the initial size and position of a workshop window.

Visual positioning before opening The simplest way to set the size and position of the window is to do so visually prior to opening the Workshop. Set the window size by using the window's resize handles. Then set the window position by dragging the window into position. When the window has the size and position you want, click the Reset to This Workshop button in the Repository Workshop Preferences dialog. This button sets the value of the Position preference to Screen Relative, and the values of the X, Y coordinates and the Columns and Lines settings to the current values of workshop on the screen.

Note that the Reset to This Workshop button also changes your Viewing preferences to the current settings on the View menu and the font preferences to the current setting in the workshop.

Columns and lines The second way you can control the size of the workshop is to set the size of the browser section in terms of lines and columns as follows:

Preference	How to specify its value
Columns	Enter an integer that specifies the width of the browser by the number of characters displayed (for proportional fonts, iPlanet UDS uses the size of a digit as the equivalent to one character).
Lines	Enter an integer that specifies the length of the browser by the number of lines displayed (the size of a line is determined by the font size).

Position and X, Y values Finally, you can set the initial position of the workshop window by setting its position properties the same way you set the initial position of a window you create using the Window Workshop. The Preferences dialog provides the same options as the Window Properties dialog for setting the window's initial position. On the Preferences dialog, the Position and X,Y options are the same as the Initial Position Policy and Initial X and Initial Y on the Window Properties dialog. See "Initial Position Property" on page 367 for a complete description of setting the initial position for a window.

Filters

Workshops that have browsers provide a filter drop list that allows you to choose the item or combination of items you wish to display. You can use the same drop list in the Workshop Preferences dialog to choose the default filter used by the workshop.

Viewing Preferences

The viewing preferences let you specify the default settings for the menu items on the workshop's View menu of the workshop. The viewing preferences correspond one-to-one with the commands on the View menu.

Font Preference

In the Workshop Preferences dialog, you can set the font used by the workshop.

➤ **To set the font for the workshop**

1. On the Workshop Preferences dialog, click the Font button.
2. In the Font dialog, choose the typeface, size, and style. The sample illustrates how your font selection will look.
3. Click the OK button to choose the new font.

Leaving iPlanet UDS

You can leave the iPlanet UDS development environment by choosing the File > Exit command in the Repository Workshop. If you have not saved your workspace before giving the command, you will be prompted to do so.

About International Support

iPlanet UDS uses the language conventions that were set on the current platform. For Windows platforms, you use the appropriate Control Panels to specify your territory conventions. For UNIX, use the LANG environment variable. For OpenVMS, use the FORTE_LOCALE logical name (see the *iPlanet UDS System Management Guide* for a complete list of the iPlanet UDS environment variables).

For information about using internationalization features in your iPlanet UDS applications, see the *iPlanet UDS Programming Guide*.

Chapter 3

Using the Repository Workshop

This chapter provides conceptual information about iPlanet UDS repositories and workspaces, and describes how to use the Repository Workshop.

In this chapter, you will learn how to:

- enter and leave the Repository Workshop
- use iPlanet UDS repositories
- create and use iPlanet UDS workspaces
- test TOOL projects
- set repository passwords
- use Repository Workshop utilities
- set Repository Workshop preferences

About Repositories

An iPlanet UDS *repository* is a database that stores the plans that you create with iPlanet UDS, along with the iPlanet UDS libraries and user libraries.

In iPlanet UDS, a *plan* is one of the following:

Icon	Plan	
	Project	A definition of an iPlanet UDS service, library, or client application.
	Library	A named collection of classes and other component definitions that can be shared by any number of iPlanet UDS applications.
	Business model	A graphical representation of the data in an iPlanet UDS Express application. Created by iPlanet UDS Express.
	Application model	A graphical representation of the windows in an iPlanet UDS Express application. Created by iPlanet UDS Express.
	Process definition	An iPlanet Integration Server (iIS) workflow representation of a business process, consisting of a set of graphical objects (activities and timers) that are linked together and have properties defined for them.
	Application dictionary	A group of application dictionary items, each of which indicates to an iIS client application the work to be done for an activity in an iIS process definition, and optionally provides access to data needed for that work. Created by iPlanet Integration Server.
	User profile	A template specifying all potential characteristics of a user in an organization, such as the user's role in the organization. Created by iPlanet Integration Server.
	User validation	A class containing a ValidateUser method that is used by the iIS process engine to verify each user's profile against information stored in a site's organizational database. Created by iPlanet Integration Server.
	Assignment rule	A rule that specifies who can perform various activities in iIS process definitions. Created by iPlanet Integration Server.

The plans in your repository include projects that you create with iPlanet UDS to define your applications, servers, and libraries. Your repository may also contain plans created with iPlanet UDS Express and iPlanet Integration Server. In this chapter, the term *plan* is used to cover both the plans created with iPlanet UDS as well as with the add-on products. For example, when we discuss how to import a plan, we are referring to the plans created with iPlanet UDS Express and iPlanet Integration Server as well as those created with iPlanet UDS. However, for specific details on working with iPlanet UDS Express and iIS plans, please refer to the documentation on those products.

Note that if your repository contains plans created by iPlanet UDS Express or iPlanet Integration Server, even if you have not installed the products that were used to create the plans, you will be able to see the plans in your repository. However, if you do not have the appropriate product installed, you will not be able to open the plans.

You will always have iPlanet UDS libraries in your repository. You may also have user libraries that you have imported into your repository.

The iPlanet UDS *libraries* contain the prefabricated classes that provide the foundation for building applications. These include basic application components, such as windows, menus, and fields, framework classes for structuring your application, database classes for accessing your DBMS, and classes for integrating with external systems.

User libraries contain specialized classes that you can use in developing your applications. These libraries may be developed within your own organization, or may be purchased from third-party vendors.

Type of Repository

There are two kinds of repositories in iPlanet UDS: central repositories and private repositories.

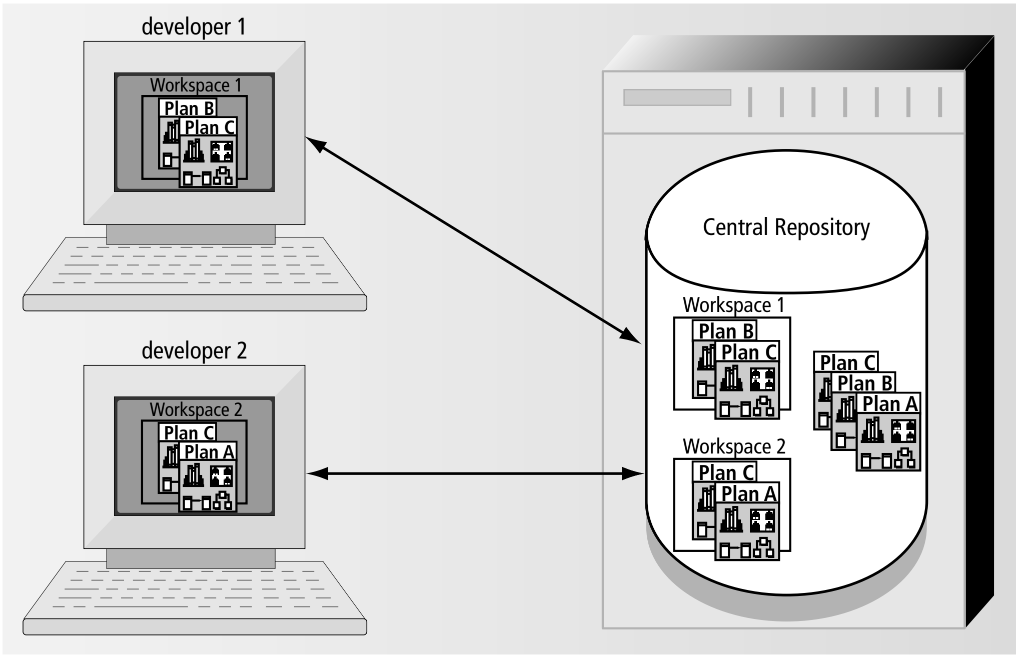
A *central repository* is a shared repository that can be used concurrently by multiple developers. The central repository contains all the iPlanet UDS libraries and all the plans integrated into it by a group of developers in your organization. Most of the time, you will be doing your work with a central repository. See "About Central Repositories" on page 143 for details about central repositories.

A *private repository* is an independent repository that can be used by a single user. A private repository runs stand-alone and does not have to be connected to the iPlanet UDS distributed development environment. Because iPlanet UDS allows you to create a local shadow of the central repository, private repositories are not needed for performance reasons. However, private repositories allow you to work completely independently of other developers. See "About Private Repositories" on page 150 for details about private repositories.

Workspaces

When you work with a repository, you always use a workspace.The workspace contains a subset of the plans in the repository, and only those plans included in the workspace are visible to the developers using that workspace. Figure 3-1 illustrates the relationship between a workspace and a central repository:

Figure 3-1 Workspaces



The workspace ensures that you can work independently of other developers who are collaborating with you on the same projects. Any changes you make in a workspace are not visible to other developers until you integrate them into the repository by using the Integrate Workspace command. And integrated changes made by other developers are not visible to you until you give an Update Workspace command to bring the workspace up to date. See "About Workspaces" on page 151 for information about creating and using workspaces.

Before you run the iPlanet UDS Workshops, you must decide which repository to use. After you enter the Repository Workshop, you can select the workspace you wish to use. If necessary, you can create a new workspace or modify existing workspaces by adding plans to or removing plans from them.

About Central Repositories

A central repository is a shared repository that can be used concurrently by multiple developers. The central repository contains all the iPlanet UDS libraries and all the plans integrated into it by the developers in your organization. You can have any number of central repositories per environment.

A central repository provides centralized access to plans in order to allow collaboration between multiple developers. Normally, all plans being shared by a department or development team are stored in a single repository.

All the plans needed by an application, that is, the main project and all its supplier plans, must be in the same central repository. (See "Supplier Plans" on page 209 for information about supplier projects.) Therefore, although you can develop a single project in a separate repository, if your project defines part of a larger application, when your project is ready, you must import it into the shared central repository where the main application is being defined. When the final application distribution is created, all the projects needed by the application must be in the repository from which the distribution is being made.

Central repositories are always on server nodes in the environment. See the *iPlanet UDS System Management Guide* for information about creating and maintaining central repositories.

Every central repository has a prefabricated workspace called FirstWorkspace. Normally you access a central repository through your own workspace, which contains a subset of the plans in the central repository. (See "About Workspaces" on page 151 for information on workspaces.)

Shadow Repositories

Because a central repository is located on a central server in the development environment, iPlanet UDS provides a special feature called a *shadow repository* that allows you to increase performance by creating a local copy of part of the repository on your workstation, while still maintaining the connection to the central repository.

When you use a shadow repository, only the data you need is cached locally. When you first create a shadow, it is essentially empty. Because your shadow is connected to the central repository (unless you detach it as described under "Detached Shadow Repository" on page 144), repository components are copied to the shadow on an as-needed basis. As you begin to work in the attached shadow, creating new plans or new components within a project, modifying existing plans, or even just browsing through the components in a project, the components you need are copied to the shadow.

Note that you do not need to use a shadow. You can work directly with the central repository. The main reason for using a shadow is for increased performance. Another reason is if you wish to temporarily do your work outside the distributed development environment by detaching the shadow (described under "Detached Shadow Repository" below).

A shadow repository is either attached to a central repository or detached from it.

Attached Shadow Repository

An *attached shadow* repository is connected directly to the central repository, and every change that you make to the local shadow is made to the central repository every time you save your workspace (unless your preferences are set otherwise). Because this is the most efficient way to work, most of the time you will do your development work in an attached shadow.

When working with an attached shadow, you have the option of saving changes in the local shadow without immediately saving to the central repository. When you set your preferences so that changes are not automatically committed to the central repository, changes you save to your workspace are not written to the central repository until you give an explicit Commit to Central command or exit from the iPlanet UDS Workshops. Although this is not the most reliable way to work, it provides improved performance for the repository.

At any time, you can give a Detach Shadow command to detach the shadow from the repository.

Detached Shadow Repository

A *detached shadow* repository is disconnected from the central repository. Changes that you make in the detached shadow are not made to the central repository until you give an Attach Shadow command to connect it back to the central repository. Because the detached shadow is disconnected from the central repository, you cannot update or integrate your workspace, and you cannot include a public plan in your workspace or remove a public plan from the central repository. Most importantly, you cannot checkout project components for modification in a detached shadow. You can modify a component in a detached shadow only if:

- You checked out the component before you detached the shadow.
- You branched the component either before or after you detached the shadow.

See "Checkout and Branching" on page 146 for information about checking out and branching project components.

A detached shadow is useful for taking your work home with you or for working outside the distributed development environment. When you have finished working outside the distributed environment, you can bring the detached shadow back into work, and give the Attach Shadow command to attach it back to the central repository. This command copies all your changes from the shadow to the central repository.

A detached shadow has only one workspace, the workspace that was open when you gave the Detach Shadow command. The workspace in a detached shadow is considered "reserved," and no other developers will be able to open it as long as you are detached.

Using the Central Repository

Central repositories are created by your system manager as part of the process of setting up your development environment. For information, see *iPlanet UDS System Management Guide*.

To use a central repository, you create a private workspace that lets you work independently of other developers. A private workspace ensures that you can make changes without conflicting with developers who are working on the same projects. Any changes you make in your workspace are not visible to other developers until you integrate them by using the Integrate Workspace command. And integrated changes made by other developers are not visible to you until you give an Update Workspace command to bring your workspace up to date. For information about creating and using workspaces, see "About Workspaces" on page 151.

In the iPlanet UDS documentation, we use the term *system baseline* to refer to the latest version of each project component that was integrated into the repository. When you give an Update Workspace command, you are synchronizing your workspace with the system baseline. And when you give an Integrate Workspace command, you are integrating the changes in your workspace into the system baseline.

Checkout and Branching

To prevent conflicts that would be caused when different workspaces modified the same project or project components, iPlanet UDS does not allow more than one workspace to modify a project component at the same time. To modify an existing project component, the workspace must either check it out of the repository or branch it.

NOTE For iPlanet UDS Express and iPlanet Integration Server plans, iPlanet UDS does not allow you to check out or branch individual components within the plan. Instead, you must check out or branch the entire plan. For information on how to check out or branch an Express or iIS plan, see the documentation on that product.

Checkout

When you wish to make permanent changes to a project component, you check it out into the workspace. Checking out a component means that your workspace has a write lock on the component. As long as you have the component checked out, no one else can check it out. You have the "official" copy of the component, and only you can integrate the changes back into the system baseline.

The changes are not visible to other developers until you check the component back in by using the Integrate Workspace command to integrate the changes in your workspace into the system baseline. While you have the component checked out, other developers can read it, but no one else can modify it (unless they branch it).

Branching

If you need to modify a component that has already been checked out by another workspace, or if you simply wish to try out some changes without locking the component, you can branch the component in the workspace.

Branching a project component gives you an alternate path to it so you can test out a change. You can also branch a component while you are working in a detached shadow if you need to make changes to a component that you did not checkout before you detached the shadow. Any number of workspaces can branch the same component.

The changes you make to a branched component are always temporary because they cannot be integrated into the system baseline. To integrate the changes into the baseline, you must either:

- Convert the branch to a check out (if possible), and then integrate the modified component into the system baseline.
- Export the changes to a file before integrating, and then recover the changes after integration.

Updating and Integrating

After you have made changes to a project or have created a new project, you can use the Update Workspace and Integrate Workspace commands to add your changes to the repository and free your project components to be checked out again.

Before you integrate, you update your workspace and test the project in the updated workspace to make sure it is compatible with the current state of the system baseline. Updating merges the changes integrated into the system baseline with the changes in your workspace, so you can test your own work against the system baseline. See "Using Workspaces" on page 168 for information about this process.

After you have tested the project in the updated workspace, you can use the Integrate Workspace command to integrate your changes into the system baseline. Integrating makes your changes available to other workspaces. After you integrate, any other developers that use the Update Workspace command will have your changes added to their workspaces. If you have created a new project (or any other plan), using the Integrate Workspace command will make the new project available to other developers to include in their workspaces.

About Repository Security

In terms of security, iPlanet UDS supports two kinds of repositories: standard repositories and secure repositories.

Standard Repository

A standard repository, by default, requires no passwords. However, to prevent unauthorized users from changing information in the repository, you can set a master password, passwords for each workspace, and a baseline password. Without a password, a user can still examine the information in the repository, however, he or she is prevented from making unauthorized *changes* to the repository.

Secure Repository

A secure repository requires passwords for reading information from the repository, changing information in the repository, using workspaces, creating new workspaces, and copying the repository. A secure repository prevents unauthorized users from *viewing* information in the repository as well as from changing it. Your system manager must create the secure repository.

File Permissions

On some platforms, your system manager can also change the file permissions to restrict access to the repository by various users. By making the file permissions more restrictive, your system manager can prevent unauthorized users from accessing the repository files directly to read the source code using various utilities.

Security for Standard Repositories

By default, standard iPlanet UDS development repositories do not require passwords. However, you can set the following passwords to provide a level of security for the repository:

master password provides global access to all password-protected functions. You can set the master password using the `rpstart -p` flag or the Fscript `SetPassword` command.

baseline password prevents unauthorized users from integrating a workspace into the system baseline in the repository. You can set the baseline password in the Repository Workshop using the File > Set Baseline Password command or in Fscript using the `SetPassword` command.

workspace passwords restricts access to an existing workspace. You can set workspace passwords when you first create new workspaces. You can also set workspace passwords in the Repository Workshop using the File > Set Workspace Password command or in Fscript using the `SetPassword` command.

See "Setting Repository Passwords" on page 193 for information on setting these passwords.

Security Limitations of Standard Repositories

Even if you set the master, workspace, and baseline passwords, you cannot prevent unauthorized users from reading and copying source code from the repositories in one of the following ways:

- copying the repository using **rpcopy**

  Users can create their own copies of repository files by using **rpcopy** on a running repository server, even if file permissions prevent them from accessing the original repository files.

- creating a new workspace

  Users can then read any public plans in the repository

- directly parsing the contents of the repository files

If you are concerned about this level of security, consider using secure repositories, as described next.

Security for Secure Repositories

A secure repository requires passwords for reading information from the repository, changing information in the repository, using workspaces, creating new workspaces, and copying the repository.

Creating a Secure Repository

To create a secure repository, your system manager must use the `rpcreate` command with the `-secure` flag. After invoking the `rpcreate` command with the `-secure` flag, your system manager must set all of the following passwords:

administrator password Prevents unauthorized users from copying the repository using **rpcopy** or creating new workspaces.

master password Provides global access to all password-protected functions.

baseline password Prevents unauthorized users from integrating the workspace into the system baseline in the repository.

workspace password Restricts access to the FirstWorkspace workspace.

See *iPlanet UDS System Management Guide* for details about creating secure repositories.

Changing Repository Passwords

You can change passwords in a secure repository using the Repository Workshop or Fscript. See "Setting Repository Passwords" on page 193 for information about using the Repository Workshop to change repository passwords. See *Fscript Reference Guide* for information about using Fscript to change repository passwords.

About Private Repositories

A private repository is an independent repository that can be used by a single user. A private repository runs stand-alone and does not have to be connected to the iPlanet UDS development environment, although it can be.

Private repositories are designed for independent, high-performance development inside or outside an iPlanet UDS distributed environment. A private repository provides somewhat better performance than an attached shadow; however, it does not provide the collaboration facilities and centralized access provided by a central repository.

You can use a private repository to build an entire distributed application on your own. You can also use it to build part of an application, and then export component and plan definitions into a central repository.

Creating a Private Repository

The *iPlanet UDS System Management Guide* provides detailed information about how to create a private repository. You can do so by:

- using the `rpcopy` or `rpcreate` utilities
- on a client, copying the `btseed.btd` and `btseed.btx` files in your iPlanet UDS installation

After you have created the private repository, you can create any number of new plans in it.

Using a Private Repository

If you are using a private repository, you normally use the prefabricated workspace called FirstWorkspace. Because no one is sharing the repository with you, it is not necessary to create personal workspaces for private repositories. It is also unnecessary to checkout components in order to modify them.

If you are creating the entire application from your private repository, you can make the application distribution files directly from the private repository. However, if you are working on part of an application that you need to integrate with other developers' work, you must import the work you did in the private repository into a central repository.

Use the Export command in the Repository Workshop to export the plan to a file, or use the Export command in the Project Workshop to export an individual class to a file. Then, working from your central repository, use the Import command in the Repository Workshop to import the entire plan or use the Import command in the Project Workshop to import an individual class into your central repository. The Export and Import commands for plans are described under "Importing and Exporting Plans and Libraries" on page 183. The Export and Import commands for individual classes are described under "Importing and Exporting Classes and Interfaces" on page 270.

About Workspaces

You always use a workspace when you are working in iPlanet UDS. A workspace contains your view of the repository, along with any changes that you make to it. It normally contains a subset of the plans in the repository, which makes it more efficient for updating, integrating, and other operations such as detaching shadows.

Using Workspaces for Collaboration

A workspace allows you to work independently of other developers. For example, when developers collaborate together on a project, each developer has her own workspace, where she can develop and test her part of the project. The same project can be included in any number of workspaces, so that any number of developers can collaborate on it. Quite often, developers use their own name for the workspace name, which enables their coworkers to identify who is using which workspace.

Source Code Control

Only one developer can open a workspace for modifying at a given time. However, more than one developer can open a workspace for reading. When you open a workspace, you have the option of opening it for reading only, allowing other developers to gain read access to the workspace as well.

When you open a workspace for modifying, the plans in the workspace are available for modification. For projects, the individual project components are in one of two states:

Writeable When a component is writeable, you can examine it and modify it. A component is writeable when any of the following is true:

- you created the component after the last time you integrated
- you used a Checkout command to check out the component
- you used a Branch command to branch the component

Read Only When a component is read only, you can examine it, but you cannot modify it. A component is read-only when any of the following is true:

- The component exists in the system baseline, but you have not yet checked it out. In this case, you can check out or branch the component to get write access to it.
- The component was previously in the system baseline, and another workspace has checked it out and integrated it. To get write access to the component, you can either branch the component, or you must first update your workspace and then check out the component.
- Another developer has checked out the component. In this case, the only way to get write access to the component is to branch it.
- iPlanet UDS Express or iPlanet Integration Server generated the component. You cannot modify components generated by Express or iIS.

Creating a Workspace

You create workspaces in the Repository Workshop. To create a new workspace, use the File > New Workspace command to initialize a new workspace that contains the iPlanet UDS libraries. You then use the Plan > Include Public Plan command to add plans to the workspace.

You should only include the plans that you need, because the more plans you have in your workspace, the less efficient it is for updating, integrating, and other operations such as detaching a shadow. Normally, you include all plans that will be suppliers for the plan on which you are working (see "Supplier Plans" on page 209 for information about supplier plans for projects).

Using a Workspace

Using a workspace most often means collaborating with others on the development of an application. When you collaborate with others, you must make sure to update and integrate your workspace appropriately.

➤ **To use a workspace for collaboration**

1. When you enter the Repository Workshop, open the workspace you wish to use (if it is not already open). If necessary, you can create a new workspace and include in it all the plans you need to work on and their supplier plans.

   If you wish to create a new plan or modify an existing plan, you must open the workspace for modifying. You will only be able to do so if no other developers have opened the workspace (see below for further information).

2. If you wish to modify existing project components, you must either check them out or branch them in your workspace.

3. As you edit your plans, use the Save All command to save your changes in the workspace.

4. To keep your workspace synchronized with the changes other developers are making in the repository, choose the File > Update Workspace command.

5. When your workspace is up to date and your plans are fully tested, use the Integrate Workspace command to integrate your work into the baseline.

   Other developers can use the Update Workspace command to see your changes in their workspaces.

Choosing a Workspace

You can open any workspace from the Repository Workshop. If you are using a central repository, you should use a workspace that contains only the plans you need for your work. If you are using a private repository, you can simply use the prefabricated workspace called FirstWorkspace.

To create a new plan or modify an existing plan, you must open the workspace for modifying. Only one person can use the workspace for modifying at a given time. If another developer has already opened the workspace for modifying, no one else will be able to open the workspace. In addition, if any other developers have opened the workspace for reading only, no one will be able to open it for modifying. A single workspace can be opened by multiple users only if all users have it open for reading only.

Checking Out Project Components

To modify an existing project component, you must either check it out or branch it in your workspace. In the Project Workshop, you can check out or branch all project components with a single command, or you can checkout or branch individual project components, that is, specific classes, interfaces, cursors, constants, or service objects. Obviously, if you are collaborating with other developers on a single project, you should only check out the individual components that you need to modify. See "Write Access to Project Components" on page 257 for information about checking out individual project components.

For iPlanet UDS Express and iPlanet Integration Server plans, you must either check out or branch the entire plan in your workspace before you can modify it. The Repository Workshop provides Checkout and Branch commands for this purpose. For information on getting write access to Express plans, see the Express documentation. For information on getting write access to iIS plans, see the iPlanet Integration Server documentation.

Saving a Workspace

When you are creating a new plan or modifying an existing one, you need to save the changes you make to your workspace. Every iPlanet UDS Workshop has a File > Save All command. This command commits all changes made since the previous Save All command.

Updating a Workspace

As you proceed with your development work, you need to keep your workspace synchronized with the changes other developers are integrating into the repository. To do this, use the Update Workspace command in the Repository Workshop. For projects, this command merges the changes in the system baseline with the changes in your workspace, so you can test your own work against the latest version of the system baseline.

You can use the Update Workspace command as often as necessary to keep concurrent with other developers. Typically, developers cycle several times through the process of editing a project, updating the workspace, and testing the project, before final integration.

Integrating a Workspace

When your workspace is updated and the project (or plan) is completely tested in the updated workspace, you can give the Integrate Workspace command to incorporate your changes into the system baseline. After you integrate, any other developers that use the Update Workspace command will have your changes added to their workspaces. If you have created a new plan, using the Integrate Workspace command will make the new plan available to other developers.

Using the Repository Workshop

The Repository Workshop is automatically opened when you start iPlanet UDS. This workshop functions as the control center of the iPlanet UDS Workshops. From the Repository Workshop, you can access any of the other workshops you need to examine or modify plans.

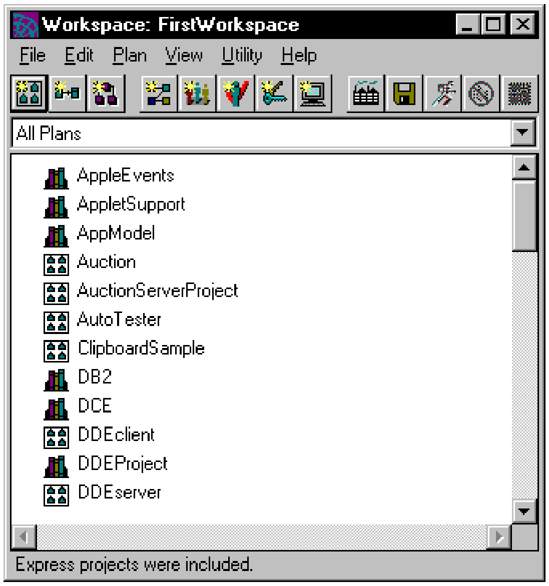
The Repository Workshop Window

The Repository Workshop window consists of a Plan browser and a toolbar. The Plan browser displays a list of the plans in the current workspace. The toolbar provides quick access to the Debugger and the Partition Workshop, as well as Save All, Run Project, and Compile All Plans buttons. It also provides a button for creation of a project.

If you have installed add-on products, such as iPlanet UDS Express or iPlanet Integration Server, the toolbar may include additional buttons.

Figure 3-2 illustrates the Repository Workshop and Figure 3-3 illustrates the Repository Workshop toolbar.

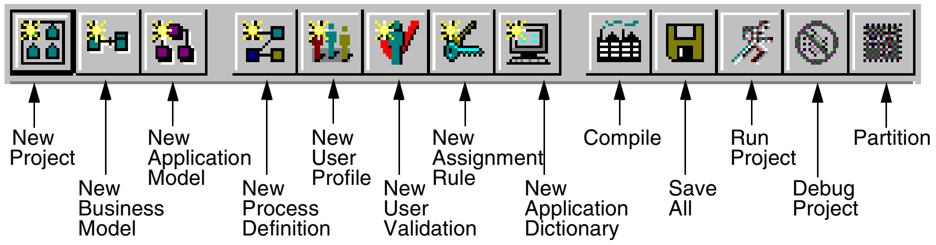
Figure 3-2 Repository Workshop



AppleEvents
AppletSupport
AppModel
Auction
AuctionServerProject
AutoTester
ClipboardSample
DB2
DCE
DDEclient
DDEProject
DDEserver

Express projects were included.

Figure 3-3 Repository Workshop Toolbar



View Menu

The View menu in the Repository Workshop provides the following toggles that let you control which parts of the workshop are displayed:

Command	Function
Toolbar	Makes the toolbar visible or invisible.
Writeable Icon	Makes the writeable icons for plans visible or invisible.
Kind Icon	Makes the kind icon for plans visible or invisible.
Status Line	Makes the status line visible or invisible.

Note that you can set your viewing preferences for the workshop by using the Workshop Preferences command. These preferences are saved as part of your current workspace. See "Setting Workshop Preferences" on page 196 for information.

Access to Other Workshops

From the Repository Workshop, you can access the following workshops:

Workshop	How to access it
Project	Choose the Plan > New Project… command to create a new project. Double-click a project name, or select a project name and choose the Plan > Open command to open an existing project.
Partition	Choose the Plan > Run > Partition… command to partition the application defined by the selected project, or single-click the Partition Workshop button. (This is not available when you are running standalone.)
Debugger	Choose the Plan > Run > T est Debug Project command to debug the application defined by selected project, or single-click the Debugger button.

Workshop	How to access it
Business Model	Choose the Plan > New Business Model… command to create a new business model. Double-click a business model name, or select a business model name and choose the Plan > Open command, to open an existing business model. The Business Model Workshop is only available if you have iPlanet UDS Express installed.
Application Model	Choose the Plan > New Application Model… command to create a new application model. Double-click a application model name, or select a application model name and choose the Plan > Open command, to open an existing application model. The Application Model Workshop is only available if you have iPlanet UDS Express installed.
Process Definition	Choose the Plan > New Process Development Plans > Process Definition… command to create a new process definition. Double-click a process definition name, or select a process definition name and choose the Plan > Open command, to open an existing process definition. The Process Definition Workshop is only available if you have iPlanet Integration Server installed.
Application Dictionary	Choose the Plan > New Process Development Plans > Application Dictionary… command to create a new application dictionary. Double-click an application dictionary name, or select an application dictionary name and choose the Plan > Open command, to open an existing application dictionary. The Process Definition Workshop is only available if you have iPlanet Integration Server installed.
User Profile	Choose the Plan > New Process Development Plans > User Profile… command to create a new user profile. Double-click a user profile name, or select a user profile name and choose the Plan > Open command, to open an existing user profile. The User Profile Workshop is only available if you have iPlanet Integration Server installed.
User Validation	Choose the Plan > New Process Development Plans > User Validation… command to create a new user validation. Double-click a user validation name, or select a user validation name and choose the Plan > Open command, to open an existing user validation. The User Validation Workshop is only available if you haveiPlanet Integration Server installed.

Workshop	How to access it
Assignment Rule	Choose the Plan > New Process Development Plans > Assignment Rule… command to create a new assignment rule. Double-click an assignment rule name, or select an assignment rule name and choose the Plan > Open command, to open an existing assignment rule. The Assignment Rule Workshop is only available if you have iPlanet Integration Server installed.

Leaving the Repository Workshop

To leave the Repository Workshop, choose the File > Exit command to end your development session and exit from the iPlanet UDS Workshops. If you have not saved your workspace before giving the command, you will be prompted to do so.

Using a Repository

The Repository Workshop lets you perform the following tasks on your repository:

- view repository information
- create and delete plans
- create and use shadow repositories
- backup a repository

Examining the Repository

The Repository Workshop provides the following information about the current repository:

- workspace status information
- repository information

Viewing Workspace Status

To view the workspace status information, use the Utility > Show Locked Workspaces command. This opens the Locked Workspaces window, where you can see a list of all the workspaces that are currently open and a list of the workspaces that have locks on the repository. When you have finished viewing the workspace status information, simply close the window.

Viewing Repository Information

To view the repository information, use the Utility > Show Repository Info command. This command opens the Repository Info window, where you can see the name of the repository, its creation date, and whether the current repository is a shadow or not. If the repository is a shadow, this window shows a list of the workspaces that are currently cached in your shadow.

The Repository Info window also provides information that may be useful when you are troubleshooting. When you have finished viewing the repository information, simply close the window.

Creating and Deleting Plans

The Repository Workshop provides the following commands for creating new plans:

Command	Description
New Project…	Creates a new project and opens the Project Workshop.
New Business Model…	Creates a new business model and opens the Business Model Workshop. Available only if iPlanet UDS Express is installed.
New Application Model…	Creates a new application model and opens the Application Model Workshop. Available only if iPlanet UDS Express is installed.
New Process Development Plans	This slide-off menu allows you to create any of the process development plans, opening the appropriate iIS Workshop. Available only if iPlanet Integration Server is installed.

When you choose a command on the Plan menu that creates a new plan, iPlanet UDS registers the new plan name in the repository so that no other developers can use the name. However, the plan itself is not visible to other developers until you use the Integrate Workspace command to integrate the new plan into the system baseline.

See "Using the New Project Command" on page 228 for information about the New Project command. See the Express documentation for information on the Express commands and the iPlanet Integration Server documentation for information on the iIS plans.

"Deleting Plans from the Repository Baseline" on page 163 provides information about deleting plans from the repository.

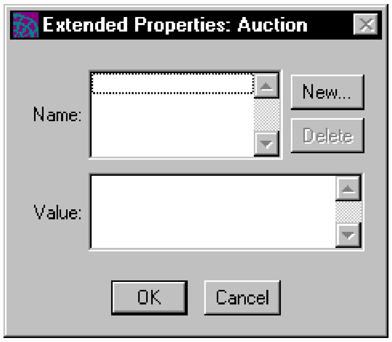
Setting Extended Properties for Projects

The projects you create in your repository have special properties called *extended properties*. This feature allows you to assign names and values to a project. Extended properties are used primarily for integrating with external systems (see *Integrating with External Systems* for more information). However, you can use an extended property for anything you wish, for example, for a comment.

➤ **To set extended properties for a project**

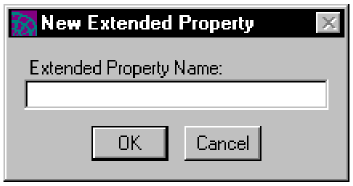
1. Select the project for which you wish to set the extended properties.
2. Choose the Plan > Extended Properties... command.

   The Extended Properties dialog opens.



3. Click the New... button.

   The New Extended Property dialog opens.

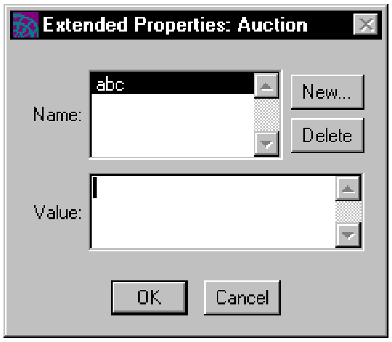


4. Enter the name of the property you wish to create and click OK.
5. In the Value field, enter the value of the extended property.
6. To enter additional extended properties, repeat steps 3 though 5.
7. Click OK.

➤ **To delete extended properties for a project**

1. Select the project whose extended property you wish to delete.
2. Choose the Plan > Extended Properties... command.

   The Extended Properties dialog opens.



3. Select the name of the extended property you wish to delete.
4. Click the Delete button.
5. Click the OK button.

Deleting Plans from the Repository Baseline

Before you can delete a plan from the repository baseline, the plan must already be deleted from all workspaces in the repository (see "Deleting Plans from a Workspace" on page 182 for information). Once the plan is no longer included in any workspaces, you can use the Utility > Delete Public Plan… command to remove it permanently from the repository.

➤ **To delete a plan from the repository**

1. Choose the Utility > Delete Public Plan… command.
2. In the Delete Public Plan dialog, select the plan you wish to delete.
3. Click the Delete button to delete the plan.
4. Confirm that you wish to delete the plan.

CAUTION If your repository contains plans created by iPlanet UDS Express or iPlanet Integration Server, even if you have not installed the products that were used to create the plans, you can delete these plans using the Utility > Delete Public Plan… command. Obviously, you should consult with your coworkers before deleting any unknown plans from the repository.

Creating and Using Shadow Repositories

The Repository Workshop allows you to create a shadow of the current central repository. The new shadow that you create is attached to the central repository.

Normally you do most of your work using an attached shadow. However, if you wish to work outside of the distributed development, you can detach your shadow and move the shadow files to another system.

The following sections describe how to create new shadows, how to work with attached shadows, and how to detach and attach shadows.

Creating a Shadow Repository

The New Shadow command creates a shadow of the current central repository.

When you create the shadow, you specify a name for it. The shadow name must be eight or fewer characters.

After the shadow is created, it automatically becomes your current repository.

➤ To create a new shadow

1. Choose the Utility > New Shadow command.
2. In the New Shadow dialog, specify the shadow name.
3. Click the OK button to create the new shadow.

The shadow repository is stored in the following two files:

System	File Names
UNIX	**$FORTE_ROOT/repos/***shadow_name***.btd** **$FORTE_ROOT/repos/***shadow_name***.btx**
OpenVMS	**$FORTE_ROOT:[REPOS]***shadow_name***.btd** **$FORTE_ROOT:[REPOS]***shadow_name***.btx**
Windows	**FORTE_ROOT\repos\***shadow_name***.btd** **FORTE_ROOT\repos\***shadow_name***.btx**

Using an Attached Shadow Repository

Normally when you give a Save All command in an attached shadow, your changes are automatically committed to the central repository. However, if you wish to improve repository performance when you are working with attached shadows, you can set your preferences so that automatic committing to the central repository is turned off. When automatic committing is turned off, changes you make in the workspace are not made in the central repository until you give a Commit to Central command or until you leave the iPlanet UDS Workshops.

The advantage to turning off automatic committing is that the central repository can handle a larger number of users or performance for the current number of users is improved. The disadvantage to turning off automatic committing is that you can lose the changes you make in your workspace. Therefore, we recommend you only turn off automatic committing when increasing repository performance is critical.

To turn off automatic committing, use the File > Workshop Preferences… command. In the Repository Workshop Preferences dialog, set the Commits to Central toggle to off. (See “Setting Workshop Preferences” on page 196 for information about using the Workshop Preferences command.)

Once you turn off automatic committing, you can proceed with your work as usual. The only restriction is that you cannot use the Undo Deleted Checkouts command.

Note that even though you turn off automatic committing, the following commands will automatically write the changes in the current workspace to the central repository:

- Integrate Workspace
- Update Workspace
- Include Public
- Delete
- Delete Public Plan
- Import (if it creates a plan)
- commands that create new plans

When automatic committing is turned off, use the Commit to Central command to write the changes in the current workspace to the central repository. Using the Commit to Central command ensures that your worked is backed up.

Multiple Workspaces in a Shadow

While the shadow is attached, you can switch to another workspace using the Open Workspace command. You can access up to 15 different workspaces using one shadow repository.

You can find out what workspaces have been cached in the shadow repository by using the Utility > Show Repository Info command. However, you cannot delete a workspace from a shadow repository after you have accessed it. If you want to access a sixteenth workspace using a shadow repository, you need to create a new shadow repository.

NOTE A cached workspace synchronizes itself with the central repository by applying to itself the changes made in the central repository. This synchronization occurs when you open a workspace in an attached shadow. If you have not synchronized a workspace in the shadow with the central repository in a long time, the workspace might not open in the shadow repository, because it cannot completely synchronize itself with the central repository. If this occurs, you need to create a new shadow for accessing this workspace.

Detaching a Shadow Repository

The Utility > Detach Shadow command disconnects the shadow repository from the central repository. The first time you use the Detach Shadow command, iPlanet UDS copies everything in the workspace into the local shadow, so the command may take a long time to execute. However, the next time you use the Detach Shadow command for the same shadow (after having re-attached the shadow), the command copies only the changed components so it will not take as long.

The detached shadow contains one accessible workspace, the workspace that was open when you gave the Detach Shadow command. Once the shadow is detached from the central repository, you can copy the shadow files to another machine, using standard operating system copy commands.

Before you give the Detach Shadow command, be sure to checkout all project components you wish to modify while the shadow is detached. Also, the workspace that you wish to use in the detached shadow should be open, because once you detach the shadow, you will not be able to open a different workspace.

After you give the Detach Shadow command, the current workspace becomes “reserved” and no other developers can open it until you give an Attach Shadow command to make it available again. Note that if you need to free the lock on the workspace without attaching the shadow, you can use the `ForceWorkspaceUnreserved` command in Fscript. However, the `ForceWorkspaceUnreserved` command makes it impossible for you to reconnect the shadow, and is therefore only for use when there is problem with the detached shadow. For information, see the `ForceWorkspaceUnreserved` command in the *Fscript Reference Guide*.

➤ **To detach your shadow and move it to another machine**

1. In the Repository Workshop, choose the Utility > Detach Shadow command.
2. Use the Exit command to end your iPlanet UDS development session.
3. Copy the .btd and .btx shadow files to the workstation where you want to use the detached shadow. These files are portable across all installations supported by iPlanet UDS.

   Note that if you use FTP, you should use a binary mode copy to copy the shadow files.

4. Start iPlanet UDS on the workstation where you want to use the detached shadow.

When you start the iPlanet UDS Workshops, use the `-fr` flag to specify the full path name for the shadow repository and the `-fw` flag to specify the name of the workspace that was open when you gave the Detach Shadow command. If you are working outside the distributed development environment, use the `-fs` flag to indicate that you are running in standalone mode. For example:

```
forte -fr bt:myshadow -fw MyWorkspace -fs
```

When you are ready to add your changes to the central repository, you must move your shadow back to a node in the distributed development environment, copy it into the appropriate directory (as described under "Creating a Shadow Repository" on page 163), and use the Attach Shadow command to connect the shadow back to the central repository.

NOTE You should never make changes to multiple copies of a detached shadow. Once you attach a detached shadow back to the central repository, you cannot attach another copy of the detached shadow to it. Therefore, any changes you made in a separate copy of the detached shadow would be lost.

Backing Up a Detached Shadow

When you work in an attached shadow, your work is automatically backed up because every change your make locally is automatically made in the central repository. However, when you work in a detached shadow, the work you do locally is not backed up in the central repository. If you run out of disk space or have a problem with your local disk, your work in the detached shadow could become corrupted. Therefore, you should backup your changes in the detached shadow by:

- using the Utility > Backup Repository... command (see "Backing Up a Repository" on page 168)
- making copies of the shadow files (see "Creating a Shadow Repository" on page 163)
- using the Export command to export the project or projects to a file

Of course, if you are working within the distributed development environment, you can use the Attach Shadow command to connect to the central repository, which automatically copies your changes from the shadow to the central repository, and then use the Detach Shadow command to detach again.

Attaching a Shadow

The Utility > Attach Shadow command lets you attach a shadow repository back to the central repository. This command copies all the changes you made in the detached shadow to the central repository.

➤ **To attach a detached shadow**

1. Start iPlanet UDS, using the `-fr` flag to specify the full path name for the shadow repository and the `-fw` flag to specify the name of the workspace that was open when you gave the Detach Shadow command.

   Do not use the `-fs` flag, because you are going to attach to a distributed central repository.

2. In the Repository Workshop, choose the Utility > Attach Shadow command.

Backing Up a Repository

For private repositories and detached shadows of central repositories, you can backup the repository by using the Utility > Backup Repository command to write the repository to a file.

➤ **To backup the repository**

1. Choose the Utility > Backup Repository command.
2. On the file selection dialog, select the directory where you wish to store the backup file. The file name for the backup file is the same name as the repository.

After you use the Backup Repository command once, each time you exit from the iPlanet UDS Workshops and are saving your changes, you will be prompted to backup the repository again.

Using Workspaces

The Repository Workshop lets you perform the following tasks for workspaces:

- open a workspace
- examine a workspace
- save a workspace
- update and integrate a workspace

- create a workspace and include plans in it
- create and delete workspaces
- import and export plans
- find and replace text in any method source code in the workspace

Opening a Workspace

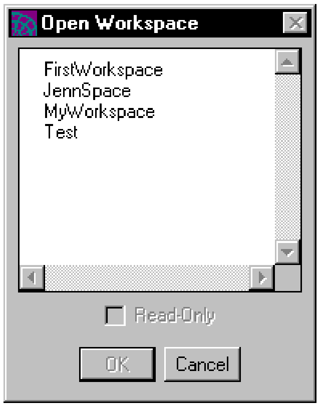
If you are not already in the workspace you wish to use, you can use the Open Workspace command to open any workspace in the repository. When you open a workspace, you have the option of opening it for reading only. If you are only planning to examine the plans in the workspace and not to modify them, you should open the workspace for reading only so other developers can also open the workspace.

If another developer has opened the workspace for modifying, you will not be able to open it. In addition, if the workspace is reserved (that is, if it is currently in use by a detached shadow), you will not be able to open it.

➤ **To open a workspace**

1. Choose the File > Open Workspace command.
2. In the Open Workspace dialog, select the workspace you wish to open.

   The workspace list indicates which workspaces are reserved and cannot be opened for modification.



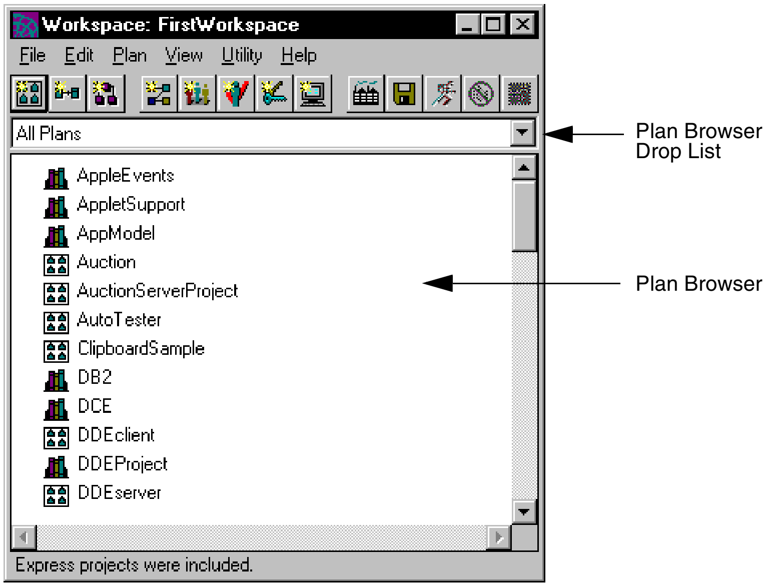
3. If you do not plan to modify any of the plans in the workspace, click the Read-only toggle to on.

If you try to open a workspace that is already open for modifying by another developer, you will get an error message.

Examining a Workspace

To view the complete list of all the plans in the workspace, select "All Plans" from the drop list in the Plan browser, shown in the figure below.

Figure 3-4 Plan Browser



By default, the Plan browser lists all the plans in your workspace. Icons indicate the type of plan.

Icon	Plan
	Project
	Library
	Business model (created by iPlanet UDS Express)

Icon	Plan
	Application model (created by iPlanet UDS Express)
	Process definition (created by iPlanet Integration Server)
	Application dictionary (created by iPlanet Integration Server)
	User profile (created by iPlanet Integration Server)
	User validation (created by iPlanet Integration Server)
	Assignment rule (created by iPlanet Integration Server)

To view a list of only one kind of plan (or a subset of the plans), use the filter drop list for the Repository Workshop. The filter drop list allows you to select a single plan kind, such as "Project "or "Process Development Plans," or various combinations, such as "Projects & Libraries."

Note that you can set your filter preferences for the workshop by using the Workshop Preferences command. These preferences are saved as part of your current workspace. See "Setting Workshop Preferences" on page 196 for information.

iPlanet UDS also displays a writeable icon by iPlanet UDS Express and iPlanet Integration Server plan names. The writeable icons indicate whether the current workspace has write access to the plan, and shows whether it is new, checked out, or branched. A blank indicates that the plan is read only.

Icon	Plan
●	New
✓	Checked Out
⑂	Branched

To turn off the writeable icons, set the View > Writeable Icon toggle to off. See the iPlanet UDS Express and iPlanet Integration Server documentation for information on getting write access to Express and iIS plans respectively.

Sorting by name or kind By default, the list of plans is in alphabetical order, by name. If you wish to view the same list sorted by plan type, choose the View > Sort by Kind command. The Sort by Name command specifies the default, sorting all plans by name.

Viewing the plan definition To display the complete definition of a plan, double-click on the plan name, or select the plan name and choose the Plan > Open command. This command opens the appropriate workshop, which displays the full definition of the plan. For information about using the Project Workshop, see Chapter 4, "Using the Project Workshop." For information about using the workshops for add-on products, see the documentation for the individual product.

You can view the integration history for any workspace in the repository. The File > Show Integration History… command shows the integration history for the workspace name you specify or for all workspaces in the repository that match the specified search string. To specify a search string, you can use an asterisk as a wild card at the end of a character string.

➤ To display the integration history for a workspace

1. Choose the File > Show Integration History… command.
2. In the Show Integration History dialog, specify the workspace name or a search string. The default is an asterisk to specify all workspaces.

   You can also specify the number of integrations, counting back from the most recent, you wish to display.
3. Click the OK button to open the Integration History window.

   This window displays the integration number, date of integration, and comment for each integration.
4. When you have finished viewing the integration information, close the Integration History window.

The Repository Workshop allows you to view all the persistent breakpoints that have been set in the code in the selected project. The Utility > Show Breakpoints command opens the Debugger's Global Breakpoint Manager window to display all the persistent breakpoints in the project.

You can use the Global Breakpoints Manager window to browse through the breakpoints or to delete them. You cannot add new breakpoints with the Global Breakpoints Manager.

➤ To browse through the breakpoints

1. Scroll through the list of breakpoints using the window's scroll bar.
2. To see a single breakpoint in context, double-click the breakpoint.

   The Method, Event Handler, or Cursor Workshop opens to the code where the breakpoint was set. Once in the workshop, you can change breakpoints or modify the code as usual.

➤ To delete a breakpoint

1. Click the stop sign icon for the breakpoint.

You can remove all breakpoints with a single command by choosing the File > Clear All command on the Global Breakpoint Manager's File window.

When you are finished using the Global Breakpoints Manager, close the Global Breakpoint Manager window by using the File > Close command.

See "Setting Breakpoints" on page 582 for information on persistent breakpoints and the Global Breakpoint Manager window.

To display the extended properties for the project, choose the Plan > Extended Properties... command. The Extended Properties dialog opens, displaying the current settings for the extended properties. Select the name of the extended property to view its value.

Saving a Workspace

The Save All command writes all changes made to anything in the current workspace since the last Save All command permanently to disk. You should use the Save All command frequently to ensure that you do not lose work. The Save All command is available on the File menu of every iPlanet UDS workshop. It is also available through the Save All button on the toolbar.

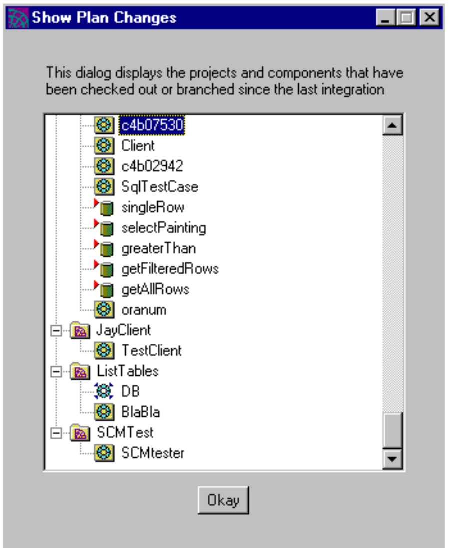
Showing Plan Changes

When working in a central repository or an attached shadow of a central repository, you need to notify other developers what changes you will integrate into the system baseline. The Repository Workshop File > Show Plan Changes command gives you a tree-view list of all projects and components that have been checked out or branched since the last integration. Use this dialog before integrating to make note of all changes you will make to the system baseline.

➤ **To view plan changes**

1. Choose File > Show Plan Changes command.

   The Show Plan Changes dialog opens:



   Use the information in this dialog to alert other developers about the changes you will integrate into the system baseline.

2. Click OK or close the window when you have made note of workspace plan changes.

Updating and Integrating a Workspace

If you are working with a central repository or an attached shadow of a central repository, you need to keep your workspace synchronized with the changes other developers are integrating into the repository. To do this, use the Update Workspace command. This command merges the changes that have been integrated into the system baseline with the changes in your workspace, so you can test your own work against the system baseline.

You can use the Update Workspace command as often as necessary to keep current with other developers. Normally, developers cycle several times through the process of editing a project, updating the workspace, and testing the project, before final integration.

If you have made changes in your workspace since your last Update Workspace command that you wish to erase, you can use the File > Revert command in the Project Workshop. The Revert command reverts all changes you have made to the project and its components since your last Update Workspace command, including creations, deletions, and modifications. See "Reverting a Project" on page 256 for information on the Revert command.

When your project is fully tested in the updated workspace, you can give the Integrate Workspace command to incorporate your changes into the system baseline. After you integrate, any other developers that use the Update Workspace command will have your changes added to their workspaces. If you have created a new plan, using the Integrate Workspace command will make the new plan available to other developers. Integrating also automatically "checks in" any components you had checked out in the workspace, and frees them for checkout by other workspaces.

Updating a Workspace

The File > Update Workspace command updates the contents of the current workspace with any changes integrated into the repository since the last time you gave an Update Workspace command for this workspace. Use this command from time to time to synchronize with concurrent changes made to any supplier plans.

Compiling all plans The Update Workspace command provides the option of compiling all plans after updating the workspace. Turning on this option in the Update Workspace dialog is equivalent to giving the Compile All Plans command as described under "Compiling Plans" on page 191.

Conflicting branches If you have branched items in your workspace and you then give an Update Workspace command, there is potential conflict between your component and the system baseline, because another workspace may have integrated one or more of the components you have branched. We call these "conflicting branches," because the changes made in the system baseline conflict with the changes you have made in your workspace.

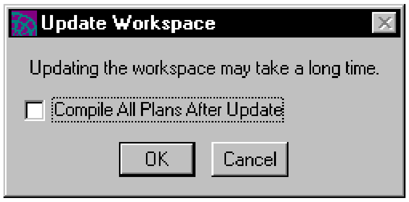
When you use the Update Workspace command and there are conflicting branches, iPlanet UDS gives you the option of cloning the conflicting component so that you do not lose your changes. The original components will be overwritten with the changes from the system baseline, and the cloned components will have the characters "_Branch" appended to their names. After a component is cloned, you can compare the cloned component side by side with the updated component, make any appropriate modifications, and then delete the extra component.

The Update Workspace command notifies you when there are conflicting branches, and you have the option of requesting that individual components not be cloned. However, when you do not clone the components, all changes you have made to them while they were branched are permanently lost.

➤ **To update your workspace**

1. Choose the File > Update Workspace command.

   The Update Workspace dialog opens.

   
   Update Workspace

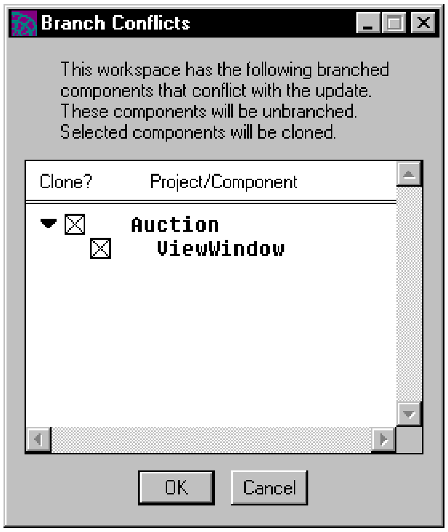
   Updating the workspace may take a long time.

   ☐ Compile All Plans After Update

   If you wish to compile all plans after updating the workspace, click the Compile All Plans After Update toggle.

2. Click the OK button to update the workspace.

3. If there are any branched components, iPlanet UDS opens the Branch Conflicts dialog, which lists all components that were integrated by another workspace since you branched the components.

   
   Branch Conflicts

   This workspace has the following branched components that conflict with the update. These components will be unbranched. Selected components will be cloned.

Clone?	Project/Component
☒	Auction
☒	ViewWindow

By default, all conflicting components are cloned. If you do not wish to clone a branched component, click off the toggle next to that component's name. Then, click the OK button to close the dialog.

4. If you made changes since the last time you saved your workspace, you will be prompted to save it.

Only changes to projects that have been integrated by other workspaces into the repository will be updated into your workspace. You will not see changes made in other workspaces that have not yet been integrated.

If the Update Workspace command fails, any changes the command made to the repository are backed out and the repository is returned to the same state it was in before you gave the command.

The Update Workspace command can fail in the following circumstances:

- the client crashes

  Use the Fscript `UnlockWorkspace` command or RepositoryServer agent `UnlockWorkspace` command to notify the repository server that the client is no longer active. The `UnlockWorkspace` command unlocks the workspace and any global repository locks held by that workspace. See *Fscript Reference Guide* for information on the Fscript `UnlockWorkspace` command. See *Escript and System Agent Reference Guide* for information on the RepositoryServer agent `UnlockWorkspace` command.

- the repository server crashes

  When the repository server is restarted, you can simply give the Update Workspace command again.

- an error occurs while the command is processing

  The workspace is rolled back to the state it was in before you gave the Update Workspace command. You should examine the command's error messages to determine how to resolve the error.

Integrating a Workspace

The File > Integrate Workspace command adds any changes you made in the current workspace since the last Integrate Workspace command to the system baseline. After your Integrate Workspace command has completed, any other workspaces can use the Update Workspace command to see the changes from your workspace.

You cannot give the Integrate Workspace command if any components in your workspace are branched. You must either convert the branches to checkouts (by using the Checkout command), or undo the branch (by using the Component/Undo command). See "Write Access to Project Components" on page 257 for information about the Component/Checkout and Component/Undo commands for project components.

Before giving the Integrate Workspace command, you must first use an Update Workspace command to get up to date with the system baseline. After checking to see that your changes are compatible with any changes made to projects and components, you can then use the Integrate Workspace command to make your changes public.

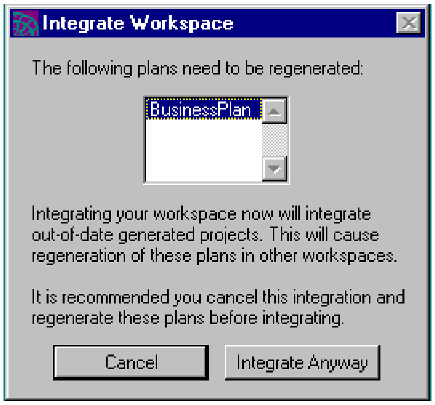
If you do not use the Update Workspace command before giving the Integrate Workspace command and your workspace is out of sync with the system baseline, you will get an error message telling you that must update your workspace. In addition, if, between the time you give the Update Workspace command and the Integrate Workspace command, some other workspace has given an Integrate Workspace command, your Integrate Workspace will be rejected. If this occurs, give the Update Workspace command again, validate that the recent changes have not caused problems, and then give the Integrate Workspace command again.

➤ **To integrate your workspace**

1. Choose the File > Integrate Workspace command.
2. In the Integrate Workspace dialog, enter the integration comment. This comment is stored in the repository, and is displayed as part of the integration history by the Show Integration History and Show Component History commands.

If your workspace contains iPlanet UDS Express or iPlanet Integration Server plans that need to be regenerated, the Integrate Workspace command displays a warning dialog. This warning displays a list of the out-of-date plans.

Figure 3-5 Integrate Workspace Dialog



The following plans need to be regenerated:

Integrating your workspace now will integrate out-of-date generated projects. This will cause regeneration of these plans in other workspaces.

It is recommended you cancel this integration and regenerate these plans before integrating.

If you see this warning dialog, we recommend that you cancel the Integrate Workspace command and regenerate the plans. After the plans are regenerated, you can give the Integrate Workspace command again. See the Express and iPlanet Integration Server documentation for information about regenerating Express and iIS plans.

If the Integrate Workspace command fails, any changes the command made to the repository are backed out and the repository is returned to the same state it was in before you gave the command.

The Integrate Workspace command can fail in the following circumstances:

- the client crashes

  Use the Fscript `UnlockWorkspace` command to notify the repository server that the client is no longer active. The `UnlockWorkspace` command unlocks the workspace and any global repository locks held by that workspace. See *Fscript Reference Guide* for information.

- the repository server crashes

  When the repository server is restarted, you can simply give the Integrate Workspace command again.

- an error occurs while the command is processing

  The workspace is rolled back to the state it was in before you gave the Integrate Workspace command. You should examine the command's error messages to determine how to resolve the error.

Creating a Workspace

To create a workspace, you use the New Workspace… command to create a named workspace that includes the iPlanet UDS libraries. You can then add any plans in the repository to the workspace with the Include Public… command. To remove a plan from the workspace, use the Edit > Delete command.

For performance reasons, you should use the New Workspace command on the central repository before creating the shadow where you intend to use the workspace.

➤ **To create a new workspace**

1. Choose the File > New Workspace command.
2. In the New Workspace dialog, specify the workspace name.
3. Click the OK button to create the workspace.

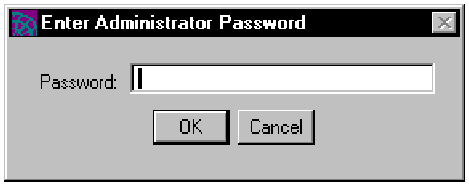
After you give the New Workspace command, the new workspace is your current workspace.

To create a new workspace for a secure repository, you need to specify the administrator password.

➤ **To create a new workspace in a secure repository**

1. Choose the File > New Workspace command.

   The Enter Administrator Password dialog opens.

2. Enter the administrator password.

   The New Workspace dialog opens.

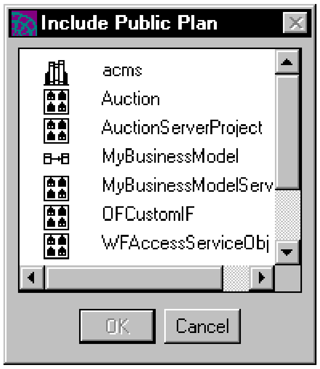
3. In the New Workspace dialog, enter the workspace name, the new password for the workspace, and the verification password (retype the workspace password.)
4. Click the OK button to create the workspace.

Including Plans in a Workspace

The Include Public… command includes a plan in the current workspace. If a newly included public plan has supplier plans that are not yet included in the current workspace, those will automatically be included in your workspace as well.

➤ To include a plan in the workspace

1. Choose the Plan> Include Public… command.
2. On the Include Public Plan dialog, select the plan you wish to include.



3. Click the OK button to include the project.

The Include Public… command brings the system baseline version of the plan into your workspace. Therefore, after you give an Include Public… command, we recommend that you give the Update Workspace command to synchronize newly included plans with the other plans in the workspace.

Deleting Plans from a Workspace

To remove a plan from a workspace, use the Edit > Delete command. Before you can delete the plan, you must make sure that neither the plan nor any of its components are checked out.

➤ To remove a plan from the workspace

1. Select the plan name.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the plan from the workspace.

Removing a plan from your workspace has no effect on the system baseline. It merely means that you cannot view or update the plan from your workspace. However, if you have created the plan in the workspace but never integrated it, the Delete command will permanently delete the plan, because it exists only in the current workspace, not in the system baseline.

For information on using the Delete Public Plan command to remove a plan from the system baseline, see "Deleting Plans from the Repository Baseline" on page 163.

Deleting a Workspace

The File > Delete Workspace command deletes the current workspace from the repository. If there are any plans in the workspace that have never been integrated, these will be also be deleted, so be careful using this command.

To delete the workspace, the workspace must be open for modifying. In addition, nothing can be checked out to the workspace.

➤ **To delete a workspace**

1. If necessary, choose the File > Open Workspace command to open the workspace you wish to delete. You must open it for modifying.
2. If plans or project components are checked out in the workspace, use the appropriate Undo command to release the checkouts.
3. Choose the File > Delete Workspace command.
4. Confirm that you wish to delete the workspace.

Importing and Exporting Plans and Libraries

The Repository Workshop lets you import an entire plan into your workspace. Using the Plan > Import command, you can take a plan that was exported from another repository and move it into your repository. The plan that you import must have been created either by the Plan > Export command in the Repository Workshop or by the ExportPlan command in Fscript. You can also import a library that is installed in the environment into your repository.

The Export command in the Repository Workshop allows you to export the entire plan to a standard text file, which you can then import into another iPlanet UDS repository.

Importing a Plan

The Plan > Import command takes an iPlanet UDS plan definition stored in a text file and adds it to your workspace. The plan definition that you import must have been created by the Export command in the Repository Workshop or by an export command in Fscript, or it must be a .pex file for a library.

You can import a new plan or an existing plan. If the plan you are importing already exists in the current workspace, iPlanet UDS overwrites the plan, unless it is a project.

Merging or Overwriting Projects

If the project you are importing already exists in the current workspace, the Import command prompts you with an Import Plan dialog, which allows you to choose whether to overwrite or merge with the existing project.

The Overwrite button on the Import Plan dialog instructs iPlanet UDS to delete the existing project components and use the imported project definition to define the entire project.

The Merge button instructs iPlanet UDS to merge the imported project with the existing project. iPlanet UDS merges the project you are importing with the existing project as follows:

- new components are added to the existing project
- components in the existing project that are not in the imported project are left intact
- components in the existing project that are also in the project being imported are overwritten

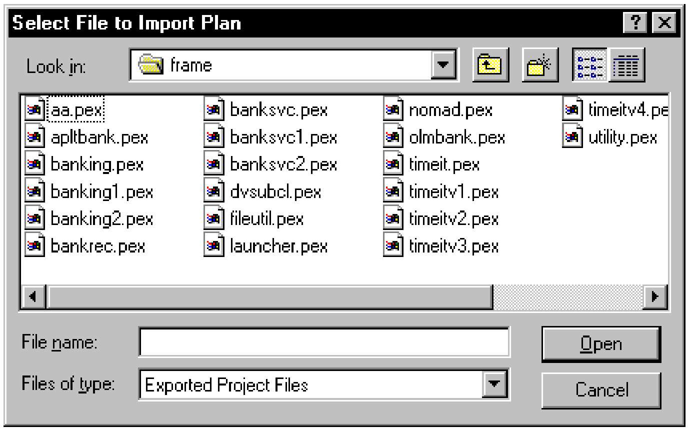
Note that if any of the components defined in the plan being imported already exist, you must be sure the existing components are new, checked out, or branched. The existing components must be in a writeable state so that the Import command can overwrite them.

Importing Project preference The Repository Workshop Preferences dialog allows you to set an Importing Project preference that affects the behavior of the Import command. By default, the Importing Project preference is set to Prompts, which prompts you with the Import Plan dialog described above. The Merges option turns off prompting for the Import command and automatically merges the imported project with the existing project. The Overwrites option turns off prompting for the Import command and automatically overwrites the existing project with the imported project. See "Importing Project Preference" on page 198 for information.

If a project you are importing has any supplier plans, these must be in your workspace before you import the project. If the supplier plans are not already in your workspace, you must import the supplier plans before importing the main project.

➤ **To import a plan**

1. Choose the Plan > Import command.



2. In the file selection dialog, specify the name of the file that contains the plan definition.

3. If you are importing a project and the project already exists in the workspace, the Import Plan dialog opens. Click the Merge button to merge with the existing project, the Overwrite button to overwrite the existing project, or the Cancel button to cancel the Import command.

After the plan has been successfully imported, iPlanet UDS displays a message indicating that it was added to the repository. If there is an error, such as a bad file or an attempt to change a read-only component, iPlanet UDS displays an error message.

After you import the plan, you can use the Integrate Workspace command to add the new plan definition to the system baseline.

Importing a Library

After a library distribution is installed in your development environment, you must import the individual libraries that it contains into each development repository where you wish to use them

Note that you can import a library into any development repository except the repository where the library was initially created. The repository that defined the library cannot access the shared library. Instead, the repository that defined the library must use the original project that provided the library definition—it can use the original project as a supplier project.

To import a library into your repository, use the Plan > Import command just as you would for any plan. The .pex file for the library contains the library definition; this is the file you must import.

➤ **To import a library**

1. In the Repository Workshop, choose the Plan > Import command.
2. In the file selection dialog, specify the name of the .pex file that contains the library definition.

After the library has been added to your workspace, you will see the library in the Repository Workshop's browser. A library icon indicates that it is a library.

Updating Library Versions

You must import the library into the repository every time a new version of it is installed in the environment. When a new version is installed in the environment, give the Import command again for the new .pex file and overwrite the existing library with the new version you are importing.

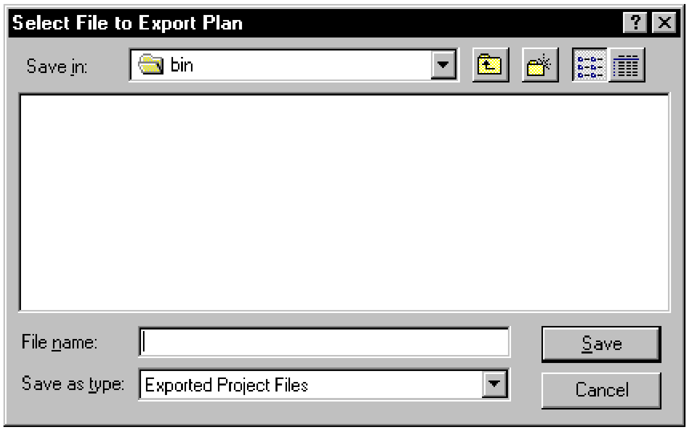
Exporting a Plan

The Plan > Export command writes the definition of the selected plan into a standard text file.

➤ **To export a plan**

1. In the Plan browser, select the plan name you wish to export.

2. Choose the Plan > Export command.



3. In the file selection dialog, specify the name of the file to contain the plan definition. If you give the name of an existing file, the Export command overwrites the file.

While the plan is being exported, iPlanet UDS displays a message indicating that the plan is being written to the specified file and prevents all input until the export is complete.

The plan is exported as it is in the current workspace. If you have made changes since the last integration, these will be included in the exported plan.

Note that if you want to save some of the plan code you wrote while you had the plan (or project components) checked out, but do not want to integrate the changes, you can use the Export command to write your plan to a text file. This preserves the modified plan in a textual form, so you can copy or edit the TOOL code that you want to save. You can also use the Export command to document a plan. The file produced by the Export command can be editing with standard text editing tools to create a definition for use with Fscript. See the *TOOL Reference Guide* for information about the definition statements in TOOL.

Finding and Replacing Text

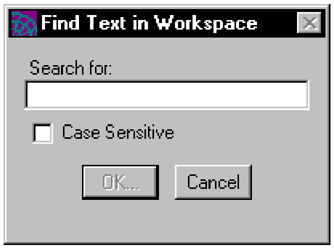
The Edit > Find Text… and Replace Text… commands in the Repository Workshop let you search for all occurrences of a specified string within the method and event handler source code for all projects in the workspace.

Finding Text

The Find Text… command finds all occurrences of the specified string in the method and event handler source code in all classes in all projects in the workspace.

➤ **To find a string**

1. Choose the Edit > Find Text… command.
2. In the Find Text in Workspace dialog, specify the string you wish to search for. The default is a case-insensitive search. If you want a case-sensitive search, set the Case Sensitive toggle to on. Click the OK button to start the search.



When the search is complete, the Find Text in Workspace dialog displays the class name, method name or event handler name, and the line itself for each source code line that contains the string.

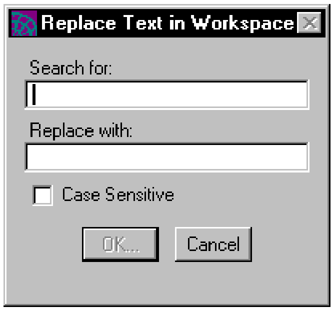
3. Double-click any line in the Find Text in Workspace dialog to open the Method or Event Handler Workshop to display the original source code.

Replacing Text

The Replace Text… command finds all occurrences of the specified string in the method or event handler source code in all classes in all projects in the workspace.

➤ To make a global replacement

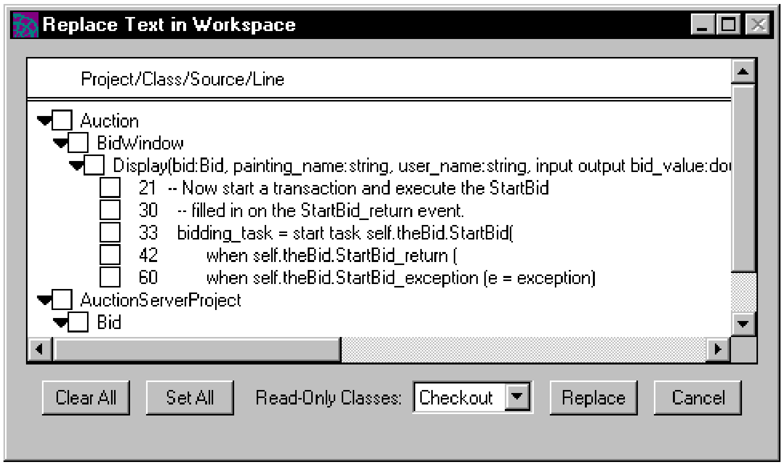
1. Choose the Edit > Replace Text… command.
2. The Replace Text in Workspace dialog opens.



3. In the Replace Text in Workspace dialog, specify the string you wish to search for and the replacement string. Click the OK button to start the search.

   The default is a case-insensitive search. If you want a case-sensitive search, set the Case Sensitive toggle to on.

4. When the search is complete, another Replace Text in Workspace dialog opens, displaying the project name, class name, method name or event handler name, and the line itself for each source code line that contains the string.



```
Project/Class/Source/Line
Auction
  BidWindow
    Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:do
      21  -- Now start a transaction and execute the StartBid
      30  -- filled in on the StartBid_return event.
      33  bidding_task = start task self.theBid.StartBid(
      42      when self.theBid.StartBid_return (
      60      when self.theBid.StartBid_exception (e = exception)
AuctionServerProject
  Bid
```

5. Click the toggles next to the lines where you wish to make the replacement.
6. In the Replace Text in Workspace dialog, click the Replace button to make the replacements.

Read-only classes By default, if a class is read-only, the Replace Text… command automatically checks it out before making the replacement in its source code. If it cannot be checked out, you will be prompted to indicate whether you wish to branch or ignore the read-only class. If you do not want read-only classes to be checked out, you can select one of the following options from the Read-only Classes drop list on the Replace Text in Workspace dialog:

Replacement Option	Description
Ignore	Will not make the replacement in the class.
Branch	Will branch the class.
Checkout	The default. Will attempt to check out the class.

The Replace Text in Workspace dialog provides several features to make it easier to select the lines where you want to make the replacement. Use the Set All button to set the toggles for all the source code lines to on. Use the Clear All button to set the toggles for all the source code lines to off. If you set the toggle for a method or event handler name to on, this turns the toggles on for all relevant lines in the method or event handler. If you set the toggle for a class name to on, this turns the toggles on for all relevant lines in all methods and event handlers in the class. If you set the toggle for a project name to on, this turns the toggles on for all relevant lines in all methods and event handlers in all classes in the project.

Testing a Plan

The Repository Workshop provides the following testing commands:

Command	Description
File > Compile All Plans	Compiles all plans in your workspace, and reports compilation errors.
Plan > Run > Test Run Project	Runs the application locally from the start method and reports errors.
Plan > Run > Test Run with Profiling	Runs the application locally, like the Test Run Project command, but also provides profiling. Provides profiling for the client partition.

Command	Description
	The profiler counts instructions executed in the TOOL interpreter, which is useful for finding problems in the interpreted code and for viewing the dynamic call flow of an application.
Plan > Run > Run Distributed	Runs the application from the start method, using the default configuration.
Plan > Run > Test Code Fragment	Executes the TOOL start-up code that you specify, which lets you run a segment of your application, or runs any TOOL code fragment.
Plan > Run > Test Debug Project	Starts the Debugger for the project, which allows you to monitor the code as it is being executed. This runs the application locally.

The following section describes how to compile the plans in your workspace. See "Testing a TOOL Project" on page 262 for information on the commands for testing a project.

Compiling Plans

The File > Compile All Plans command compiles all the plans in your workspace, and reports the compilation errors. While it is not necessary to compile your plans before running the application, this command allows you to check for syntax errors without actually executing the code. Any compilation errors are reported in the Error window.

➤ **To compile all plans**

1. Choose the File > Compile All Plans command, or single-click the Compile All button on the toolbar.

   Because compiling all the plans in the workspace will take some time, you must confirm that you wish to do it. Once the Compile All Plans command starts executing, you cannot cancel it.

The Compile All Plans command automatically saves your workspace before beginning the compilation, and saves incrementally during the compilation process.

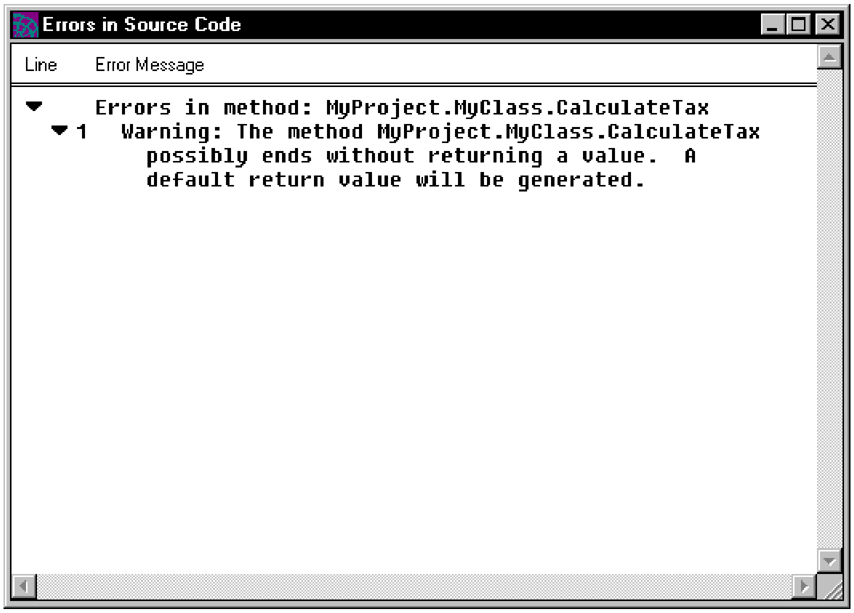
The Compile All Plans command compiles only those components that have changed since your last compilation. If you wish to compile all your components, regardless of whether or not they have changed, you can use the Utility > Force Compile command. The submenu for the Force Compile command lets you choose either All Plans, to force compilation of all plans in the workspace, or Selected Plan, to force compilation of the currently selected plan.

The following section provides information about using the Error window for project errors. For information about compiling Express models and generated projects, see the Express documentation. For information about compiling iIS plans, see the iPlanet Integration Server documentation.

Using the Error Window for Project Errors

The Error window displays the errors messages for the project in an outline field.

Figure 3-6 Error Window



```
▼     Errors in method: MyProject.MyClass.CalculateTax
  ▼ 1   Warning: The method MyProject.MyClass.CalculateTax
          possibly ends without returning a value.  A
          default return value will be generated.
```

You can jump directly from one of these messages to the code that caused the error, by double-clicking on the error you wish to find. The Method, Event Handler, or Cursor Workshop will then open, moving the cursor to the beginning of the line that contains the code that caused the error.

You can keep the Error window open as long as you need it. When you are finished using the window, simply close it.

Setting Repository Passwords

For standard repositories, you can set workspace and baseline passwords using the Repository Workshop. After setting the passwords, you can, of course, change them at any time.

For secure repositories, the system manager must set the passwords when the repository is initially created. However, after the secure repository has been created, you can change the administrator, baseline, and workspace passwords using the Repository Workshop.

Setting Passwords in a Standard Repository

For a standard repository, you can set workspace and baseline passwords. When you set a workspace password, iPlanet UDS prompts for the password when the user opens a workspace. When you set a baseline password, iPlanet UDS prompts for a password when the user integrates the workspace.

A legal repository password is a string of 7-bit ASCII characters, of any length, with no spaces. To turn password protection off, simply specify an empty password.

Workspace password To set the workspace password on the current repository, choose the File > Set Workspace Password command. On the Enter Password dialog, you must specify the existing password (if one was previously set), the new password, and the verification password (retype the new password).

Baseline password To set the baseline password on the current repository, choose the File > Set Baseline Password command. On the Enter Password dialog, you must specify the existing password (if one was previously set), the new password, and the verification password (retype the new password).

Changing Passwords in a Secure Repository

For a secure repository, you can change the administrator, baseline, workspace, and master passwords that were set when the secure repository was originally created. In the Repository Workshop, you can change the administrator, baseline, and workspace passwords. However, you must use Fscript to change the master password (see *Fscript Reference Guide*).

A legal repository password is a string of 7-bit ASCII characters, of any length, with no spaces. For secure repositories, a password is required and you cannot specify an empty password.

Administrator password To change the administrator password on the current repository, choose the File > Set Administrator Password command. On the Enter Password dialog, you must specify the existing password, the new password, and the verification password (retype the new password).

Baseline password To change the baseline password on the current repository, choose the File > Set Baseline Password command. On the Enter Password dialog, you must specify the existing password, the new password, and the verification password (retype the new password).

Workspace password To change the password for the current workspace, choose the File > Set Workspace Password command. On the Enter Password dialog, you must specify the existing password, the new password, and the verification password (retype the new password).

Utilities

The Repository Workshop provides the following two utilities:

- memory collection utility
- iPlanet UDS log utility

Collecting Memory

The Utility > Collect Memory command performs memory reclamation on the memory of the development system. Normally you do not have to use this command. However, it can be useful when you are testing timings to ensure that no extra memory reclamations will cause the timings to be invalid from run to run.

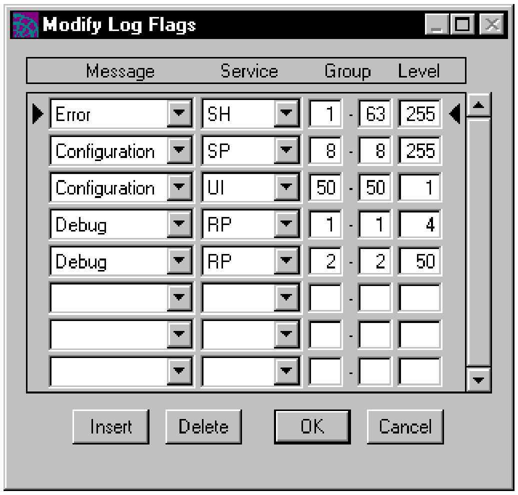
Modifying Log Flags

As you develop and test applications in the iPlanet UDS workshops, iPlanet UDS logs messages in the trace window or log file as specified by the `-fl` flag on the commands used to start the iPlanet UDS Workshops (see "Starting the iPlanet UDS Workshops" on page 110) or the FORTE_LOGGER_SETUP environment variable (described in the *iPlanet UDS System Management Guide*). If you did not specify a log file name, iPlanet UDS logs the messages in the trace window.

The Repository Workshop allows you to change the filter settings used for logging the messages. By default, iPlanet UDS uses the filter settings you specified with the `-fl` flag or, if you did not use the **-fl** flag, the filter settings specified by the FORTE_LOGGER_SETUP environment variable.

If you want to change the default filter settings at any point during your development session, you can use the Utility > Modify Log Flags command. The Modify Log Flags command opens a window, where you view and/or change the filter settings in an array field. Figure 3-7 illustrates the Log Flags dialog:

Figure 3-7 Log Flags Dialog



Message	Service	Group	Level
Error	SH	1 - 63	255
Configuration	SP	8 - 8	255
Configuration	UI	50 - 50	1
Debug	RP	1 - 1	4
Debug	RP	2 - 2	50

To change the filter settings for an individual message, edit the fields in the array row as follows:

Field	How to fill it in
Message	Select the message type from the drop list.
Service	Select the service type from the drop list.
Group	Enter integers in the two data fields to specify a range. The integers can be from 1 to 63.
Level	Enter an integer from 1 to 255. In general, lower numbers provide less detail and higher numbers provide more detail.

See "-Fl Flag (iPlanet UDS Logger)" on page 795 for information about these settings.

Inserting and Deleting Log Settings

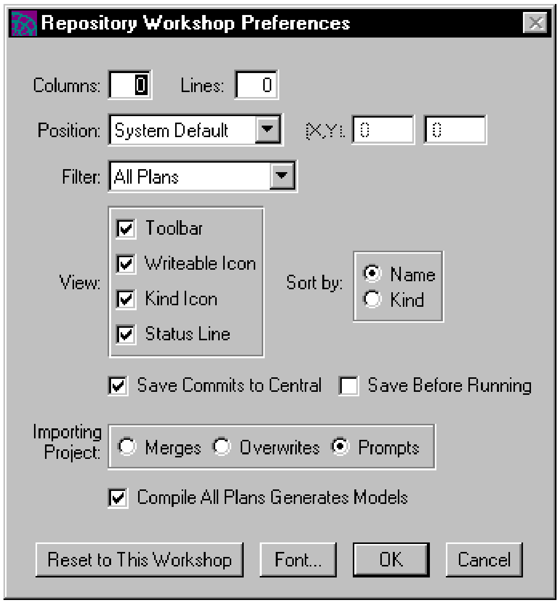
You can request an additional log setting by inserting a new row in the array field or eliminate a log setting by deleting the row from the array field. The Insert button adds a new row above the selected row, using default values for each of the fields. The Delete button removes the currently selected row.

Setting Workshop Preferences

The Repository Workshop allows you to set preferences that are saved as part of your current workspace.

To set the workshop preferences, choose the File > Workshop Preferences… command. This opens the Repository Workshop Preferences dialog, where you can set any number of preferences.

Figure 3-8 Repository Workshop Preferences Dialog


Repository Workshop Preferences

Columns: 0 Lines: 0

Position: System Default X,Y: 0 0

Filter: All Plans

View: [✓ Toolbar] [✓ Writeable Icon] [✓ Kind Icon] [✓ Status Line]

Sort by: [✓ Name] Kind

[✓ Save Commits to Central] Save Before Running

Importing Project: Merges Overwrites [✓ Prompts]

[✓ Compile All Plans Generates Models]

The preferences you can set for the Repository Workshop fall into the following general categories:

- workshop window size and position
- filter
- viewing preferences
- sorting preference
- saving preferences
- importing project preference
- font preference

The workshop window size and position, filter, viewing, and font preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Repository Workshop.

NOTE	If you have iPlanet UDS add-on products installed, such as iPlanet UDS Express or iPlanet Integration Server, you may have additional preferences available in the Repository Workshop Preferences dialog. For information about these preferences, see the documentation on the particular add-on product.

Sorting Preference

The sorting preference let you specify the default sorting used for the Plan browser. The following table describes the sorting options:

Preference	Description
Name	Sorts the plans by name.
Kind	Sorts the plans by kind.

Importing Project Preference

The Importing Project preference allows you to set the default behavior for the Plan > Import command when the project being imported already exists in the workspace. By default, the Importing Project preference is set to Prompts, which prompts you with the Import Plan dialog described under "Importing a Plan" on page 184.

The Merges option turns off prompting for the Import command and automatically merges the imported project with the existing project. The Overwrites option turns off prompting for the Import command and automatically overwrites the existing project with the imported project. See "Importing a Plan" on page 184 for information about overwriting and merging.

Saving Preferences

There are two saving preferences in the Repository Workshop Preferences dialog: Save Commits to Central and Save Before Running. Both these saving preferences take effect for all the iPlanet UDS workshops, not just the Repository Workshop.

Saving Commits to Central

By default, when you give a Save All command in an attached shadow, your changes are automatically committed to the central repository. However, if you wish to improve repository performance when you are working with attached shadows, you can set the Save Commits to Central toggle to off so that automatic committing to the central repository is turned off. When automatic committing is turned off, changes you make in the workspace are not made in the central repository until you give a Commit to Central command or until you leave the iPlanet UDS Workshops.

The advantage to turning off automatic committing is that the central repository can handle a larger number of users or performance improves for the current number of users. The disadvantage to turning off automatic committing is that you can lose the changes you make in your workspace. Therefore, we recommend you only turn off automatic committing when increasing repository performance is critical.

Saving Before Running

Normally, changes to your workspace are saved when you give the Save All command. The Save All command writes all changes made to anything in the current workspace since the last Save All command permanently to disk. By setting the Save Before Running toggle to on, you can request that your workspace be automatically saved when you give a Run command. This ensures that you will not lose your work if a problem occurs while you are running the application.

Chapter 4

Using the Project Workshop

This chapter provides conceptual information about TOOL projects and their components, and describes how to use the Project Workshop.

In this chapter, you will learn how to:

- examine TOOL projects and external projects
- create a TOOL project
- modify a TOOL project
- checkout and branch project components
- test a TOOL project
- export and import classes
- set Project Workshop preferences

About Projects

A project is an application definition independent of the environments in which it will run. A project can also be a library, providing definitions for use by other projects.

There are three kinds of projects in iPlanet UDS:

Client project A client project defines a client application. The client project defines a client partition, which usually provides the user interface, and one or more server partitions. A client project also defines a start class and method. You can run a client project using any of the commands on the Run menu.

Server project A server project defines a shared service that you wish to deploy for use by one or more client applications. The server project does not include a client partition or start class and method. Because a server project does not run as an independent application, you can run the project only with the Test Code Fragment command.

Library project A library project provides a set of definitions that you wish to deploy as a shared library. There are no partitions in a library, and you cannot run it.

When you create a project, you do not need to specify its type. It is only when you are ready to partition the project that you need to specify its type by using the Configure As command. See "Using the Configure as Command" on page 678 for information about the Configure As command.

Projects consist of:

- classes
- interfaces
- service objects
- project constants
- cursors
- start class and method (client projects only)
- supplier plans
- project properties

Each of these components is described in detail below.

Classes

The most important components of a project are its classes. The classes in the project define the objects that make up the application. For example, each window in the user interface is defined by a class. The classes also provide the methods that control the flow of the application. For example, the Display method for a user window class displays the window to the end user and provides the code that is executed when the user interacts with the window.

The classes in a project come from three sources:

iPlanet UDS library classes These classes are automatically accessible to every project. The iPlanet UDS library classes provide the services for building the user interface, manipulating data, and interacting with external services.

Custom supplier classes These are classes that were created by developers in your organization specifically for use in building other projects. To use supplier classes in your project, you designate the projects or libraries to which they belong as “suppliers” for your project. Any classes in your supplier projects and libraries are available to you for use in writing your methods.

Custom classes These are classes that you create specifically for your project. Custom classes are always subclasses of selected iPlanet UDS classes or other custom classes. For example, to design a new window for your application, you create a subclass of the iPlanet UDS UserWindow class.

Structuring your application means deciding which classes to include in your project. Some classes represent data that the application manipulates, such as a “painting” in the Art Auction application. Other classes represent services that the application uses, such as an image service.

Interfaces

An interface defines a set of class elements, without providing the code that implements them. The interface provides the method and event handler signatures that define a standard “interface” to an object. The code for the methods and event handlers in the interface is provided by the classes that *implement* the interface.

Implementing an interface in a class means providing the code for all methods and event handlers defined in the interface. Any number of classes can implement a single interface, which provides multiple implementations for a single interface. In addition, a single class can implement multiple interfaces.

You can use an interface as data type for any data item. The interface is the data item’s *declared type*. However, when you create the actual object associated with the data item, the object’s runtime type must be one of the classes that implement the interface. In other words, the implementing class is the data item’s *runtime type*.

iPlanet UDS interfaces allow you to take advantage of two features: dynamic class loading and multiple inheritance. See Chapter 6, "Using the Interface Workshop," for complete information about interfaces.

Service Objects

As described under "About Service Objects" on page 71, service objects provide the basis for creating a distributed application. All of the distributed services with which your application interacts are represented by service objects. For example, to access a database from your application, you must create a service object to represent that database. Likewise, to access an existing external application from your iPlanet UDS application, you can create a service object to represent the external application.

A service object is a named object that represents an existing external resource, an iPlanet UDS shared business service that is going to be shared by multiple users, or a service that you wish to replicate to provide failover or load balancing. The service object contains information needed by the service as well as operations that the service can perform.

When you create a service object, you name it so that it is accessible throughout the distributed application. Any other services in the application can interact with the service object in the same way that any two objects can interact, by invoking methods and posting events.

In the Project Workshop, you need to declare all the service objects to be included in the project. Later on, when you partition the application in the Partition Workshop, iPlanet UDS assigns the service objects in the main project and all its supplier projects to particular nodes in the environment.

There are three kinds of service objects. The class you specify for the service object determines which kind it is:

Service Object	Class	Description
DBResourceMgr	DBResourceMgr	Represents a DBMS installation.
DBSession	DBSession	Represents a database session for a particular database resource manager.
TOOL Class	Any custom class (including C classes) or any applicable iPlanet UDS class	Represents a user-defined service.

The individual properties for a service object depend on which kind of service object it is. The *iPlanet UDS Programming Guide* provides conceptual information about service objects and details about the service object properties. *Accessing Databases* provides complete information on DBResourceMgr and DBSession service objects.

Briefly, the service object properties are:

Visibility By default, a service object is shared between all the users and services in the environment. There is only one copy of the service object in the application, and any changes made to that service object are visible to any user or service in the application that accesses the service object. However, you have the option of limiting this sharing by setting the visibility to "User," which creates a private copy of the service object for each user of the service object.

Dialog Duration The dialog duration for a service object specifies the interval over which the service object retains its connection with a particular caller after the first reference to the partition starts the connection.

Specifying a particular dialog duration ensures that any caller that makes requests from the service is bound to that particular service object for the specified interval. Dialog duration also reflects the period of time that state information is stored in the service object and affects the error handling techniques you use for the service object.

Failover Failover means providing backup service objects to be used if the primary service object fails. Failover provides built-in fault tolerance for the application.

Load Balancing Load balancing means using multiple replicates of the service object to provide simultaneous access for several clients at once. iPlanet UDS automatically coordinates the connections to all the replicates by using a special router partition. When you turn on load balancing, iPlanet UDS creates the router automatically.

Environment Search List When the current environment is connected to other environments, you can use service objects from other environments within the current configuration. You can share an existing service in a connected environment rather than starting that same service in the current environment. Or, you can provide a list of service objects in connected environments to be used for failover for a service object in the current environment.

The environment search path for a service object simply specifies a list of connected environments. iPlanet UDS uses *all* service objects it finds in the path, in the order in which they were specified.

Database Manager Name For DBResourceMgr and DBSession service objects, you must specify the database manager name by choosing the appropriate external manager name from the list of those that were previously defined in your environment.

Database For a DBSession service object, you have the option of specifying the name of the database for which you wish to start a session. If you do not specify the name at this time, you can do so when you partition the project in the Partition Workshop.

User Name and User Password For a DBSession service object, you have the option of specifying the user name and password for the session. If you do not specify them at this time, you can do so when you partition the project in the Partition Workshop.

Attribute Values For a TOOL service object, you have the option of specifying initial values for the simple public attributes.

Constants

A constant is a literal string or numeric value that has a name. When you declare the named constant, you specify a constant name and a value. You can then use the name in place of the value in your TOOL code.

Project constant In the Project Workshop, you can define a constant as part of a project. This means that any code in any of project's classes can reference the constant.

Class constant iPlanet UDS also allows you to define a constant as part of a class using the Class Workshop. When a constant is defined as part of a class (rather than a project), only the class that defines the constant can reference it directly by name. To reference a class-level constant from any other class than the one that defines it, you must qualify the constant name by including the class name.

Local constant In addition to the project and class constants, iPlanet UDS allows you to declare local constants within your TOOL code. Like a local variable, a local constant is available only within the statement block that declares it (see the *TOOL Reference Guide*).

CAUTION Note that although you can use constants to specify values in your TOOL code, you cannot use them to specify values in dialogs in the iPlanet UDS Workshops. For example, you cannot use a constant to specify the default value for a parameter in the Method Workshop.

Cursors

If you are planning to interact with a database, you may wish to define one or more cursors for your project. A cursor is a row marker that you use to work with a set of rows from a database. For example, working with a cursor, you can use the **for** statement to loop through each of the rows in the result set of a **select** statement. Because cursors are associated with the project as a whole, any code in any of the project's classes can reference them.

See "About Cursors" on page 613 for further background information about cursors.

Start Class and Method

For end users to start the application, there must be an entry point. For an iPlanet UDS application, you create an entry point by specifying a start class and a start method. iPlanet UDS begins execution of the application by creating an object of the start class and then invoking the start method on it.

Start class The start class specifies the object on which the start method will operate. iPlanet UDS constructs a new object of the start class by creating an empty object and invoking the Init method on it.

Init method Every class has an Init method that it inherits from the iPlanet UDS Object class. The Init method is provided so that you can override it in your own class and use it to initialize the object. Every time you construct an object of a given class, iPlanet UDS automatically invokes the Init method on it.

Frequently the start class for a project is a UserWindow subclass. Creating a UserWindow subclass provides the window that will be displayed to the user by the start method.

Start method The start method is the method that will be invoked on the start object. You can specify any method defined for the class as the start method.

When the start class for the project is a UserWindow subclass, the start method is usually the Display method, which initializes the data for the window, opens the window, and handles the window's events.

Supplier Plans

A *supplier plan* is an existing project or library that you can include as part of your main project. Everything in the supplier project, including classes, interfaces, service objects, constants, and cursors, becomes accessible to your current project. Supplier plans can be TOOL projects, external projects (for example, C projects), or libraries. If your project needs to access definitions or services defined in another project or library, you must add that project to your list of suppliers. One project or library can be a supplier to any number of other projects.

The *main project* is the project that you create to define the overall logic of an application or service. The suppliers contain components that you need to use in the main project. Together, the main project and its suppliers create the application that you partition using the Partition Workshop. See "About Distributed Applications" on page 651 for information about the relationship between applications, projects, and partitions.

Restricted availability You can include any project in your workspace as a supplier. However, if any of the projects you include are defined as having "Restricted Availability," you may need to define your own project as being restricted. See "Project Properties" on page 210 for information.

Libraries as suppliers You can include any library in your workspace as a supplier. However, before you can use a library as a supplier, it must be installed in the development environment, and you must import it into the repository where you wish to use it. Then, you must include the library in your workspace. See the *iPlanet UDS Programming Guide* for information about installing and importing libraries.

iPlanet UDS libraries Three iPlanet UDS libraries are automatically included in every project. These libraries provide the iPlanet UDS classes that are the basis for building your applications.These three iPlanet UDS libraries are:

iPlanet UDS Library	Description
DisplayProject	Classes for creating windows and the widgets on them.Usually called "the Display library."
Framework	Foundation classes for building applications.
GenericDBMS	Classes for accessing a DBMS.

To use iPlanet UDS, you must include the Framework library. The Display and GenericDBMS libraries are optional, and if it is appropriate, you can exclude them to increase efficiency. However, if your project has a user interface, you need to include the Display library. And if your project interacts with any database management system, you need to include the GenericDBMS library.

Optional iPlanet UDS libraries iPlanet UDS also provides several optional libraries that you should include when you need them. For example, if you are going to use Microsoft's OLE 2 in your project, you must include the OLE library as a supplier for the main project. You can see a list of the optional iPlanet UDS libraries in the browser in the Repository Workshop.

Supplier plans for libraries Because libraries are static, classes within a library cannot inherit from a TOOL class in a project that could be modified. Therefore, all supplier plans for libraries must be libraries. You should not use a project as a supplier plan for a library.

Project Properties

Projects have the following properties:

Property	Description
Project type	A read-only property that indicates whether the project was configured as a client, server, or library.
Restricted availability	If this property is turned on, the TOOL project is restricted. You may need to turn this property on only when the main project includes a supplier project that is defined as "restricted."
Compatibility level	An integer that indicates the current compatibility level of the application, server, or library.
Library Name	A name that is used when the project is configured as a library or included within a library distribution. When there is more than one library within a library configuration, all the library names must be unique. The library name can be any length, however, on platforms where there is an eight-character limit for file names, the library name will be truncated to eight characters.

The following sections provide a general discussion of these properties.

Project Type

As described under "About Projects" on page 201, there are three kinds of projects. The project type property for the project is a read-only property that shows the project kind: Client, Server, or Library.

The default project type is Client. If you have not yet given a Configure As or Partition command for your project, the project's type will be Client.

Setting the Project Type

When you are ready to create the configuration for your project, you must choose whether to deploy it as a client, server, or library. If you give the Configure As Server command to create a server configuration, this sets the project type to Server. Likewise, if you give the Configure As Library command to create a library configuration, this sets the project type to Library. You can change the project type at any time by giving a different Configure As command for it.

You can also change the project type using the Run > Partition command. The Partition command lets you run a client project using its distributed configuration, and is intended for client projects only. However, if you give the Partition command for a server or library project, iPlanet UDS prompts you to change the project type to Client.

See Chapter 14, "Using the Partition Workshop," for information about the Configure As and Partition commands.

Compatibility Level

The first time you create a new project, you do not need to be concerned about its compatibility level. This feature is relevant only after you have already released the application, server, or library and now want to create a new release. For detailed information about upgrading applications, see the *iPlanet UDS Programming Guide*.

Normally if you plan to create a new release of your application, you should raise its compatibility level. Incrementing the compatibility level enables you to install and run the new release of the application in the same environments where older releases of the application are installed.

However, it is not always necessary to increase the compatibility level of your project. If you want to update an individual partition so that it is still compatible with currently installed partitions, you can keep the same compatibility level. In this case, the existing clients will be able to use the new server, or vice versa. For example, you could update the AuctionMgr service for the Art Auction without affecting the project's client partition. See the *iPlanet UDS Programming Guide* for information.

Compatibility level for libraries A library distribution has a compatibility level based on the compatibility level of the project for which you give the Configure As Library command. Like an application's compatibility level, the library distribution's compatibility level allows you to deploy different releases of the same library within a single environment. The rules for when you need to raise the compatibility level of a library distribution are the same as those for an application. See the *iPlanet UDS Programming Guide* for information.

Restricted Availability

The Restricted Availability property is intended for use when you are integrating your iPlanet UDS application with certain C routines, as described in *Integrating with External Systems*. If your TOOL project does not include a restricted supplier project—either a C project or another TOOL project— you can simply ignore this property.

A C project is defined as having restricted availability when it can run only on particular hardware or software. For example, if a C project encapsulates a C electronic mail routine that calls out to an API, the C project may be restricted because the API is available only on certain nodes in the environment. If an unrestricted TOOL project includes any restricted projects (either C or TOOL), you can use the service objects provided by those projects but you cannot create objects using their classes. This is because the necessary hardware or software required by the restricted project may not be available.

If you declare your TOOL project as restricted, all of its classes become restricted and can thus create instances of restricted classes. However, such a declaration has serious repercussions on how you can partition the project—the restricted TOOL project will be limited to those nodes in the environment that can support its restricted supplier projects.

NOTE In many cases, it is sufficient to restrict an individual class, rather than the entire project (see "Restricted Property" on page 282 for information). When you restrict an individual class, the method code for the restricted class can create objects from classes in the restricted project, while the "unrestricted" classes cannot. Restricting an individual class rather than a project provides more flexibility when you are partitioning the project.

Writeable and Read Only Components

If you have opened your workspace for modifying, the project components are in one of two states:

Writeable When a component is writeable, you can examine it or modify it. A component is writeable when:

- The component is *new*. A component is new if you created the component after the last time you integrated.
- The component is *checked out*. A component is checked out if you used a Checkout command to check out the component.
- The component is *branched*. A component is branched if you have used a Branch command to branch the component.

Read Only When a component is read only, you can examine it but not modify it. A component is read only when:

- The component exists in the system baseline, but you have not yet checked it out. In this case, you can check out the component to get write access to it.
- Another developer has checked out the component. In this case, the only way to get write access to the component is to branch it.
- The component was *generated* by iPlanet UDS Express or iPlanet UDS iPlanet Integration Server. You cannot modify project components that were created by Express or iPlanet Integration Server from the Project Workshop.

Writeable icons Note that the writeable icons in the Project Component browser indicate whether the component is new, checked out, branched, or generated. See "Examining the Components" on page 218 for information.

See "Write Access to Project Components" on page 257 for information about checking out and branching components.

If you have opened your workspace for reading only, then all the project components are read only, and you cannot get write access to them.

Using the Project Workshop

You enter the Project Workshop from the Repository Workshop either by opening an existing project or library, or by creating a new project.

➤ **To open an existing project or library**

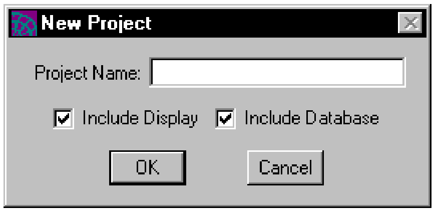
1. Double-click the project or library name, or select the project name or library and choose the Plan > Open command.

   iPlanet UDS opens the selected plan in the Project Workshop.

➤ **To create a new project**

1. Choose the Plan > New Project command.

   The New Project dialog opens.



New Project

Project Name:

[✓] Include Display [✓] Include Database

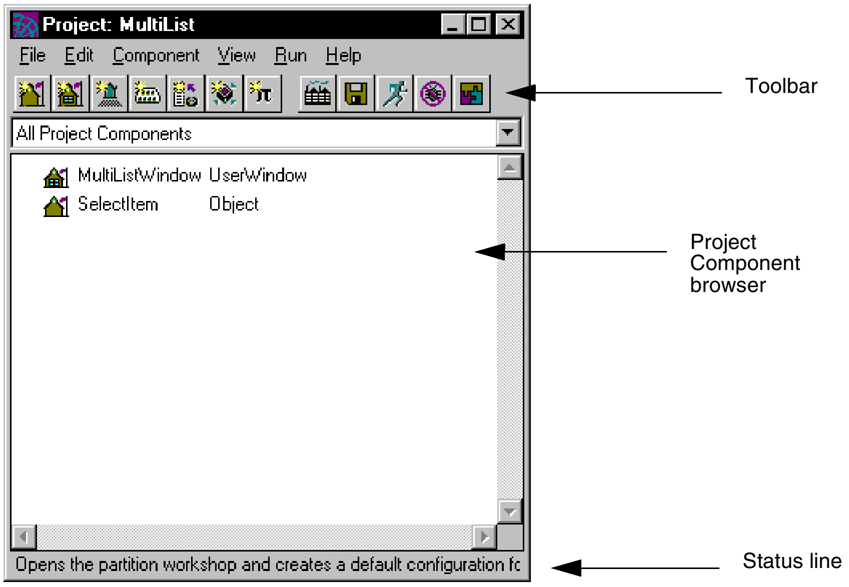
2. In the New Project dialog, enter the project name and click OK.

   The Project Workshop opens.

The Project Workshop Window

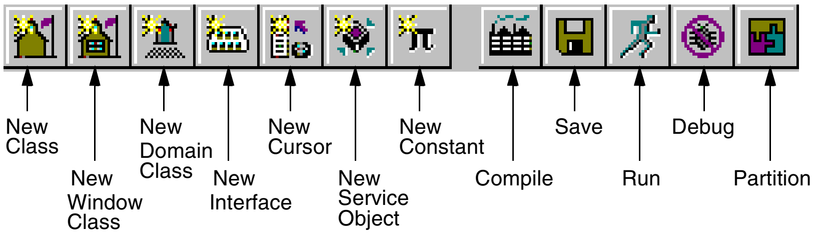
The Project Workshop window, shown in Figure 4-1, consists of three parts: the Project Components browser, the status line, and the toolbar, shown in Figure 4-2.

Figure 4-1 The Project Workshop


Project: MultiList
File Edit Component View Run Help
All Project Components
MultiListWindow UserWindow
SelectItem Object
Toolbar
Project Component browser
Status line

Opens the partition workshop and creates a default configuration fc

Figure 4-2 The Project Workshop Toolbar



View Menu

The View menu in the Project Workshop provides the following toggles that let you control which parts of the workshop are displayed:

Command	Function
Toolbar	Makes the toolbar visible or invisible.
Writeable Icon	Makes the writeable icons visible or invisible.
Kind Icon	Makes the component type icons visible or invisible.
Status Line	Makes the status line visible or invisible.

Note that you can set your viewing preferences for the workshop by using the Workshop Preferences command. These preferences are saved as part of your current workspace. See "Setting Workshop Preferences" on page 272 for information.

Access to Other Workshops

From the Project Workshop, you can access the following workshops:

Button	Workshop	How to access it
	Class Workshop	To create a new class, click the toolbar New Class button, or choose the Component > New > Nonwindow Class command. To open an existing class, double-click the class name or select the class name and choose the Component > Open… command.
	Window Workshop	To create a new window class, click the toolbar New Window Class button, or choose the Component > New > Window Class command.
	Domain Workshop	To create a new domain class, click the toolbar New Domain Class button, or choose the Component > New > Domain Class command.
	Interface Workshop	To create a new interface, click the toolbar New Interface button, or choose the Component > New > Interface command.

Button	Workshop	How to access it
	Cursor Workshop	To create a new cursor, click the Cursor Workshop button in the toolbar, or choose the Component > New > Cursor command. To open an existing cursor, double-click the cursor name in the component list, or select the class name and choose the Component > Open… command.
	Partition Workshop	Click the Partition Workshop button in the toolbar, or choose the Run > Partition… command.
	Debugger	Click the Debugger button in the toolbar, or choose the Run > Test Debug command.

Leaving the Project Workshop

To leave the Project Workshop, use the File > Close command to close the Project Workshop. You can also use the system close box to close the window. Only the current Project Workshop will be closed.

Examining a Project or Library

The Project Workshop allows you to examine any project or library in your workspace. To examine a project or library, you must start from the Repository Workshop.

➤ **To open a project or library**

1. In the Repository Workshop, double-click the project or library name.

or

1. In the Repository Workshop, select the project or library name.
2. Choose the Plan > Open… command.

The Project Workshop displays information about classes, interfaces, cursors, service objects and constants directly on the main workshop window. The supplier plans, start class and method, and project properties are not displayed on the main workshop window. To view them, you must use the appropriate commands. The following sections provide detailed information about how to examine each of the project or library components.

The Project Workshop also provides you with a history of any component. This history shows you the checkout status of the component and provides a history of the integrations performed on the component.

You can examine a library in the Project Workshop the same way you examine a project. If the library was created with source code included, you will be able to view the source code for the library's methods, event handlers, and cursors. Otherwise, you will be able to view only the definitions of the methods, event handlers, and cursors, without the source code.

Note that all libraries are read only. You cannot modify the source code for a library from the iPlanet UDS Workshops.

Examining the Components

By default, the Project Workshop displays all project or library components. A kind icon by each component name indicates the kind of component.

Icon	**Project Component**
	Nonwindow class
	Window class
	Domain class
	Interface
	Service object
	Cursor
π	Constant

To turn off the kind icons, switch off the View > Kind Icon toggle.

Filter drop list To view a list of only one kind of component, use the filter drop list for the Project Workshop. The filter drop list allows you to choose a single component kind, such as "Window Class" or "Service Object," or various combinations, such as "All Project Components" or "Nonconstant Components."

Note that you can set your filter preferences for the workshop by using the Workshop Preferences command. These preferences are saved as part of your current workspace. See "Filters" on page 137 for information.

Writeable icons iPlanet UDS also displays a writeable icon by each component name. The writeable icons indicate whether the current workspace has write access to the component, and shows whether it is new, checked out, or branched. A blank indicates that the component is unavailable. The icons are:

Icon	Plan
	New
✓	Checked out
|	Branched
⇨	Generated by iPlanet UDS Express or iPlanet Integration Server

To turn off the writeable icons, switch off the View > Writeable Icon toggle.

Sorting by name or kind By default, the list of components is in alphabetical order, sorting all components by name. If you wish to view the same list sorted by component type, choose the View > Sort by Kind command. The Sort by Name command specifies the default, sorting all components by name.

Component history The Project Workshop lets you display the following history information about any project component:

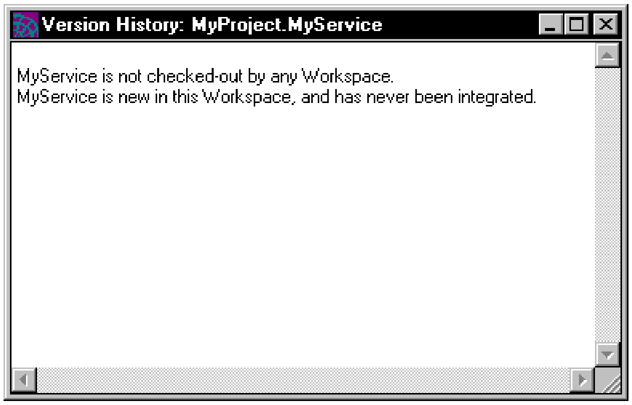
- integration history
- version in the current workspace
- name of the workspace that has it checked out (if it is checked out)

To display the component history, you use the Component > Show History... command and specify how many previous integrations you wish to view.

➤ To display the component history

1. Select the component name from the Project Components browser.
2. Choose the Component > Show History… command.
3. In the Show Component History dialog, specify how many previous integrations you wish to view and whether you wish to view the full or abbreviated history.
4. Click the OK button to view the component history.

   The Component History window opens.



```
MyService is not checked-out by any Workspace.
MyService is new in this Workspace, and has never been integrated.
```

5. When you are finished viewing the component history, simply close the Component History window.

For further details about the information provided by the Show History… command, see the Fscript ShowCompHistory command in *Fscript Reference Guide*.

The following sections describe how to examine the individual components.

Examining a Class

To view a list of the standard classes in the current project or library, excluding the other components, choose "Nonwindow Classes" from the drop list in the Project Components browser. The browser displays the names of all the standard classes.

To display the complete class definition, double-click the class name, or select the class name and choose the Component > Open… command. This command opens the Class Workshop, where you can examine the original class definition. See "Examining a Class" on page 296 for information about using the Class Workshop to examine a class.

If the class you wish to examine is not displayed in the Project Components browser, you can use the Find Class/Interface… command to search for the specified class in the current project and all its supplier plans.

➤ **To search for a class**

1. Choose the Edit > Find Class/Interface… command.
2. In the Find Class/Interface dialog, specify the class name and click the OK button.

Examining a Window Class

To view a list of window classes in the current project or library, excluding the other components, choose "Window Classes" from the drop list in the Project Components browser. The browser displays the names of all the window classes.

To display the complete class definition, double-click the class name, or select the class name and choose the Component > Open... command. This command opens the Class Workshop, where you can examine the original class definition. See "Examining a Class" on page 296 for information about using the Class Workshop to examine a class.

Examining a Domain Class

To view a list of the domain classes in the current project or library, excluding the other components, choose “Domain Classes” from the drop list in the Project Components browser. The browser displays the names of all the domain classes.

To display the complete domain class definition, double-click the domain name, or select the domain name and choose the Component > Open… command. This command opens the Class Workshop, where you can examine the original domain class definition. See “Examining a Class” on page 296 for information about using the Class Workshop to examine a class.

Examining an Interface

To view a list of the interfaces in the current project or library, excluding the other components, choose “Interfaces” from the drop list in the Project Components browser The browser displays the names of all the interfaces.

To display the complete interface definition, double-click the interface name, or select the interface name and choose the Component > Open... command. This command opens the Interface Workshop, where you can examine the original interface definition. See “Examining an Interface” on page 341 for information about using the Interface Workshop to examine an interface.

If the interface you wish to examine is not displayed in the Project Components browser, you can use the Find Class/Interface… command to search for the specified interface in the current project and all its supplier plans.

➤ **To search for a interface**

1. Choose the Edit > Find Class/Interface… command.
2. In the Find Class/Interface dialog, specify the interface name and click the OK button.

Examining a Cursor

To view a list of cursors in the current project or library, excluding the other components, choose “Cursors” from the drop list in the Project Components browser. The browser displays the names of all the cursors.

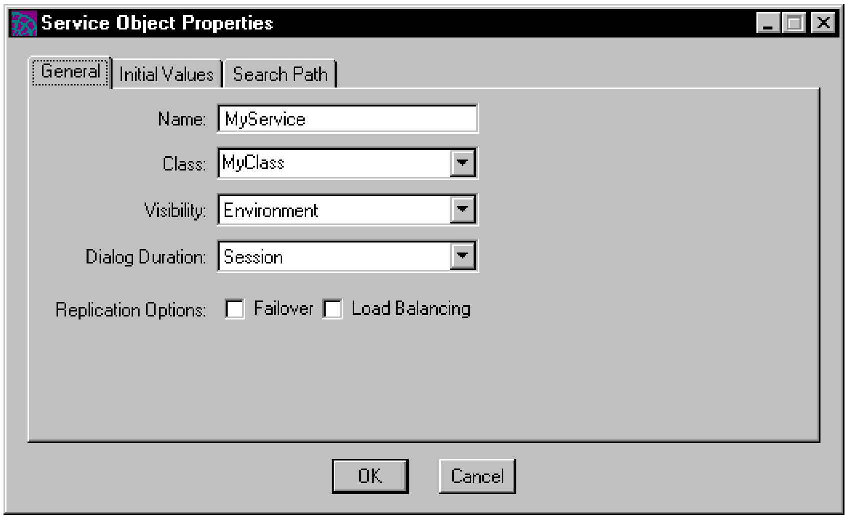
To display the complete definition of an individual cursor, double-click on the cursor name, or select the cursor name and choose the Component > Open... command. This command opens the Cursor Workshop, where you can view the original cursor definition. See Chapter 12, “Using the Cursor Workshop,” for information on the Cursor Workshop.

Examining a Service Object

To view a list of service objects in the project or library, excluding the other components, choose “Service Objects” from the drop list in the Project Components browser. The browser displays the names of all the service objects.

To display the complete definition of an individual service object, double-click on the service object name, or select the service object name and choose the Component > Open... command. This command opens the Service Object Properties dialog, shown below, where you can view the original definition.

Figure 4-3 Service Object Properties Dialog



Name: MyService

Class: MyClass

Visibility: Environment

Dialog Duration: Session

Replication Options: ☐ Failover ☐ Load Balancing

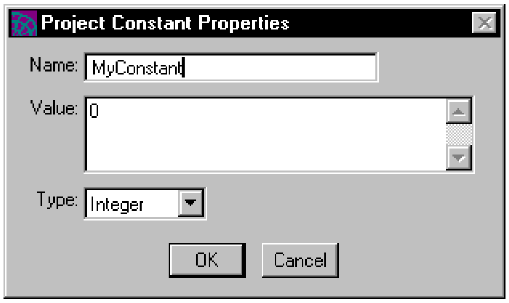
See "Defining Service Objects" on page 234 for information about the Service Object Properties dialog.

Examining a Constant

To view a list of constants in the project, excluding the other project components, choose "Constants" from the drop list in the Project Components browser. The browser displays the names and values of all the constants.

To display the complete definition of an individual constant, double-click the constant name, or select the constant and choose the Component > Open... command. This command opens the Project Constant Properties dialog, shown in Figure 4-4, where you can view the original definition.

Figure 4-4 Project Constant Properties Dialog

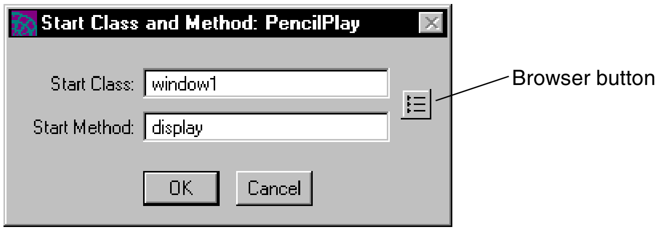


Name: MyConstant
Value: 0
Type: Integer

Examining Start Class and Method

To view the start class and method for the project, choose the File > Start Class Method command. This command opens the Start Class Method dialog, shown in Figure 4-5, where you can view the current settings.

Figure 4-5 Start Class and Method Dialog

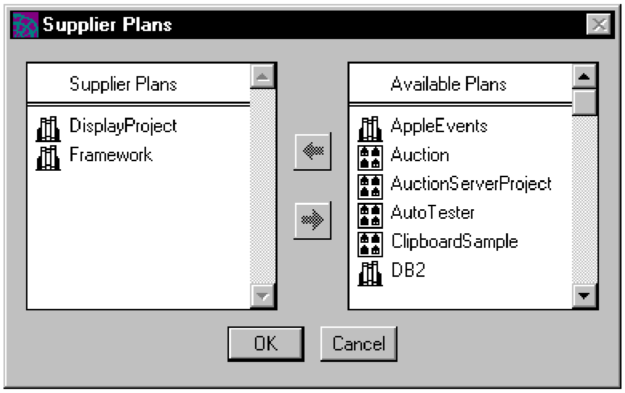


Start Class: window1
Start Method: display

Examining Supplier Plans

To view the supplier plans for the project, choose the File > Supplier Plans… command. This command opens the Supplier Plans dialog, shown in Figure 4-6, which displays the list of projects and libraries included in the current project.

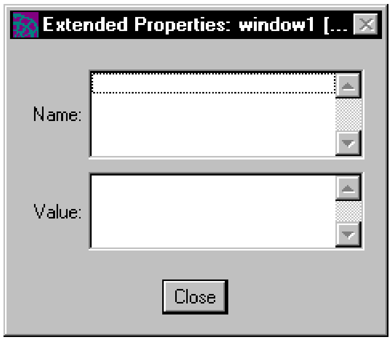
Figure 4-6 Supplier Plans Dialog



Examining Extended Properties

To view the extended properties for a project component, select the component name and choose the Component > Extended Properties... command.This command opens the Extended Properties dialog, where you can view a list of the extended properties set for the component. To view the value for an individual property, select the property name in the Name field; the corresponding value is displayed in the Value field.

Figure 4-7 Extended Properties Dialog



Creating a TOOL Project

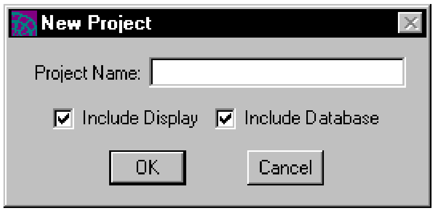
The Project Workshop allows you to create only TOOL projects. To create any kind of external project, such as a C project or a DCE project, you must write the project definition in a file and use Fscript to load the definition. See *Integrating with External Systems* for information.

To create a new TOOL project, you must start from the Repository Workshop.

➤ To create a new TOOL project

1. In the Repository Workshop, open your workspace for modifying.
2. In the Repository Workshop, choose the Plan > New Project command.

   The New Project dialog (described under "Using the New Project Command" on page 228) opens.

3. In the New Project dialog, enter the project name. If your project will not contain windows or access a database, click off corresponding Include Display or Include Database toggle.
4. In the Project Workshop, define the project components (as described in this chapter).
5. Use the Save All command as necessary to save the changes to your workspace.

To make the project available to other developers with whom you are sharing the same repository, you must give the Update Workspace command and then test your changes against the system baseline. After the project is tested, use the Integrate Workspace command to add your new project to the system baseline. See "Updating and Integrating a Workspace" on page 175 for information.

➤ To define project components

1. Define the classes, interfaces service objects, cursors, and constants for the project.
2. Specify the supplier plans for the new project.

3. Set the start class and method if appropriate.
4. Set the project properties if necessary.

The following sections describe these steps in complete detail.

Note that you can test your project at any time during development. See "Testing a TOOL Project" on page 262 for information.

Using the New Project Command

The New Project command in the Repository Workshop creates a new project. At this point, you specify the project name and indicate if your project will not include windows or will not access a database.

A project name must be unique for all plans in the repository and for all environments in which the application will run. (You cannot have more than one application with the same name running in the same environment.) Also, iPlanet UDS uses the first six characters of the project name for the application file names. So keep this in mind when you choose the project name.

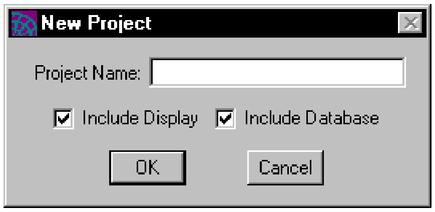
Excluding Display and GenericDBMS Libraries

Normally, when you create a new project, iPlanet UDS automatically includes the iPlanet UDS Display and GenericDBMS libraries as suppliers for your project. However, if you specify that the new project will not contain windows or will not access a database, iPlanet UDS excludes the appropriate project, providing increased efficiency. If you change your mind later and want to include an excluded project, you can use the File > Supplier Plans… command to do so.

Note that you can explicitly control which projects and libraries are included in your project with the Supplier Plans command. See "Specifying Supplier Plans" on page 244 for information.

Figure 4-8 illustrates the New Project dialog:

Figure 4-8 New Project Dialog



When you create a new project, other workspaces will not be able to see the contents of that project until after you give an Integrate Workspace command for your workspace. See "Updating and Integrating a Workspace" on page 175 for information.

Defining Classes

There are four ways to add a new class to your project:

How to add a class	Description
Create a new class	Use the Component > New > Nonwindow Class, Window Class, or Domain class commands. See below for descriptions.
Import a class	Use the Component > Import Class/Interface… command to import the class from a standard text file.
Copy a class from another project using drag and drop	You can copy an entire class definition from another Project Workshop by dragging its name from the other Project Component browser and dropping it into the current project.
Copy a class from another project using Copy or Cut command with the Paste command	You can copy an entire class definition from another Project Workshop by using the Edit > Copy or Cut command menu to copy the class the clipboard and then use the Edit > Paste command to paste the class into the current project.

The following sections describe how to use the Project Workshop to define a new standard class, new window class, and new domain class. See "Importing and Exporting Classes and Interfaces" on page 270 for information on the Import Class/Interface… command.

Creating a Standard (Nonwindow) Class

The Component > New > Nonwindow Class command creates a new class with the name and superclass you specify. When you give the New Nonwindow Class command, iPlanet UDS opens a Class Properties dialog, where you specify the name and superclass for the class. See "Creating a Nonwindow Class" on page 303 for information about setting the class properties.

After you create the class, iPlanet UDS opens the Class Workshop. See Chapter 5, "Using the Class Workshop," for information about using the Class Workshop.

Creating a Window Class

The Component > New > Window Class command creates a new window class with the name you specify. When you give the New Window Class command, iPlanet UDS opens a Class Properties dialog, where you specify the window class name. The superclass "UserWindow" is filled in automatically. You can change this to a UserWindow subclass to create an inherited window. See "Creating a Window Class" on page 304 for information about setting the class properties.

Opening the Window Workshop

After you create the window class, iPlanet UDS opens the Class Workshop. At this point, you can either define the class elements for the window class or open the Window Workshop to create the form for the window. To open the Window Workshop, click the Window Workshop button or choose the File > Window… command in the Class Workshop.

See "Creating a Class" on page 302 for information about defining the class elements. See Chapter 7, "Using the Window Workshop," and Chapter 9, "Using the Menu Workshop," for information about creating the window and menu bar for the window class.

Creating a Domain Class

The Component > New > Domain Class command creates a new class with the name and superclass you specify. When you give the Component > New > Domain Class command, iPlanet UDS opens a Domain Class Properties dialog, where you specify the name, superclass, form widget, and array widget for the class. See "Creating a Domain Class" on page 305 for information about setting the class properties.

After you create the domain class, iPlanet UDS opens the Class Workshop. See Chapter 5, "Using the Class Workshop," for information about using the Class Workshop.

Defining Interfaces

There are four ways to add a new interface to your project:

How to add a interface	Description
Create a new interface	Use the Component > New > Interface... command. See below for a description.
Import an interface	Use the Component > Import Class/Interface… command to import the interface from a standard text file.
Copy an interface from another project using drag and drop	You can copy an entire interface definition from another Project Workshop by dragging its name from the other Project Component browser and dropping it into the current project.
Copy an interface from another project using Copy or Cut command with the Paste command	You can copy an entire interface definition from another Project Workshop by using the Edit > Copy or Cut command menu to copy the interface the clipboard and then use the Edit > Paste command to paste the interface into the current project.

The Component > New > Interface command creates a new interface with the name you specify.

➤ **To create an interface**

1. Click the New Interface button on the toolbar or choose the Component > New > Interface command.
2. In the New Interface dialog, enter the interface name and click OK.

After the Interface Workshop opens, you can define the individual elements in the interface. See "Creating an Interface" on page 347 for information.

Defining Project Constants

There are three ways to add a new constant to your project:

How to add a constant	Description
Create a new constant	Use the Component > New > Constant command. See below for a complete description.
Copy a constant from another project using drag and drop	You can copy an entire constant definition from another Project Workshop by dragging its name from the other Project Component browser and dropping it into the current project.
Copy a constant from another project using the Copy or Cut command with the Paste command	You can copy an entire constant definition from another Project Workshop by using the Edit > Copy or Cut command to copy the constant to the clipboard and then using the Edit > Paste command to paste the constant into the current project.

The Component > New > Constant command creates a new constant with the name, type, and value you specify.

➤ To create a constant

1. Click the New Constant button, or choose the Component > New > Constant command.
2. In the New Constant dialog, specify the name, type, and value for the constant. The constant types are described below.
3. Click the OK button to add the constant to the project or the New button to create another constant.

Constant types The constant types are the following:

Constant Type	Description
Automatic	iPlanet UDS determines the type based on the constant value (see below). Use the Automatic type for strings.
Boolean	Allows a value of TRUE or FALSE.
Double	Allows a floating point number.
Integer	Allows a positive or negative whole number.
String	Allows an alphanumeric string.

Automatic type For an automatic type, the value you specify determines the type of the constant. The types are the following:

Value	Type
TRUE, FALSE	boolean
Positive or negative whole number	integer
Floating point number	float
Character data (without quotes)	string

Single quotes for strings Normally you do not need to enclose a string value in single quotation marks. However, if you wish to create a string constant with a value of an integer or a floating point, you must use single quotes. In addition, because iPlanet UDS automatically truncates trailing spaces, you need to use single quotes to specify a string value with trailing spaces.

Defining Service Objects

There are three ways to add a new service object to your project:

How to add a service object	Description
Create a new service object	Use the Component > New > Service Object command. See below for a complete description.
Copy a service object from another project using drag and drop	You can copy an entire service object definition from another Project Workshop by dragging its name from the other Project Component browser and dropping it into the current project.
Copy a service object from another project using Copy or Cut command with the Paste command	You can copy an entire service object definition from another Project Workshop by using the Edit > Copy or Cut command to copy the service object the clipboard and then using the Edit > Paste command to paste the service object into the current project.

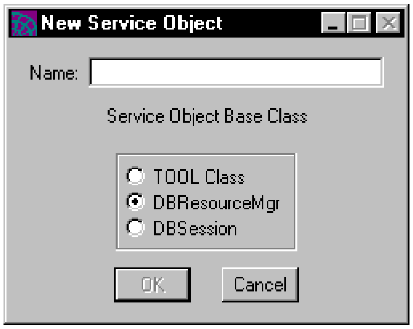
The Component > New > Service Object command creates a new service object. This command opens the New Service Object dialog, where you must specify the name of the service object and its base class.

After you fill in the New Service Object dialog, the Service Object Definition dialog opens, where you enter the full service object definition.

➤ **To create a service object**

1. Click the New Service Object button, or choose the Component > New > Service Object command.

   The New Service Object dialog (described below) opens.



New Service Object

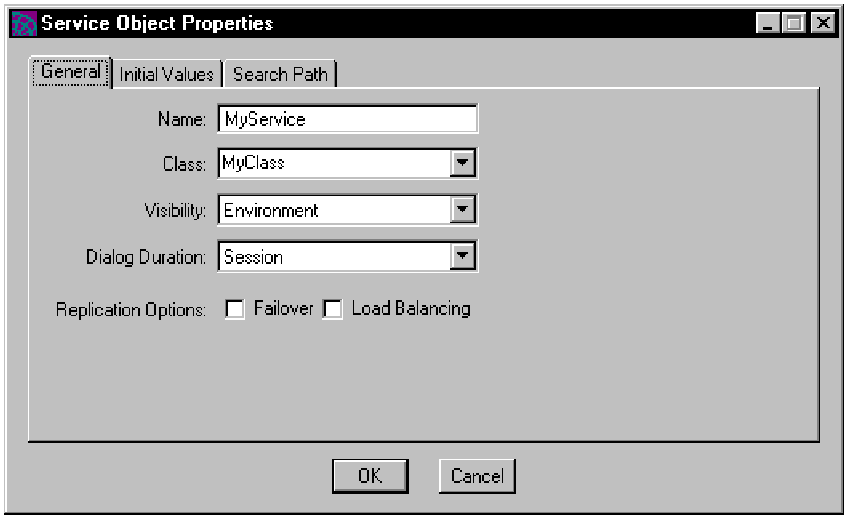
Name:

Service Object Base Class

( ) TOOL Class
(●) DBResourceMgr
( ) DBSession

2. In the New Service Object dialog, enter the service object name and choose the base class of the service object: TOOL class, DBResourceMgr, or DBSession. Click the OK button when finished.

   The Service Object Properties dialog (described below) opens.



Service Object Properties

Name: MyService

Class: MyClass

Visibility: Environment

Dialog Duration: Session

Replication Options: ☐ Failover ☐ Load Balancing

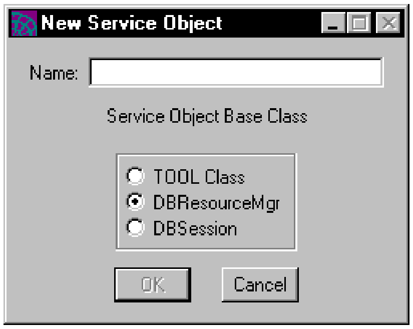
3. In the Service Object Properties dialog, specify the class for the service and set other properties.

4. Click the OK button to add the service object to the project.

New Service Object Dialog

The New Service Object dialog, shown below, allows you to name the service object and to choose the base class. The base class determines the properties that will be available on the Service Object Properties dialog.

Figure 4-9 New Service Object Dialog



Service object name The service object name must be unique for the project. You can use any legal iPlanet UDS name as described under "Using iPlanet UDS Names" on page 131.

Base class for service object The base class for the service object is its general type. The default base class for a service object is TOOL class, which provides a user service object. If you wish to create a DBResourceMgr or DBSession service object, click the appropriate radio button.

Service Object Properties Dialog

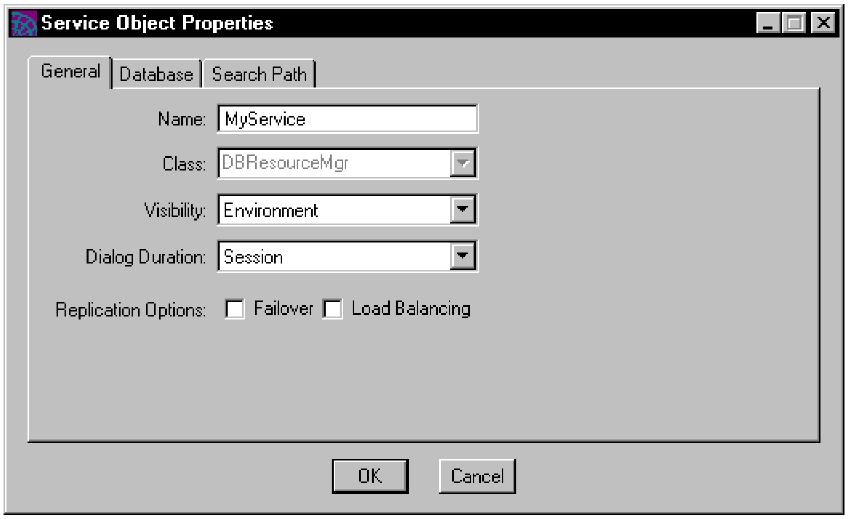
The Service Object Properties dialog contains a tab folder; the individual tab pages that appear in the folder depend on the base class of the service object.

Base Class for Service Object	Tab Pages in Service Object Properties Dialog
TOOL Class	General, Initial Values, Search Path.
DBResourceMgr	General, Database, Search Path.
DBSession	General, Database, Search Path, Connection.

The following sections describe the options available on each of these tab pages.

General Page The General tab page contains the options that apply to all service objects.

Figure 4-10 General Tab Page



Name: MyService
Class: DBResourceMgr
Visibility: Environment
Dialog Duration: Session
Replication Options: ☐ Failover ☐ Load Balancing

Class property For a TOOL class service object, you must specify the class for the service object. The service class can be any custom class, or it can be any iPlanet UDS class that is allowed for service objects. (The iPlanet UDS documentation on individual classes indicates whether a class can be used for a service object.) The drop list for this field displays the classes in the current project. To enter a class that is not on the list, type its name into the fillin field.

The class for a service object must be defined with the Distributed object runtime property set to "Allowed." At runtime, when iPlanet UDS creates the service object, it automatically sets the IsAnchored attribute to TRUE. See "Distributed Property" on page 286 for information about the Distributed object runtime property.

For a DBResourceMgr service object, the service class is DBResourceMgr, and you cannot change it. For a DBSession service object, the service class is DBSession, and you cannot change it.

Visibility property For TOOL class and DBSession objects, the default visibility is Environment. For DBResourceMgr service objects, the default is User. Choose whichever is appropriate for your particular service. See *iPlanet UDS Programming Guide* for information on visibility.

Dialog Duration property You must choose the dialog duration that the creator of the class has implemented for the service. For DBResourceMgr service objects, the dialog duration must be Session. For DBSession service objects, the default dialog duration is Transaction, and the value can only be Session or Transaction. See *iPlanet UDS Programming Guide* for information on dialog duration.

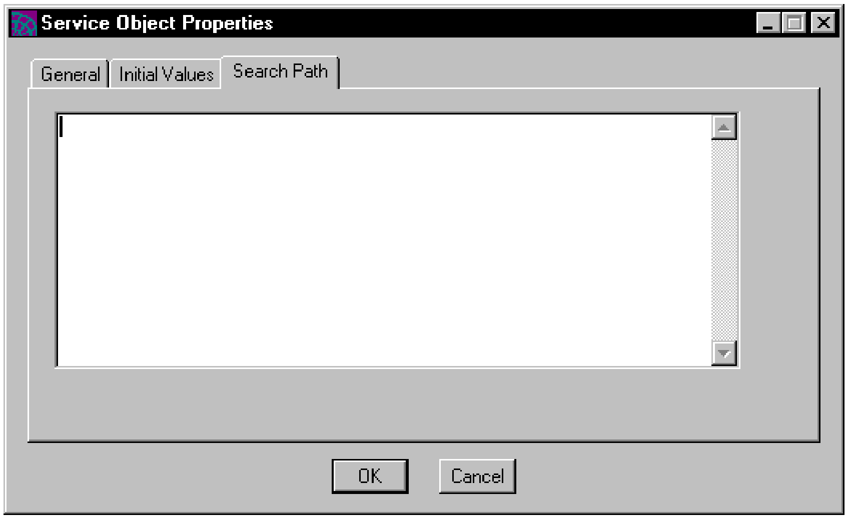
Failover property To provide failover services for your service object, click the Failover toggle to on. This property allows you to replicate the partition that contains the service object. See *iPlanet UDS Programming Guide* for information on failover.

Load Balancing property To provide load balancing for your service object, set the Load Balancing toggle to on. This property allows you to replicate the partition that contains the service object. See *iPlanet UDS Programming Guide* for information on load balancing.

NOTE iPlanet UDS does not automatically replicate your service objects for load balancing or failover. Instead, in the Partition Workshop, you must replicate the partitions to which the service objects are assigned. You can replicate a partition either by assigning it to an additional node (as described under "Assigning Partitions" on page 699) or by setting the replication count for an individual assigned partition on a single node (described under "Setting Assigned Partition Properties" on page 700).

Search Path property The Search Path tab page allows you to specify a default environment search path for your service object.

Figure 4-11 Search Path Tab Page



In the search path field, enter a string that lists, in the correct order, the environments to be searched. The syntax for the search path string is:

path **[(a)]** **[:** *path* **[(a)]**...

path is:

(@ | @*environment_name*)

A special (a) option allows you to specify that the service object identified by a specific path should automatically be started if necessary.

Environment variables You can use an environment variable to specify an environment name. The value for the environment variable is set on first access to the service object, using the value of the environment variable as set on the service object's partition. The syntax is:

${*environment_variable_name***}**

Be sure to include the braces!

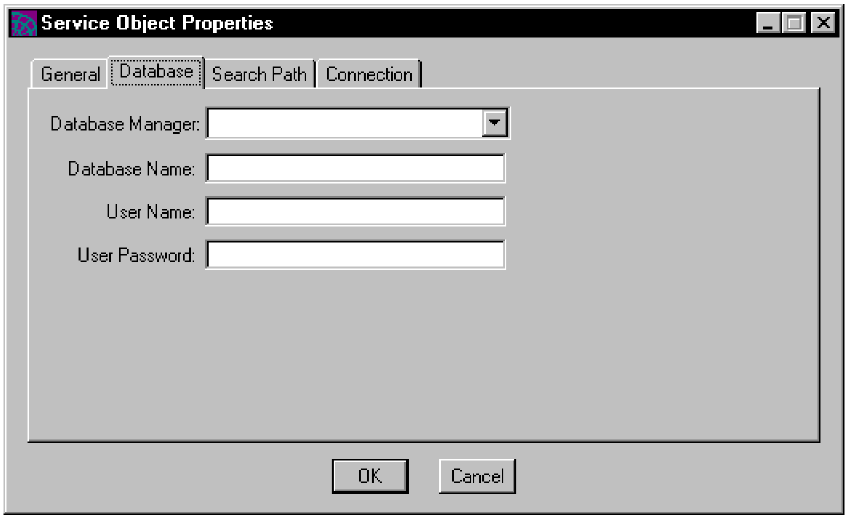
The following example illustrates a search path that looks first in the current environment, second in the "la" environment, and last in the "sf" environment:

```
@:@la:@sf
```

See *iPlanet UDS Programming Guide* for complete information about the service object's environment search path. See "Modifying a Service Object Definition" on page 695 for information about setting the service object's environment search path in the Partition Workshop.

Database Page The Database tab page contains options for the DBResourceMgr and DBSession service objects.

Figure 4-12 Database Tab Page



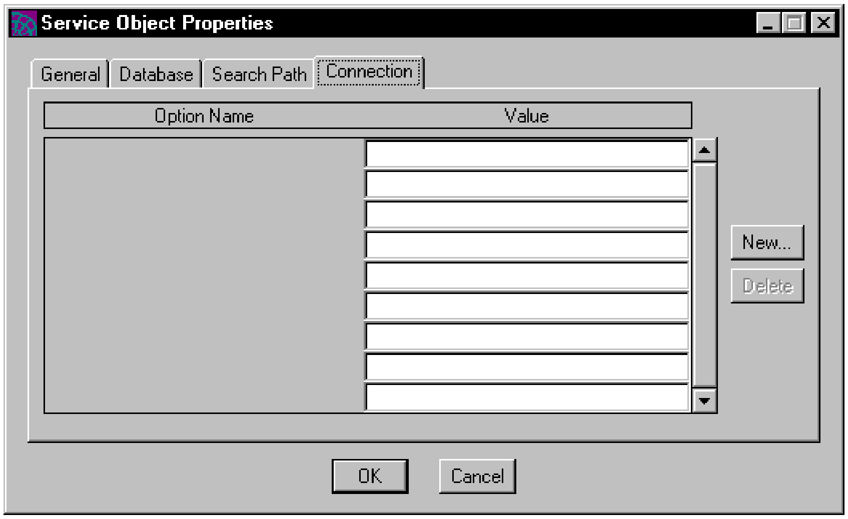
Database Manager property For DBResourceMgr and DBSession service objects, the drop list for the Database Manager field displays all the external manager names that were previously defined in your environment. You must choose a manager name at this time, however, you can override this setting in the Partition Workshop.

Database Name property For a DBSession service object, you have the option of specifying the name of the database for which you wish to start a session. If you do not specify it at this time, you can do so when you partition the project in the Partition Workshop. The syntax for the database name is database management system dependent (see *Accessing Databases* for information).

User Name and user Password properties For a DBSession service object, you have the option of specifying the user name and password for the session. If you do not specify them at this time, you can do so when you partition the project in the Partition Workshop. The syntax for the user name and user password is database management system dependent (see *Accessing Databases* for information).

Connection Page The Connection tab page allows you to specify options that take affect when the connection is made to a database. For example, a common option is `DB_MAX_STATEMENTS`, which controls how many TOOL SQL statements to keep in the cache. See *Accessing Databases* for information about the Connection options.

Figure 4-13 Connection Tab Page



➤ **To specify a new connection option**

1. On the Connection tab page, click the New... button.

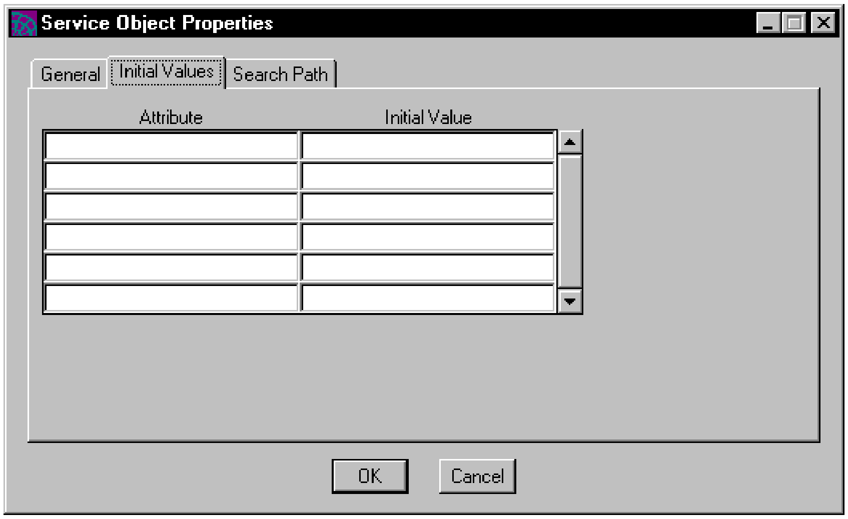
   The New Connection dialog opens.

2. Enter the name of the new connection option and click OK.

3. On the Connection tab page, enter the value for the new connection option and click OK.

Initial Values Page The Initial Values tab page allows you to specify initial values for the simple public attributes of TOOL class service objects.

Figure 4-14 Initial Values Tab Page



The array field on this page displays a list of all the simple public attributes in the service object's class. To specify the initial value for an attribute, simply enter its value into the Value column opposite the attribute's name. The value for an attribute can be any constant that is compatible with the data type of the attribute. Any simple attribute for which you do not specify a value will be set to its default value. All attributes with class types will have an initial value of NIL.

Defining Cursors

There are three ways to add a new cursor to your project:

How to add a cursor	Description
Create a new cursor	Use the Component > New > Cursor command. See below for a complete description.
Copy a cursor from another project using drag and drop	You can copy an entire cursor definition from another Project Workshop by dragging its name from the other Project Component browser and dropping it into the current project.

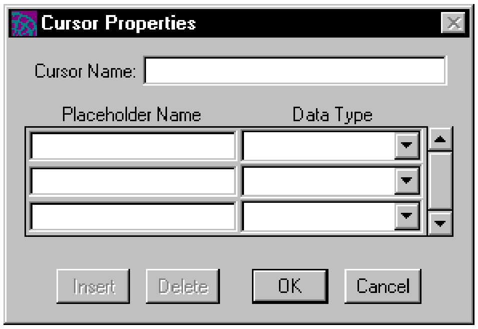
How to add a cursor	Description
Copy a cursor from another project using Copy or Cut command with the Paste command	You can copy an entire cursor definition from another Project Workshop by using the Edit > Copy or Cut command to copy the cursor to the clipboard and then using the Edit > Paste command to paste the cursor into the current project.

The Component > New > Cursor command creates a new cursor. This command opens the Cursor Properties dialog, where you specify the cursor name and placeholders.

➤ **To create a cursor**

1. Choose the Component > New > Cursor command.

   The Cursor Properties dialog opens.



2. In the Cursor Properties dialog, specify the cursor name and placeholders (see "Creating a Cursor" on page 621 for information).

The Cursor Workshop opens.

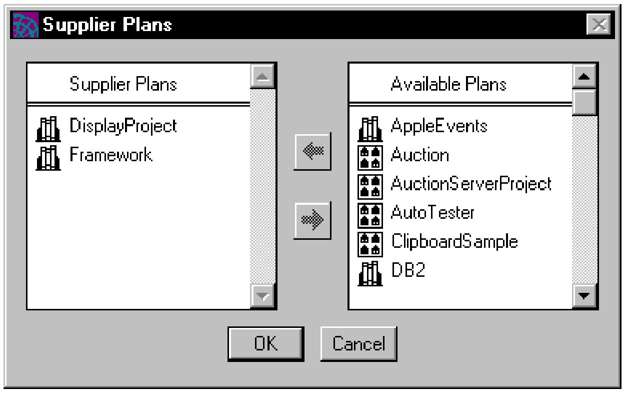
3. In the Cursor Workshop, write the source code for the cursor.

See Chapter 12, "Using the Cursor Workshop," for information on using the Cursor Workshop.

Specifying Supplier Plans

To specify the supplier plans, choose the File > Supplier Plans… command. This command opens the Supplier Plans dialog, shown below, which displays the projects and libraries that are already included in your project along with a list of the available projects and libraries. In this dialog, you can either add or remove items from the list of suppliers.

Figure 4-15 Supplier Plans Dialog



➤ **To add a new project or library to the list of suppliers**

1. Select the project or library name you wish to add.
2. Click the left-pointing arrow.

There are two other ways to add a new project or library to the list of suppliers:

- drag the name from the list of available plans and drop it onto the list of suppliers
- double-click the name you wish to add; iPlanet UDS automatically moves it onto the list of suppliers

If the project or library you wish to add to the list of suppliers is not in your workspace, it will not be displayed on the list of available plans. In this case, you must first use the Include Public Plan command in the Repository Workshop to include the project or library in your workspace. Once the plan has been included in your workspace, you can then add it to your list of suppliers.

➤ **To delete a supplier from the list**

1. Select the project or library you wish to remove from the list.
2. Click the right-pointing arrow.

There are two other ways to remove a project or library from the list of suppliers:

- drag the name from the list of suppliers and drop it onto the list of available projects
- double-click the name you wish to remove; iPlanet UDS automatically moves the project or library off the list of suppliers and back to the list of available items

Using a Library as a Supplier Plan

The *iPlanet UDS Programming Guide* provides complete information on creating, deploying, and installing libraries. Once a library has been installed in your development environment, you can use it as a resource for developing iPlanet UDS applications. To use a library for development purposes, you must include the library as a supplier plan:

➤ **To use a library as a supplier plan**

1. Import the library into the development repository where you wish to use it.

   The repository into which you import the library must be a different one than the one that contains the project originally used to define the library.
2. In the Repository Workshop, use the Include Public Plan command to include the library in your workspace.
3. If desired, browse through the library using the Project Workshop.
4. In the Project Workshop, use the Supplier Plans... command to include the library as a supplier for your project.

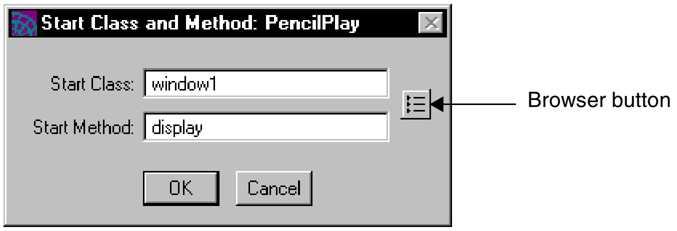
You include a library as a supplier for your project the same way you include a project. Once the library is a supplier for your project, you can use its definitions the same way you use the definitions in a supplier project.

Specifying Start Class and Method

For a client application, you must specify a start class and method in the main project. Although you do not need to set the start class and method when you first create the project, you must set them before you can test the project with the Test Run, Run Distributed, or Test Debug commands.

To specify the start class and method for the project, choose the File > Start Class Method command. This opens the Start Class and Method dialog, shown below. The start class can be any class in the project. The start method can be any method defined for that class. Neither the start class nor method have to exist at the time you fill in the properties. However, they do need to exist by the time you first test the application.

Figure 4-16 Start Class and Method Dialog


Start Class and Method: PencilPlay

Start Class: window1

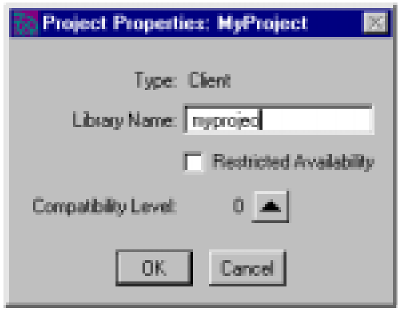
Start Method: display

To make it easier for you to specify the start method, the dialog provides a class browser. When you click the browser button, iPlanet UDS opens a class browser dialog, which displays the project classes and their methods. If you select a method from this browser and click the OK button on the dialog, this will automatically fill in the Start Class and Start Method field on the Start Class and Method dialog.

Setting Project Properties

To set the properties for the project, choose the File > Properties… command. This command opens the Project Properties dialog, shown below, where you can set one or more properties. Simply ignore the properties that are not relevant for your project.

Figure 4-17 Project Properties Dialog



Library Name property The Library Name property allows you to assign a name to the project that is used when the project is configured as a library or included within a library configuration. When there is more than one library within a library configuration, all the library names must be unique. The library name can be any length, however, on platforms where there is an eight-character limit for file names, the library name will be truncated to eight characters.

Restricted Availability property The Restricted Availability property is intended for use when you are integrating your iPlanet UDS application with C routines. If your TOOL project does not include a supplier project, either a C project or another TOOL project, that is defined as being restricted, you can simply ignore this property.

Turn on the Restricted Availability property only if you know that you want your TOOL project to have restricted availability. Turning this on means that iPlanet UDS will then allow you to use the classes in your restricted supplier projects. However, this has serious repercussions on the way you can partition the project. See *Integrating with External Systems* for information on when to use the Restricted Availability property.

Compatibility Level property If you wish to raise the release number for the project, click the Increment button once. The compatibility level will increase by one.

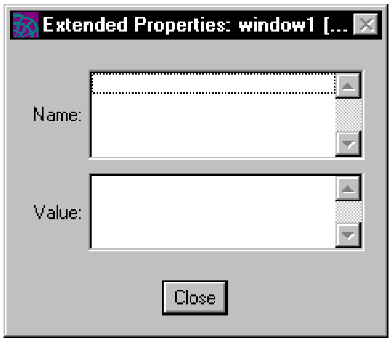
Setting Extended Properties on Project Components

To set the extended properties on an individual project component, select the component name and choose the Component > Extended Properties... command. This command opens the Extended Properties dialog, where you can set any number of properties for the component.

➤ **To create a new property**

1. Select the component name and choose the Component > Extended Properties... command.

   The Extended Properties dialog opens.

2. On the Extended Properties dialog, click the New button.
3. On the New Extended Property dialog, enter the property name and click OK.
4. On the Extended Properties dialog, enter the value for the property in the Value field.

You can delete an extended property on the Extended Properties dialog by selecting the property name and clicking the Delete button.

Modifying a TOOL Project

The Project Workshop allows you to modify only TOOL projects. To modify an external project, you must edit the project definition in the text file where it was originally defined. See *Integrating with External Systems* for information.

Before modifying a TOOL project, you must have write access to the components you wish to change. See "Write Access to Project Components" on page 257 for information.

When you open your workspace in the Repository Workshop, you must open it for modifying.

➤ **To modify a project**

1. In the Repository Workshop, choose the Plan > Open command, or double-click the project name.
2. In the Project Workshop, choose the Component > Checkout (or Branch) command to get write access to the components you wish to change (if they are read only).
3. In the Project Workshop, make the desired changes.
4. Use the File > Save All command as necessary to save the changes to your workspace.

If you are sharing a repository with other developers, you need to update your workspace to test how your changes interact with the other developer's changes. You then need to integrate your workspace to make the modifications available to your collaborators. See "Updating and Integrating a Workspace" on page 175 for information.

The Project Workshop allows you to modify the project by:

- modifying the definitions of classes, interfaces, constants, service objects, and cursors
- deleting classes, interfaces, constants, service objects, and cursors
- copying project components from other projects by dragging and dropping component names
- modifying the supplier plans, the start class and method, the project properties, and the extended properties for project components

The Edit menu in the Project Workshop provides commands that let you use the clipboard to copy or move project components from one project to another. The Edit > Find Text… and Replace Text… commands let you find all occurrences of a specified string within the source code for the project and to make global replacements.

The following sections provide detailed information about how to modify a TOOL project. "Reverting a Project" on page 256 provides information about how to remove the modifications you have made to a project.

Modifying a Class

To modify a class, double-click the class name in the Project Components browser, or select the class name and choose the Component > Open… command. This command opens the Class Workshop, which displays the original definition of the class. See "Modifying a Class" on page 322 for information about modifying a class.

Modifying an Interface

To modify an interface, double-click the interface name in the Project Components browser, or select the interface name and choose the Component > Open... command. This command opens the Interface Workshop, which displays the original definition of the interface. See "Modifying an Interface" on page 358 for information about modifying an interface.

Modifying a Constant

To modify a constant, double-click the constant name in the Project Components browser, or select the constant name and choose the Component > Open… command. This command opens the Constant Properties dialog, where you can make any desired changes.

Modifying a Service Object

To modify a service object, double-click the service object name in the Project Components browser, or select the service object name and choose the Component > Open… command. This command opens the Service Object Properties dialog, where you can make your changes.

Modifying a Cursor

To modify a cursor, double-click the cursor name in the Project Components browser, or select the cursor name and choose the Component > Open… command. This command opens the Cursor Workshop, where you can modify the selected cursor. See "Editing a Cursor" on page 622 for information on modifying a cursor.

Modifying Project Properties

The following commands allow you to modify the project properties:

Command	Description
File > Start Class Method…	Lets you change the start class and method for the project.
File > Supplier Plans…	Lets you change the supplier plans for the project.
File > Properties…	Lets you change the other project properties, including the compatibility level and whether the project has restricted availability.
Component > Extended Properties...	Lets you change the extended properties set on individual project components.

Deleting Components

The Edit > Delete command lets you delete project components from the Project Components browser. To delete a project component, it must be either new or checked out (it cannot be branched or read only).

➤ **To delete a project component**

1. Select the component you wish to delete by clicking on the name.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the component.

Copying and Moving Components

The Edit menu in the Project Workshop allows you to cut or copy a project component onto the clipboard the same way you cut or copy text in a text editor. Once the project component is in the clipboard, you can paste it into another project definition.

The commands on the Edit menu for using the clipboard are:

Menu item	Function
Cut	Removes the selected project component from the current project and copies to the clipboard.
Copy	Copies the selected project component onto the clipboard.
Paste	Pastes the project component in the clipboard into the current project.

You can also copy a component from one Project Workshop to another using "drag and drop."

➤ **To copy a project component**

1. Drag the component name to the Project Component browser where you wish to add the copy.
2. Drop the component name in the Project Component browser.

When you drag and drop a project component from one Project Workshop to another, iPlanet UDS copies the entire component definition. For example, if you drag and drop a class name, iPlanet UDS copies all the attributes, methods, events, and event handlers associated with the class.

Order dependencies Note that if the component you are copying from an existing project references other components that are not defined in the new project, you will get an error. For example, if you try to copy a service object whose class is not in the current project (or its suppliers), iPlanet UDS will not allow you to copy the service object. Therefore, you must be sure to copy the components in the correct order.

Finding and Replacing Text

The Edit > Find Text… and Replace Text… commands in the Project Workshop let you search for all occurrences of a specified string within the method and event handler source code for the project.

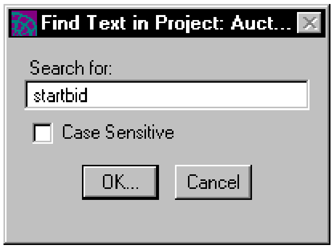
Finding Text

The Find Text… command finds all occurrences of the specified string in the method and event handler source code in all classes in the project.

➤ To find a string

1. Choose the Edit > Find Text… command.

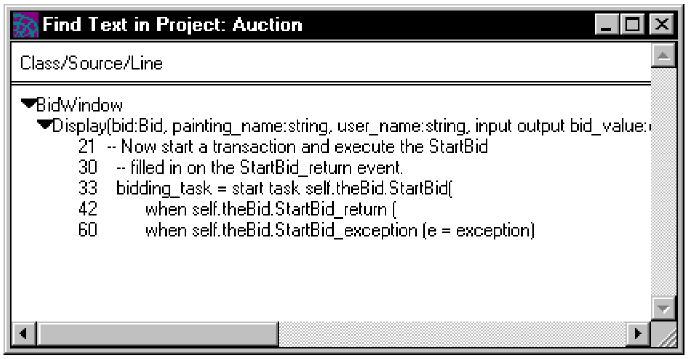
   The Find Text dialog opens.



2. In the Find Text dialog, specify the string you wish to search for and click the OK button to start the search.

   The default is a case-insensitive search. If you want a case-sensitive search, click the Case Sensitive toggle to on.

3. When the search is complete, the Find Text in Project dialog opens, displaying the class name, method or event handler name, and the line itself for each source code line that contains the string.



```
▼BidWindow
  ▼Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:
      21  -- Now start a transaction and execute the StartBid
      30   -- filled in on the StartBid_return event.
      33   bidding_task = start task self.theBid.StartBid(
      42        when self.theBid.StartBid_return (
      60        when self.theBid.StartBid_exception (e = exception)
```

4. Double-click any line in the Find Text in Project dialog to open the Method or Event Handler Workshop to display the original source code.

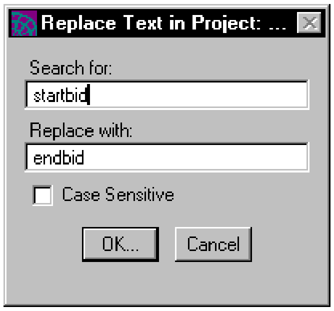
Replacing Text

The Replace Text… command finds all occurrences of the specified string in the method and event handler source code in all classes in the project.

➤ To make a global replacement

1. Choose the Edit > Replace Text… command.

   The Replace Text dialog opens.



Replace Text in Project: ...

Search for: startbid

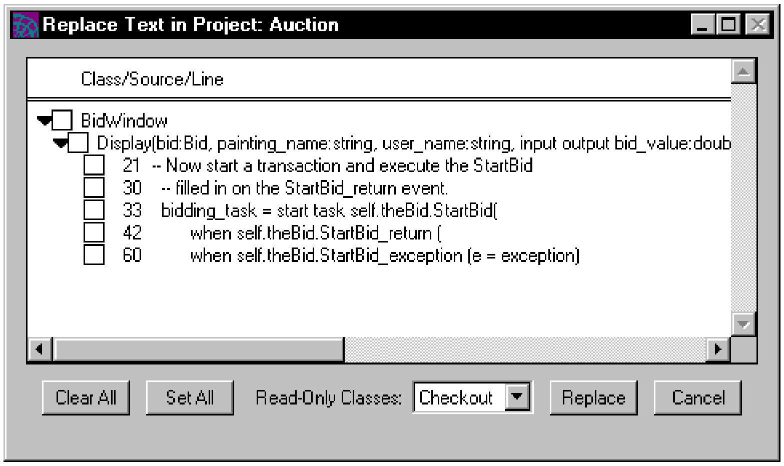
Replace with: endbid

☐ Case Sensitive

2. In the Replace Text dialog, specify the string you wish to search for and the replacement string. Click the OK button to start the search.

   The default is a case-insensitive search. If you want a case-sensitive search, click the Case Sensitive toggle to on.

3. When the search is complete, another Replace Text in Project dialog opens, displaying the class name, method or event handler name, and the line itself for each source code line that contains the string.



Replace Text in Project: Auction

Class/Source/Line

```
☐ BidWindow
  ☐ Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:doub
     ☐  21  -- Now start a transaction and execute the StartBid
     ☐  30   -- filled in on the StartBid_return event.
     ☐  33  bidding_task = start task self.theBid.StartBid(
     ☐  42       when self.theBid.StartBid_return (
     ☐  60       when self.theBid.StartBid_exception (e = exception)
```

Read-Only Classes: Checkout

4. In the Replace Text in Project dialog, click the toggle next to the line where you wish to make the replacement. See below for further information about using the Replace Text in Project dialog.

5. In the Replace Text in Project dialog, click the Replace button to make the replacements.

Read-only classes By default, if a class is read only, the Replace Text… command automatically checks it out before making the replacement in its method source code. If it cannot be checked out, you will be prompted to indicate whether you wish to branch or ignore the read-only class.

If you do not want read-only classes to be checked out, you can choose one of the following options from the Read-only Classes drop list on the Replace Text in Project dialog.

Replacement Option	Description
Ignore	Will not make the replacement in the class.
Branch	Will branch the class.
Checkout	Will attempt to check out the class (the default).

Selecting Lines

The Replace Text in Project results dialog provides several features to make it easier to choose the lines where you want to make the replacement. Use the Set All button to click the toggles for all the source code lines to on. Use the Clear All button to click the toggles for all the source code lines to off. If you click the toggle for a method or event handler name to on, this turns the toggles on for all relevant lines in the method or event handler. If you click the toggle for a class name to on, this turns the toggles on for all relevant lines in all methods and event handlers in the class.

Reverting a Project

The File > Revert command lets you erase all changes you have made to the project and its components since your last Integrate Workspace command. The changes that are reverted include the following:

- new components are removed
- deleted components are restored
- project properties are reverted
- modifications to existing components are removed

After the project is reverted, the entire project will be in the same state it was in the system baseline (not your workspace) the last time you used the Update Workspace command.

For any project components that were checked out, the Revert command frees the component for checkout by another workspace. For any project components that were branched, the Revert command removes the branch.

➤ **To revert your changes**

1. Choose the File > Revert command.
2. Confirm that you wish to revert the project.

Write Access to Project Components

Until you give your first Integrate Workspace command for a new project, you have write access to all its components. However, after you have integrated, the project components are read only until you do one of the following:

- checkout the component
- branch the component

Typically, you make permanent changes to a component. Therefore, you should checkout the component. The changes that you make to a component that you have checked out are added to the system baseline when you give an Integrate Workspace command. This makes your changes available to other developers when they use the Update Workspace command on their own workspaces.

However, if you wish to make temporary changes or if the component is already checked out by another user and you wish to test some changes, you can branch the component. Branching a component gives you temporary write access to the component without permanently affecting the repository. You cannot integrate a branched component, so changes you make to a component that you have branched are lost when you integrate your workspace. It is possible to convert a branched component to be checked out and therefore make the changes permanently, but this will work only if no other workspace has checked out the component since you gave your last Update Workspace command.

When you want to modify an existing project, you have the choice of checking out individual components of the project that you want to modify, that is, specific classes, interfaces, cursors, constants, or service objects, or you can check out all the project components. Obviously, if you are collaborating with other developers, you should only check out those components that you need. The following sections describe first how to checkout and branch individual project components and then how to check out and branch all the project components.

Checking Out a Component

The Checkout command gives you an exclusive write lock on the component. Before you can checkout a component, the workspace must be open for modifying. Only one workspace can checkout a component at a time.

After you have finished your modifications, you do not have to check the component back in. It is automatically checked in when you integrate your workspace as described under "Integrating a Workspace" on page 179.

➤ **To checkout a project component**

1. In the Project Component browser, select the component you wish to checkout.
2. Choose the Component > Checkout command.

   ✓

3. When the component is checked out, the Project Components browser displays a checkout icon by the component name.

If the component you checkout is branched, the branch will be converted to a checkout, as long as no other workspace has checked out the component since you gave your last Update Workspace command.

The Checkout command will fail if the component is already checked out by another workspace. The error message will give the name of the workspace that has checked out the component. The Checkout command will also fail if the workspace does not have the latest version of the component. In this case, use the Update Workspace command to bring the latest version of the component into your workspace (see "Updating a Workspace" on page 176 for information).

If you wish to undo the changes you have made since the Checkout command, and revert the component to the state it was in after your last Update Workspace command, you can use the Undo Checkout/Branch command as described under "Undoing Changes."

Branching a Component

The Component > Branch command gives you temporary write access to a component in order to test a change while someone else has checked out the component. You cannot integrate changes in a branched component into the system baseline.

Before you can branch a component, the workspace must be open for modifying. Any number of workspaces can branch a component at the same time.

➤ **To branch a project component**

1. In the Project Component browser, select the component you wish to branch.
2. Choose the Component > Branch command.
3. When the component is branched, the browser displays a branch icon by the component name.

If the component you branch is checked out, the checkout will be converted to a branch. When a checkout is converted to a branch, your changes to the component are retained in the workspace, but the component is free to be checked out by another workspace after the next Save All command.

If you wish to undo the changes you have made since the Branch command, and revert the component to the state it was in before your last Branch command, you can use the Undo Checkout/Branch command as described next.

Undoing Changes

The Component > Undo command lets you erase all changes you have made to a component since you gave the Checkout or Branch command. The component reverts to the state it was in before the Checkout or Branch command, and the component is freed for checkout by another workspace.

➤ **To undo your changes**

1. In the Project Component browser, select the component you wish to revert.
2. Choose the Component > Undo > Checkout/Branch command.

If you have deleted a component after having checked it out, you can restore the component with the Undo command.

➤ **To restore a component**

1. Choose the Component > Undo > Deleted Checkouts... command.
2. In the Deleted Checkouts dialog, select the component you wish to restore.
3. Click the OK button to undelete the component.

The component you have restored will revert to the state it was in before you originally checked it out. The restored component will appear in the Project Components browser, and it will no longer be checked out.

Of course, if you have deleted a component that was never integrated, you cannot use the Undo Deleted Checkouts… command to restore it.

Checking out All Project Components

The File > Checkout All Components command gives you an exclusive write lock on all the existing project components. It does not, however, prevent other workspaces from creating new project components or from changing the project properties.

After you have finished your modifications to the project components, you do not have to check the components back in. They are automatically checked in when you integrate your workspace.

Before you can checkout all the project components, the workspace must be open for modification.

➤ **To checkout all project components**

1. Choose the File > Checkout All Components command.
2. If some of the components are unavailable, you will be prompted to indicate whether or not you wish to checkout only the available project components.

   Click Yes if you wish to checkout the components that are available.

   Click No if you want to cancel the operation, that is, you do not wish to checkout any components.

There are two situations in which you will not be able to check out a project component:

- another workspace has already checked out the component
- your workspace does not contain the latest version of the component because another workspace integrated a new version of the component into the system baseline since the last time you gave an Update Workspace command

If any of the project components are unavailable for checkout for the above reasons and you indicated that you wish to checkout the available components, the Checkout All Components command will checkout only the available components and will display a warning message telling you that not all the components were checked out. If you click the More button on the error dialog, you can see specific information on the individual components. If any of the project components are unavailable for checkout for the above reasons and you indicated that you only wanted to checkout all the project components, none of the project components will be checked out.

If any of the components in the project are branched, the Checkout All Components command will convert the branches to checkouts. This will work only if no other workspace has checked out the component since you gave the Branch command.

If you wish to undo the changes you have made since the Checkout All Components command, and revert the components to the state they were in after your last Update Workspace command, you can use the Revert command as described under "Reverting a Project" on page 256. You can also revert individual project components with the Undo commands described under "Undoing Changes" on page 259.

Branching All Project Components

The Branch All Components command gives you temporary write access to the project components to test out changes while someone else has checked out the components (or to patch a baseline). The changes you make to a branched component cannot be integrated into the system baseline unless you convert the branch to a checkout.

➤ **To branch all project components**

1. Choose the File > Branch All Components command.

If any of the components in the project are checked out, the checkouts will be left intact. They will not be converted to branches.

If you wish to undo the changes you have made since the Branch All Components command, and revert the components to the state they were in after your last Update Workspace command, you can use the Revert command as described under "Reverting a Project" on page 256. You can also revert individual project components with the Undo commands described under "Undoing Changes" on page 259.

Testing a TOOL Project

The Project Workshop provides several different ways to test a project:

Command	Description
File > Compile	Compiles all the classes in the project and reports the compilation errors.
Run > Test Run	Executes the project locally from the project's start class and method, and reports errors.
Run > Test Code Fragment…	Executes the TOOL start-up code that you specify, which lets you run a branch of your application, or runs any TOOL code fragment.
Run > Test Debug	Starts the Debugger for the project, which allows you to monitor the code as it is being executed.
Run > Test Run With Profiling	Executes the project locally from the project's start class and method, using the profiling options that you select. Provides profiling for the client partition.
Run > Run Distributed	Executes the project from the project's start class and method, using the default configuration.

Testing a server project Because a server project does not have a start class and method, you cannot run or debug the project. (Of course, if you run an application that includes the server project, you can run or debug the server as part of that application.) However, you can test a server project by compiling the classes or by using the Test Code Fragment command to specify start-up code for the project.

Compiling the Project

The File > Compile command compiles all the classes and interfaces in the project and reports the compilation errors. While it is not necessary to compile your project before you run it, this command allows you to check for syntax errors without actually executing the code. Any compilation errors are reported in the Error window (see "Using the Error Window" on page 269 for information).

Running the Project Locally

To run the project from the start class and method, choose the Run > Test Run command or click the Run Project button. iPlanet UDS starts executing the project by invoking the start method on an object of the start class, and it runs until the application exits. Any errors are reported in the Error window (see "Using the Error Window" on page 269 for information).

If you are unable to exit the application, you can use the Run > Cancel Run command at any time to cancel execution.

Partitioning for testing purposes When you run a project locally, iPlanet UDS automatically partitions your project so that as much of it as possible will run on your client machine. This allows you to test the project independently of its deployment environment. When you are using the Debugger, it also allows you to step through code that would normally be executing remotely.

To partition your project for testing, iPlanet UDS changes the visibility for all your Environment visible service objects to User visible. The user interface for the application, the application start code, and all service objects that the client node can support are assigned to the client partition. iPlanet UDS runs the client partition directly on your workstation. (For information about client partitions, see "About Partitions" on page 664.)

Any service objects that the client node cannot support (because the external manager or a restricted project is not available on the client node) are assigned to a special private partition, which iPlanet UDS automatically installs on a suitable node in your development environment. This ensures that two people testing the same project will not accidentally share the same service object. For further information about partitioning, see "About Application Configurations" on page 661.

Running with Profiling

When you test run your application, you can request profiling for the client partition. When you turn on profiling, the profiler counts instructions executed in the TOOL interpreter, which is useful for finding problems in the interpreted code and for viewing the dynamic call flow of an application.

Note that profiling takes affect for the client partition only. No profiling is provided for the server partitions or for any method calls into the server partitions.

The following table describes the profiling options that you can request.

Setting	**Description**
Log File Path	Enter the name of the directory in which you want the log file stored. When you open the Profiling Options dialog, the initial log file path is `$FORTE_ROOT/log`. If you erase the initial log file path from the properties dialog, and leave the Log File Path setting blank, the log file path will default to the path from which you started iPlanet UDS.
Log File Name	Enter the name of the file in which to store the profile information. The default is profile.txt.
Logger Flags	Enter any additional trace flags, including user-defined flags (see *iPlanet UDS Programming Guide* for information about setting log flags).
Call Frequency By Application	This setting, which is on by default, records all the methods that are called during the execution of an application. The output displays how many times each method was called, and the total number of instructions executed during the execution of the method. The methods are sorted in descending order of instructions executed.

Setting	Description
Call Frequency By Task	Turn on this toggle to record all the methods that are called during the execution of a task. The output displays how many times each method was called, and the total number of instructions executed during the execution of the method. The methods are sorted in descending order of instructions executed.
Calltree for Application	Turn on this toggle to record the call tree of an application. The call tree shows a method and all of the methods that it calls. For each of the called methods, all called methods are shown, and so on. The methods are sorted in descending order of instructions executed within each level of the call tree.
Calltree for Task	Turn on this toggle to record the call tree of a task. The call tree shows a method and all the methods that it calls. For each of the called methods, all called methods are shown, and so on. The methods are sorted in descending order of instructions executed within each level of the call tree.
Method Entry and Exit	Turn on this toggle to record each time a method is entered and exited.
Memory Recovery	Turn on this toggle to record each time a memory reclamation occurs.
List Objects Collected	Turn on this toggle to record the set of reclaimed objects.
List Objects Retained	Turn on this toggle to record the set of retained objects.

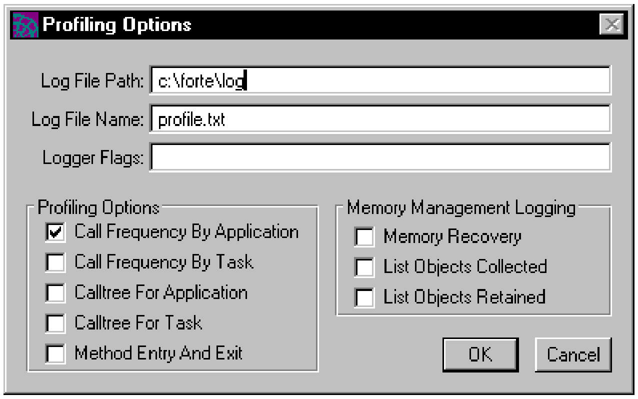
As your application runs, trace information will appear in the log window. After the application completes, you can open the log file for a complete record of all the information that you requested in the Profiling Options dialog.

To test run with profiling, you use the Run > Test Run with Profiling command. The Test Run with Profiling command opens the Profiling Options dialog, where you must set the profiling options. At least one setting must be turned on (the Call Frequency By Application options is on by default), otherwise, no profiling will be provided.

➤ **To test run an application with profiling**

1. Choose the Run > Test Run With Profiling command.

   The Profiling Options dialog opens.



Log File Path: c:\forte\log
Log File Name: profile.txt
Logger Flags:

Profiling Options
- [✓] Call Frequency By Application
- [ ] Call Frequency By Task
- [ ] Calltree For Application
- [ ] Calltree For Task
- [ ] Method Entry And Exit

Memory Management Logging
- [ ] Memory Recovery
- [ ] List Objects Collected
- [ ] List Objects Retained

2. Select the desired profiling options. By default, there is no profiling, so you must choose at least one option.

3. Click OK.

As your application runs, trace information will appear in the log window. After the application completes, you can open the log file for a complete record of all the information that you requested in the Profiling Options dialog.

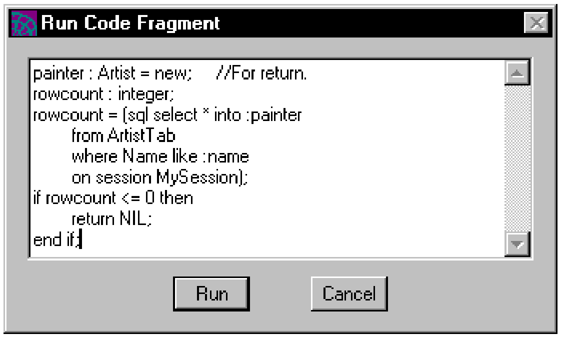
Running a Code Fragment

You can use the Test Code Fragment… command to run any TOOL code fragment or to specify temporary start-up code for your project.

Start-up code To specify start-up code for your project, create an object of a given class and invoke a method on it (see the *TOOL Reference Guide* for information on how to do this). For example, to run a single window, you could create a UserWindow object and invoke the Display method on it.

The Test Code Fragment command opens the Run Code Fragment dialog, shown below, where you can enter the TOOL code to be executed.

Figure 4-18 Run Code Fragment Window



```
painter : Artist = new;     //For return.
rowcount : integer;
rowcount = (sql select * into :painter
       from ArtistTab
       where Name like :name
       on session MySession);
if rowcount <= 0 then
       return NIL;
end if;
```

➤ **To run a code fragment**

1. Choose the Run > Test Code Fragment command.
2. In the Run Code Fragment dialog, enter the TOOL code.
3. Click the Run button to execute the code.

Like the Test Run command, the Test Code Fragment command executes the code until it exits or until you use the Cancel Run command to stop it. Any errors are reported in the Error window (see "Using the Error Window" on page 269 for information on this).

Partitioning used for project When you use the Test Code Fragment command to start up a project, iPlanet UDS partitions your project the same way it does when you run a project locally (described above).

Running the Project in a Distributed Environment

The Run > Run Distributed command runs the project using the current configuration. The current configuration is the configuration that was open the last time you used the Partition workshop. The Run Distributed command provides a simple way to test the project in the development environment. See "Testing a Client Configuration" on page 712 for information about testing a configuration.

To run the project using the current configuration, choose the Run > Run Distributed command. iPlanet UDS starts executing the project by invoking the start method on an object of the start class, and it runs until the application exits. Any errors are reported in the Error window (see "Using the Error Window" on page 269 for information).

The first time you run the project using the Run Distributed command, iPlanet UDS starts up the remote partitions for the application and leaves them running, even when you exit from the application. This is for efficiency reasons, so that the next time you run the application, it is not necessary to restart the remote partitions.

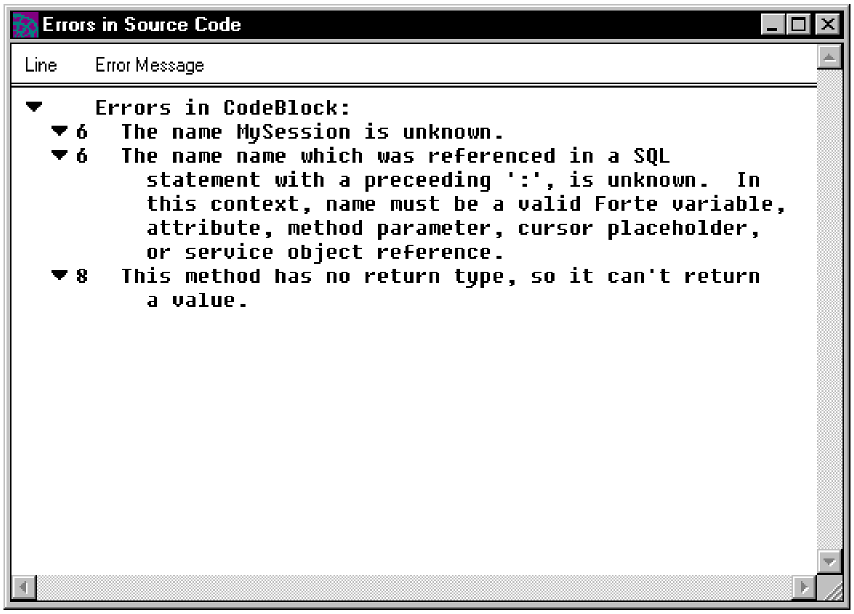
iPlanet UDS stops and restarts remote partitions as you make changes to the project code that affects partitioning. iPlanet UDS also stops all remote partitions when you exit from the iPlanet UDS workshops. However, if necessary, you can stop remote partitions explicitly at any time by using the Run > Stop Remote Partitions command. See "Running the Application" on page 714 for more information about stopping remote partitions.

If you are unable to exit the application, you can use the Run > Cancel Run command at any time to cancel execution. The Cancel Run command cancels the client partition for the application. The remote partitions will still continue to run; use the Stop Remote Partitions command to stop these.

Using the Error Window

The Error window displays the error messages for the project or code fragment in an outline field.

Figure 4-19 Error Window



```
▼     Errors in CodeBlock:
  ▼ 6   The name MySession is unknown.
  ▼ 6   The name name which was referenced in a SQL
          statement with a preceeding ':', is unknown.  In
          this context, name must be a valid Forte variable,
          attribute, method parameter, cursor placeholder,
          or service object reference.
  ▼ 8   This method has no return type, so it can't return
          a value.
```

You can jump directly from one of these messages to the code that caused the error. In the Error window, double-click the error you wish to find. The appropriate workshop will then open to highlight the code that caused the error.

You can keep the Error window open as long as you need it. When you are finished using the window, simply close it.

Debugging the Project

To debug the project, choose the Run > Test Debug command or click the Debugger tool. The Debugger uses the start class and method as the starting point for running the application. iPlanet UDS begins executing the application by constructing a new object of the start class. The Debugger displays the code for the start method in the Task Window, and suspends execution immediately before the first statement in the start method.

Partitioning for debugging purposes When you debug a project from the Project Workshop, iPlanet UDS partitions your project the same way it does when you run a project locally (described under "Running the Project Locally" on page 263). Running as much of the project as possible on the client partition allows you to step through code that would normally be executing remotely. The Debugger only allows you to step through code that is executing locally. If a method is invoked on a remote object, iPlanet UDS executes the complete method. See Chapter 13, "Using the Debugger," for information about using the Debugger.

Importing and Exporting Classes and Interfaces

From the Project Workshop, you can import a single class or interface into your project. The Import Class/Interface... command allows you to take a class or interface that was exported from another project and add it to your project. The class or interface export file must have been created either by the Export Class/Interface… command or by the iPlanet UDS Fscript utility (see *Fscript Reference Guide* for information on Fscript).

The Export Class/Interface… command in the Project Workshop allows you to export a class or interface definition to a standard text file, which you can then import into another iPlanet UDS project.

Importing a Class or Interface

The Component > Import Class/Interface… command takes an iPlanet UDS class or interface definition stored in a text file and adds it to your project. The class or interface definition that you import must have been created either by the Export Class/Interface… command or by the iPlanet UDS Fscript utility.

Overwriting an Existing class or Interface

You can import a new class or interface, or an existing class or interface. If the class or interface you are importing already exists, the Import Class/Interface… command will overwrite it. Therefore, you must be sure the existing class or interface is in a writeable state, either new, checked out, or branched.

➤ **To import a class or interface**

1. Choose the Component > Import Class/Interface… command.
2. In the file selection dialog, specify the name of the file that contains the class or interface definition.

After the class or interface has been successfully imported, iPlanet UDS displays a message indicating that it was added to the project. The new class or interface name is then displayed in the Project Components browser. If there is an error, such as a bad file, iPlanet UDS displays an error message.

Importing multiple components The file that you import with the Import Class/Interface... command can define any number of project component definitions, including class, interface, service object, cursor, and constant definitions. If the file that you import contains more than one component definition, all component definitions in the file will be imported into the current project.

Exporting a Class

The Component > Export Class/Interface… command writes the definition of the current class or interface into a standard text file.

➤ **To export a class or interface**

1. In the Project Components browser, select the class or interface you wish to export.
2. Choose the Component > Export Class/Interface… command.
3. In the file selection dialog, specify the name of the file to contain the definition. If you give the name of an existing file, the Export Class/Interface… command overwrites the file.

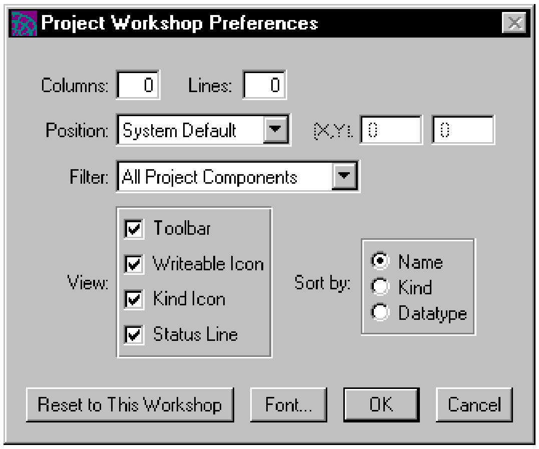
While the class or interface is being exported, iPlanet UDS displays a message indicating that the component is being written to the specified file and prevents all input until the export is complete.

Setting Workshop Preferences

The Project Workshop allows you to set preferences that are saved as part of your current workspace.

To set the workshop preferences, give the File > Workshop Preferences… command. This command opens the Project Workshop Preferences dialog, where you can set any number of preferences.

Figure 4-20 Project Workshop Preferences Dialog


Project Workshop Preferences

Columns: 0 Lines: 0

Position: System Default (X,Y): 0 0

Filter: All Project Components

View: [✓ Toolbar] [✓ Writeable Icon] [✓ Kind Icon] [✓ Status Line]

Sort by: [✓ Name] Kind Datatype

The preferences you can set for the Project Workshop fall into the following general categories:

- workshop window size and position
- default filter
- viewing preferences
- sorting preference
- font preference

The workshop window size and position, filter, viewing, and font preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Project Workshop.

Sorting Preference

The sorting preference lets you specify the default sorting used for the Project Components browser. The following table describes the sorting options:

Preference	**Description**
Name	Sorts the project components by name.
Kind	Sorts the project components by kind.
Datatype	Sorts the project components by data type.

Chapter 5

Using the Class Workshop

This chapter provides conceptual information about classes and their elements, and describes how to use the Class Workshop.

In this chapter, you will learn how to:

- examine a class
- create a class
- modify a class
- test a class
- set Class Workshop preferences

About Classes

The Class Workshop allows you to create custom classes specifically for your project or library. Custom classes are always subclasses of iPlanet UDS classes or supplier classes.

There are three kinds of classes in iPlanet UDS: standard classes, window classes, and domain classes.

Nonwindow class A standard class (also called a "nonwindow class") is a template that defines an object. A standard class can be a subclass of any iPlanet UDS class that allows subclassing or any custom class.

Window class A window class is always a subclass of the iPlanet UDS UserWindow class, or one of its subclasses. When you create a window class, iPlanet UDS automatically creates a window that is associated with the class. You design the window using the Window Workshop.

Domain class A domain is a special kind of class that combines a DataValue subclass with an iPlanet UDS widget. A domain class is always a subclass of one of the iPlanet UDS nullable DataValue subclasses. The domain's superclass determines the type of data stored in the object and provides a SetValue method for performing automatic data validation on it. A domain is also always associated with a widget class. The widget class specifies the kind of widget, such as a data field or a radio list, that is used to display the data when the domain is added to a form.

Domain classes are intended mainly for use with the optional Business Model Workshop (see *A Guide to Express* for information). However, you can also create domains as customized widget types for use with the Window Workshop.

This section begins by providing conceptual information about the elements common to all classes:

- attributes
- methods
- events
- event handlers
- constants

After the descriptions of the individual class elements, this section explains the various class properties. The section concludes with background information about window classes and domain classes.

Attributes

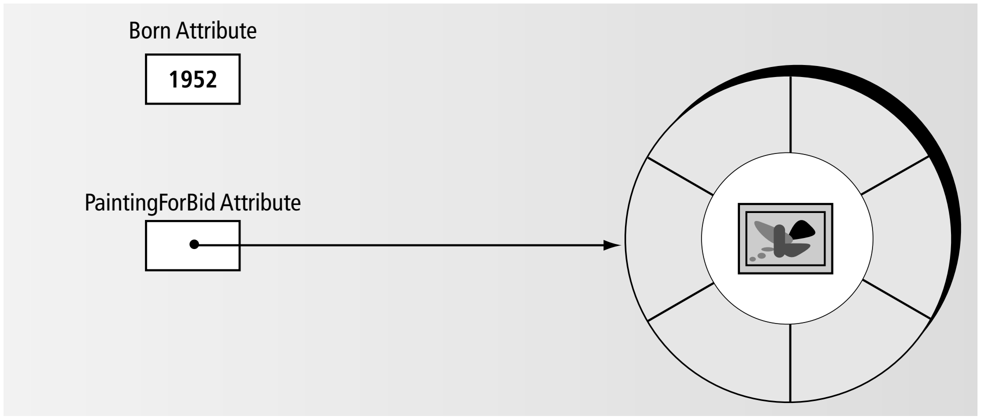
An attribute is a data item. You can think of it as a variable that is associated with a single object (in fact, some object-oriented languages call attributes "instance variables").

Every attribute has a data type. The data type can be any simple data type or any class.

Simple types The simple data types are boolean, numeric (with a choice of integer, long, float, or double), string, and pointer. If an attribute has one of these simple types, the attribute itself contains the data. For example, in the Art Auction's Painter class, the Born attribute is an integer type. This attribute stores a numeric value, such as 1856, 1742, or 1910. When you create an attribute with a simple type, its initial value is the default value for the data type (see the *TOOL Reference Guide*).

Class types If the attribute has a class type, the attribute points to an object of that class. The attribute contains a reference to the object and the object contains the data. For example, the Bid class has an attribute called PaintingForBid of type Painting. This attribute points to an object of the Painting class, which has its own set of attributes.

Figure 5-1 Simple and Object Attributes



When you create the attribute with a class type, it has an initial value of NIL, which means "no object." You create the object for the attribute when you assign a value to it in TOOL by using an object constructor. An object constructor creates a new object and specifies values for one or more of its attributes. For information on object constructors, see the *TOOL Reference Guide*.

Virtual Attributes

A virtual attribute is a special kind of attribute that does not store a value or point to an object. Instead, a virtual attribute consists of two expressions, one that is evaluated when the program sets the value of the attribute and another that is evaluated when the program gets the value of the attribute. A virtual attribute is useful when you want to provide an attribute that is based on a calculation. For an example of a virtual attribute definition, see "Defining Virtual Attributes" on page 311.

The Get expression for the virtual attribute is required, but the Set expression is optional. A virtual attribute without a Set expression is a read-only attribute.

Methods

A method is a procedure that is specially written to operate on an object. Every method has a statement block that contains the TOOL code that performs the operations on the object. A method may also have a return type and one or more parameters.

As an example, the Auction Manager's Bid class has a CompleteBid method that updates the bid on a painting. The bid parameter of this method specifies the latest amount for the bid. The TOOL code for the method performs the work of updating the Bid object and notifying interested clients that a new bid was made.

Chapter 10, "Using the Method Workshop," contains detailed information about methods and the Method Workshop.

Overloading Methods

In object-oriented programming, there can be more than one method with the same name. This is called *overloading* the method. Overloaded methods have the same name, but different parameters. See "Overloading Methods" on page 573 for further information about overloaded methods.

Method signatures When there are two or more methods with the same name, each method has its own method signature. A *method signature* is the method name and the parameter list. For example, here are two method signatures for the RequestFocus method:

RequestFocus( )
RequestFocus(row:TemplateField, scrollPolicy:integer)

When you are working with an overloaded method in the Class Workshop, you work with the individual method signatures associated with the overloaded method as separate methods. For example, to update an overloaded method, you open each method signature separately.

Converter methods A *converter* method is a special kind of method that you create to adjust for the differences in a class element between two (or more) versions of a class. When you are performing a class version upgrade of an iPlanet UDS application (described under "Class Versions" on page 289), you must write converter methods. The purpose of converters is to enable code that calls or expects one version of a class (by invoking a method or posting an event, for example) to actually use a different version of the class. A converter allows older and newer code to communicate by "bridging" the differences between two versions of a method or an event. The *iPlanet UDS Programming Guide* provides complete information on converter methods.

CAUTION You should *not* use converters without a thorough understanding of iPlanet UDS upgrading features. Upgrading an iPlanet UDS production environment requires very careful planning. See the *iPlanet UDS Programming Guide* for complete information.

Events

fAn event is a signal that something has changed. You use custom events with the `post` statement to notify the rest of the application that something significant has occurred. For example, the Art Auction uses the BidCompleted event to notify all clients displaying a particular painting that a bid was made on that painting.

Every event has a name. You use this name when you want to trigger the event (with the `post` statement) and respond to the event (with the `event` statement).

Event parameters An event may also have one or more parameters. The event parameters allow you to pass data along with the notification that the event has occurred. For example, the BidCompleted event has parameters that specify the value of the bid, the time the bid was made, and the name of the bidder. Any method responding to the event can then use those values for processing.

Like attributes, event parameters can be simple types or class types. A parameter with a simple type contains the data. A parameter of a class type contains a reference to an object of that class.

iPlanet UDS allows you to assign a default value for an event parameter. iPlanet UDS uses this value for the parameter when the event is posted without specifying a value for the parameter. The default value for the parameter must be compatible with the parameter's data type. For parameters with a class data type, the default value can only be NIL, which means "no object."

For further information on using events in TOOL, see the *TOOL Reference Guide*.

Event Handlers

An event handler is a named block of TOOL code that provides programming to be executed in response to one or more events. The event handler provides reusable, modular event handling code that you can include in any number of event statements.

Storing named event handlers as part of a class definition allows you to provide event handling code that is inherited by subclasses. This is especially useful when you are creating window classes that inherit part of their physical layout from a superclass. When window classes inherit event handlers along with widgets from their superclass, they are inheriting the appropriate event handling code for the inherited widgets. To use the inherited event handlers, a window subclass uses the `register` statement within its event loop.

Event handlers can also be useful for storing event handling code in a modular, reusable form. For example, when you are creating a window that is going to be nested within other windows, you can define named event handlers for the nested window. This provides event handling code that can be used by any other window that runs the nested window.

Parameters for event handler All event handlers consist of a name and TOOL event handling code to be included in one or more event statements. Some handlers also include parameters, which allow the programmer to pass information to the event handler, such as the object for which the events are being handled.

Chapter 11, "Using the Event Handler Workshop," contains detailed information about event handlers and the Event Handler Workshop. For information on using event handlers in TOOL, see the *TOOL Reference Guide*.

Constants

A constant is a literal string or numeric value that has a name. When you declare the named constant, you specify a constant name and a value. You can then use the name in place of the value in your TOOL code.

In the Class Workshop, you can define a constant as part of a class. This means that any methods in the class can reference the constant. Other classes must reference the constant with the syntax *class_name.constant_name*.

Note that using TOOL you can declare a local constant within a statement block (only the current statement block can reference it). See the *TOOL Reference Guide* for information on referencing constants and on declaring local constants.

CAUTION Remember, although you can use constants to specify values in your TOOL code, you cannot use them to specify values in dialogs in the iPlanet UDS Workshops. For example, you cannot use a constant to specify the default value for a parameter in the Method Workshop.

Visibility of Class Elements

Class elements can be public or private.

Public class elements By default, the methods, attributes, events, event handlers, and constants in a class are public, which means that any other classes in the project can access them. When a method is public, a method in any class can invoke it. When an attribute is public, a method in any class can set its value. When an event is public, a method in any class can use the `post` statement to post it. When an event handler is public, any method in any class can register the event handler. And when a constant is public, any method in any class can reference it.

Private class elements When you define any method, attribute, event, event handler or constant, the properties dialog where you define the class element provides the option of specifying the class element as "private." A private method, attribute, event, event handler, or constant can be accessed only by the class that defines it, not by any other classes (even its own subclasses). Only methods in the current class can set the value of the private attribute, invoke a private method, post a private event, register a private event handler, or reference a private constant.

Extended Properties of Class Elements

The extended properties feature allows you to assign arbitrary name-value pairs to the individual class element. You can use the extended properties on a class element for whatever purpose you choose. For example, you might wish to use them for comments.

Restricted Property

The Restricted property for a class is intended for use when you are integrating your iPlanet UDS application with certain C routines, as described in *Integrating with External Systems*. You only use the property if your TOOL class needs to create instances of a restricted class—otherwise, ignore this property.

A C project is defined as having restricted availability because it can run only on particular hardware or software. For example if a C project encapsulates a C electronic mail routine that calls an API, the C project may be restricted because the API is available only on certain nodes in the environment. If a restricted C project is a supplier plan of a TOOL project, it is likely that one of the TOOL classes will need to create an instance of the restricted C class.

If all the classes in a project will have the restricted property, you can choose the Restricted project property in the Project Workshop. See "Restricted Availability" on page 212 for information.

Implementing Interfaces

An *interface* defines a set of class elements, without providing the code that implements them. The interface provides the method and event handler signatures that define a standard "interface" to an object. The code for the methods and event handlers in the interface is provided by the classes that *implement* the interface.

For example, the AdaptableAuction sample application defines a TaxCalculationIFace interface, which provides a method signature and an event signature related to calculating taxes on a sale. The method is called CalculateTax and the event is called TaxCalculated. The code for CalculateTax is provided by the TaxCalculationImp class, which implements the TaxCalculationIFace interface.

See Chapter 6, "Using the Interface Workshop," for background information on interfaces and instructions about using the Interface Workshop to create them.

Implementing an interface in a class means providing the code for all methods and event handlers defined in the interface. The implementing class must also define every event in the interface. You cannot exclude any of the methods, event handlers, or events, or you will get a compile error. Constants and virtual attributes do not need to be implemented in the implementing class.

A single class can implement multiple interfaces. In addition, the class may also define functionality of its own. The multiple inheritance example described in the *iPlanet UDS Programming Guide* shows classes that implement the Sortable interface to complement their basic functionality.

To implement an interface in a class, you use the Class Workshop. The Class Properties dialog for an individual class allows you to list one or more interfaces which the given class will be implementing. Any number of interfaces can be implemented by a given class. And any number of classes can implement the same interface.

Object Runtime Properties

The object runtime properties of a class affect basic behavior of the objects you create from the class. Briefly, the properties are:

Property	Description
Shared	Lets you create shared objects, which allow multiple tasks to access and change the data.
Distributed	Provides access to objects on remote partitions.
Transactional	Lets you create transactional objects, which can participate in iPlanet UDS transactions.
Monitored	For classes mapped to window widgets, automatically refreshes the display when data is updated.

The reason the object runtime properties are optional is that turning on any of the object runtime properties affects performance. Therefore, you should only turn on a property when you need the functionality for your particular application. For example, in the Art Auction application, the Painting and Artist classes do not turn on the Distributed property because they do not need to be distributed. However, the AuctionMgr class is a distributed service that is designed to be service object; therefore the Distributed property is turned on for this class.

For best performance, you should set the object runtime properties as follows:

Property	Allowed?	Subclass Override?
Shared	no	no
Distributed	no	no
Transactional	no	no
Monitored	no	no

The following sections provide information about the individual object runtime properties.

Shared Property

The Shared property determines whether or not an object of the given class can be shared.

Shared objects A shared object allows multiple tasks to access and change its data. iPlanet UDS provides the locking necessary to prevent conflicts. If one task invokes a method on a shared object in order to modify its state, iPlanet UDS does not allow any other methods to be invoked on the object until the first method completes. If one task sets the value of an attribute, iPlanet UDS locks the object until the change is complete. It is illegal for multiple tasks to operate on non-shared objects. See the *iPlanet UDS Programming Guide* for more information on shared objects.

IsShared attribute for object Turning on the Shared property for a class means that you can set the IsShared attribute for an object of the class to TRUE. The lock required for shared objects incurs some overhead, so you should set the IsShared attribute of the object on only when necessary. For information about the IsShared attribute of the Object class, see the *Framework Library online Help*.

The following settings are available for the Shared property:

Setting	Description
Disallowed	Objects of the current class cannot set the IsShared attribute to TRUE.
Allowed	Objects of the current class can set the IsShared attribute to TRUE.
Is Default	Specifies an initial setting of the IsShared attribute to TRUE. Normally, the initial setting for the IsShared attribute is FALSE. The Is Default setting does not take effect for the start class of a project. To set the IsShared attribute to TRUE for the start class, you must do so in the Init method of the start class.

Setting	Description
Subclass Override	Determines if subclasses inherit the Allowed and Is Default settings from the current class, and specifies whether the subclasses can change the setting. Turning on Subclass Override specifies that subclasses do not inherit the settings and can change their values. The subclass is initially set to the default values for the Shared property, but can then set the Shared property as appropriate. Turning off Subclass Override specifies that subclasses inherit their superclass' values and the subclass cannot change them. If you change a superclass with existing subclasses from Subclass Override on to off, all its subclasses will be changed to use the same values as the superclass Is Default setting.

Distributed Property

When you define a service for which all partitions must access the same object, iPlanet UDS allows you to turn on the Distributed property for the class. When the Distributed property is on, any changes you make to the object will be visible to all partitions in the application because all partitions are referencing the same object. In fact, if you are planning to create a service object from the class, you must turn on the Distributed property. All service objects must be distributed. In the Art Auction, the AuctionMgr class that defines the Auction Manager service is distributed because it provides an environment-wide shared service and it is therefore necessary for all partitions to access the same object.

See the *iPlanet UDS Programming Guide* for further information about distributed objects.

The following settings are available for the Distributed property:

Setting	Description
Disallowed	Distributed references are not allowed for objects of the current class.
Allowed	If an object of the current class is anchored (that is, its `isAnchored` attribute is set to TRUE), a distributed reference is allowed for an object.
Is Default	When Is Default is on, all objects of this class will be anchored (that is, the IsAnchored attribute set to TRUE) and will allow distributed references. The Is Default setting does not take effect for the start class of a project. To set the IsAnchored attribute to TRUE for the start class, you must do so in the Init method of the start class.

Setting	Description
Subclass Override	Determines if subclasses inherit the Allowed and Is Default settings from the current class, and specifies whether the subclasses can change the setting. Turning on Subclass Override specifies that subclasses do not inherit the settings and can change their values. The subclass is initially set to the default values for the Distributed property, but can then set the Distributed property as appropriate. Turning off Subclass Override specifies that subclasses inherit their superclass' values and the subclass cannot change them. If you change a superclass with existing subclasses from Subclass Override on to off, all its subclasses will be changed to use the same values as the superclass Is Default setting.

Transactional Property

The Transactional property determines whether or not an object of the given class can participate in transactions. If an object is transactional, operations that you perform on the object during a transaction will be part of the transaction. If you change the object's attributes during the transaction, the changes will be rolled back if the transaction aborts. If the object is not transactional, operations that you perform on the object will not be part of the transaction. If you change an object's attributes during the transaction, the changes will not be rolled back if the transaction aborts. See the *iPlanet UDS Programming Guide* for more information about transactional objects.

IsTransactional attribute for object Turning the Transactional property on for a class means that you can set the IsTransactional attribute for an object of the class to TRUE. For information about the IsTransactional attribute of the Object class, see the *Framework Library online Help*.

The following settings are available for the Transactional property:

Setting	Description
Disallowed	Objects of the current class cannot set the IsTransactional attribute to TRUE.
Allowed	Objects of the current class can set the IsTransactional attribute to TRUE.
Is Default	Specifies an initial setting of the IsTransactional attribute to TRUE. Normally, the initial setting for the IsTransactional attribute is FALSE.

Setting	Description
	The Is Default setting does not take effect for the start class of a project. To set the IsTransactional attribute to TRUE for the start class, you must do so in the Init method of the start class.
Subclass Override	Determines if subclasses inherit the Allowed and Is Default settings from the current class, and specifies whether the subclasses can change the setting. Turning on Subclass Override specifies that subclasses do not inherit the settings and can change their values. The subclass is initially set to the default values for the Transactional property, but can then set the Transactional property as appropriate. Turning off Subclass Override specifies that subclasses inherit their superclass' values and the subclass cannot change them. If you change a superclass with existing subclasses from Subclass Override on to off, all its subclasses will be changed to use the same values as the superclass Is Default setting.

Monitored Property

The Monitored property determines whether or not the display is refreshed when objects of the class are updated. Normally, when you make changes to the value of an object and the object is being displayed on a window, iPlanet UDS automatically refreshes the display to reflect the changes. (See "Window Classes" on page 290 for information about classes that are displayed on windows.) However, if you know that a certain class of objects will never be displayed, you can set this property off for the class to increase efficiency. See the IsMonitored attribute of the Object class in the Framework Library online Help for more information about monitored objects.

The following settings are available for the Monitored property:

Setting	Description
Disallowed	Objects of the current class will never be displayed.
Allowed	Objects of the current class may be displayed.

Setting	Description
Subclass Override	Determines if subclasses inherit the Allowed setting from the current class, and specifies whether the subclasses can change the setting. Turning on Subclass Override specifies that subclasses do not inherit the setting, but can change their values. Turning off Subclass Override specifies that subclasses inherit their superclass' values and the subclass cannot change them.

Class Versions

A *class version* is a property of a class that uniquely identifies one definition of that class. Class versions are for performing class version upgrades (rolling upgrades) of iPlanet UDS user applications that are currently deployed. This upgrade approach is recommended primarily for applications that must always be running ("high-availability" or 7 x 24 applications) or applications for which the time to distribute software to all parties may be very long (weeks).

CAUTION You should *not* use class versions without a thorough understanding of iPlanet UDS upgrading features. Upgrading an iPlanet UDS production environment requires very careful planning. See the *iPlanet UDS Programming Guide* for complete information.

Class versions and project compatibility level The *class version* uniquely identifies one definition of that class (all its attributes, methods, events, and so on). Within a project's compatibility level, a class can have multiple versions. While only one version of a class can be loaded in a single partition, a deployed application can have multiple versions of one or more classes (on different partitions) that work together. For more information about using class versions, see the *iPlanet UDS Programming Guide*. You can specify the version of a class by using the Version property on the Class Properties dialog (see "Using the Class Properties Dialog" on page 306 for information).

Class versions require hand-coded *converters* (a special type of method) so that partitions based on different class versions can communicate during the upgrade process. Converters contain code that makes up for differences between the class versions. For more information about converters, see the *iPlanet UDS Programming Guide*.

Default Init Method

When you create any class, iPlanet UDS provides a default Init method. After the Init method is created, you can modify it as desired using the Method Workshop.

The Init method is automatically invoked on an object immediately after you construct it. The default Init method simply invokes the Init method that the class inherits from its superclass.

```
super.init;
```

You can edit the default Init method to initialize attributes with values, and to create objects that you plan to reference.

Window Classes

A window class provides a window that you can display to the end user, along with methods for manipulating the window. You will use window classes in any project that has a user interface.

To create a window for your user interface, you can either create a subclass of the iPlanet UDS UserWindow class or a subclass of another window class.

UserWindow class Creating a subclass of the UserWindow class automatically provides you with an empty window, which you can then format using the Window Workshop. The UserWindow class also provides the methods you need for manipulating your window (for example, to open it or close it). For details, see the Display Library online Help.

Inherited windows Creating a subclass of another custom window class (rather than the UserWindow class) creates an inherited window. An inherited window inherits its initial appearance from its superclass window. You can extend the inherited form and menu bar by adding new widgets to the subclass window in the Window Workshop.

Note that you can also use either kind of window class as a page template for printing. See "About Windows as Page Templates" on page 378 for information about page templates.

Window attributes When you format the window in the Window Workshop, you add one or more widgets to the window's form and menu bar (a widget is a window control, such as a radio box or a push button). Adding a widget to the window is equivalent to creating a new attribute for the class. The name that you assign to the widget in the widget's properties dialog becomes the attribute name that you can use to reference the widget. For example, if you add a push button to a form called "OKButton," this adds an OKButton attribute to your window class that has the type PushButton. When you examine a window class in the Class Elements browser, iPlanet UDS displays both the window attributes and the other attributes.

Window attributes and data attributes The window attributes in a user window class have a special relationship to the attributes that contain the data that is displayed in the window. To display data in a widget, iPlanet UDS maps the attribute that contains the data and the window attribute that displays the data together by name. For example, if you want your text field to display the artist's name for a painting, you name your text field widget "Artist" because you already have a TextData attribute called "Artist" that stores the artist's name. iPlanet UDS then displays the current value of the Artist attribute in the Artist text field widget. When you examine a window class in the Class Elements browser, you can use the workshop filter to display the window attributes for the class in a separate list. See "Examining a Class" on page 296 below for further information on this.

Display method To display a window, you normally write a Display method. As a convenience, iPlanet UDS provides a default Display method, which you can modify to suit your needs. Typically, the Display method for a window initializes the data for the window and displays it on the end user's screen. In addition, the Display method includes the `event loop` statement for the window, which handles all the events on the window. The default Display method contains the following code:

Code Example 5-1 Default Display Method

```
self.Open();
event loop
  when task.Shutdown do
    exit;
end event;
self.Close();
```

See the *iPlanet UDS Programming Guide* for information about writing the Display method.

Domain Classes

A domain class combines a DataValue subclass with an iPlanet UDS widget. A domain class is always the subclass of an iPlanet UDS nullable DataValue subclass, which determines the type of data stored in the domain object, and it is always associated with a widget class, which specifies the widget used to display the domain object's data. Domain classes are intended mainly for use with the optional Business Model Workshop, where you use them to assign the type of an attribute (see *A Guide to Express* for information). However, you can also use domains as customized widget types in the Window Workshop.

DataValue subclass When you create a domain class, you must specify one of the iPlanet UDS nullable DataValue subclasses as its superclass. The domain's superclass determines the type of data stored in the object and provides methods for manipulating it.

The following table describes the classes that you can use to create a domain:

DataValue Subclass	Description
IntegerNullable	Stores and manipulates whole numbers of any size, and the NULL value.
TextNullable	Stores and manipulates character data, and the NULL value.
DoubleNullable	Stores and manipulates floating point data, and the NULL value.
BooleanNullable	Stores and manipulates logical values of TRUE and FALSE, and the NULL value.
BinaryNullable	Provides access to database BLOB types, and the NULL value.
DateTimeNullable	Stores and manipulates date-time data, and the NULL value.
DecimalNullable	Stores and manipulates floating point data to a specific precision of up to thirty decimal places, and the NULL value.
ImageNullable	Stores and manipulates image data in a standard, portable format, and the NULL value.
IntervalNullable	Stores and manipulates intervals of time, and the NULL value.

See the Framework Library online Help for detailed information on the nullable DataValue subclasses.

A domain class could define a numeric value that needs special validation and formatting, such as a credit card number, salary, or phone number. It could also define a predefined list of text values, such as a list of three marital states (Married, Single, and Divorced), that you want to use throughout your application. iPlanet UDS Express provides several predefined domains, including IntegerDomain, DateDomain, and MoneyDomain.

Default SetValue method When you create a domain class, you can request that iPlanet UDS provide a default SetValue method, which provides automatic data validation and specialized input conversion. After the domain is created, you can open its SetValue method in the Method Workshop to examine it or modify it as desired. See the Framework Library online Help for information about the SetValue method.

Default FillString method You can also request that iPlanet UDS provide a default FillString method, which converts the data in the domain into string form. After the domain is created, you can open its FillString method in the Method Workshop to examine it or modify it as desired. See the Framework Library online Help for information about the FillString method.

Form widget Every domain class is associated with a widget class, called its *form widget*. The form widget class specifies the class of widget, such as DataField or RadioList, that is used to display the data when the domain is added to a form. When you create the domain class, you must choose one of the widget classes that is appropriate for the data type. For example, imagine you have MaritalStatus domain, which is a subclass of IntegerNullable. When the MaritalStatus domain is displayed on the form, you can request that it be displayed as a radio list.

When you specify the form widget for the domain, you can set many of the individual properties for the widget, including its size properties and help topic. For example, for a radio list, you could specify the text values for each of the radio buttons in the list. For MaritalStatus domain, you can specify the text values of Married, Single, and Divorced as part of the form widget properties.

Array widget Optionally, the domain class can also be associated with an *array widget*. The array widget specifies the widget that is used when the domain is displayed within an array field. By default, the array widget is the same as the form widget. However, because arrays need to display information in a condensed form, you may wish to specify a different widget type for this case. For example, if the form widget for MaritalStatus is a radio list, you may wish to specify a drop list as the array widget, because a drop list would look much more appropriate in the array field.

When you specify the array widget for the domain, you can set many of the individual properties for the widget, including its size properties and help topic. For example, for the drop list, you could specify the text for each item in the list.

Using the Class Workshop

You enter the Class Workshop from the Project Workshop either by opening an existing class or by creating a new class.

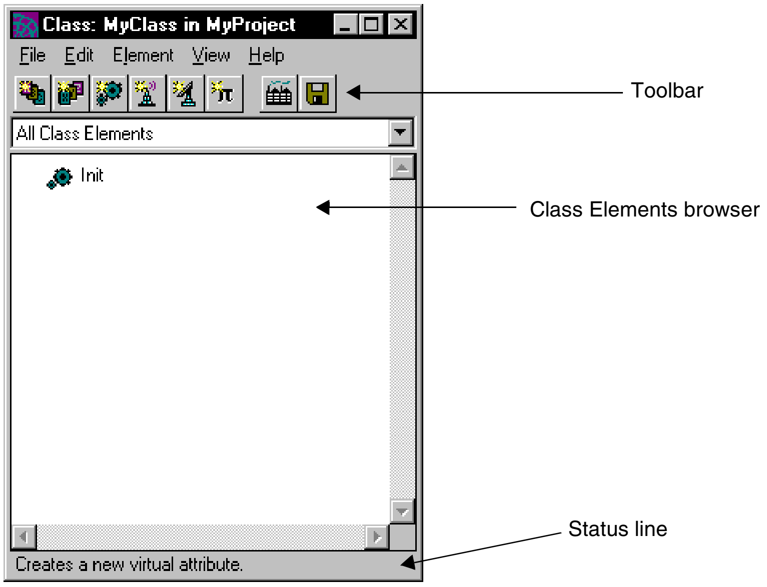
Opening an existing class If you wish to examine or edit an existing class, double-click the class name, or click the class name and choose the Component > Open command.

Creating a new class If you wish to create a new class, click the New Class, New Window Class, or New Domain Class button, or choose the Component > New Nonwindow, New Window Class, or New Domain Class command.

The Class Workshop Window

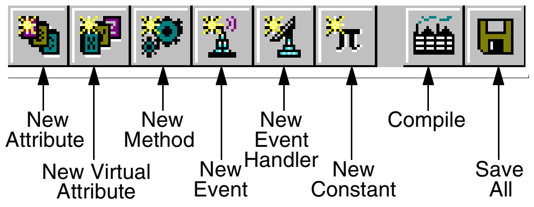
The Class Workshop window, shown in Figure 5-2, consists of three parts: the Class Elements browser, the status line, and the toolbar, as shown in Figure 5-3.

Figure 5-2 Class Workshop Window



Creates a new virtual attribute.

Figure 5-3 Class Workshop Toolbar



View Menu

The View menu in the Class Workshop provides the following toggles to control which parts of the workshop are displayed:

Command	Function
Toolbar	Makes the toolbar visible or invisible.
Kind Icon	When this toggle is turned on, the kind icons are displayed in the Class Elements browser. When turned off, the kind icons are not displayed.
Status Line	Makes the status line visible or invisible.

Command	Function
Inherited	When this toggle is turned on, the Class Elements browser displays the inherited class elements. When turned off, the inherited class elements are not displayed.
Converter	When this toggle is turned on, the Class Elements browser displays the converter methods defined for the class. When turned off, the converter methods are not displayed.

Leaving the Class Workshop

To leave the Class Workshop, use the Close command to close the workshop or use the system close box to close the window. This closes only the current workshop.

Examining a Class

If the class you wish to examine is not already displayed, you have the following options. You can:

- use the Project Workshop to open a class in the current project
- use the Open Superclass command in the Class Workshop to open one of the superclasses for the current class
- use the Find Class/Interface… command in the Class Workshop to search for the specified class in the current project and all its supplier plans

➤ **To examine a class from the Project Workshop**

1. In the Project Components browser, double-click the class name.

   You can also select the class name, and choose the Component > Open command.

➤ **To examine a class with the Open Superclass command**

1. Choose the File > Open Superclass command.
2. On the Open Superclass submenu, select the superclass you wish to open.

➤ **To examine a class with the Find Class/Interface… command**

1. Choose the Edit > Find Class/Interface… command.
2. In the Find Class/Interface dialog, specify the class name and click the OK button.

The Class Workshop displays information about attributes, methods, events, event handlers, and constants directly on the main workshop window.

The class properties are not displayed on the main workshop window. To view them, you must use the appropriate command. The next sections provide detailed information about how to examine each of the class elements. This is followed by information about viewing the class properties, and information about viewing the window and menu for a window class.

A special feature of the Class Workshop is the ability to use the Find Text… command to search for any source code in the class that contains a specified string. See "Modifying a Class" on page 322 for information.

Examining the Class Elements

By default, the Class Workshop displays all class elements. A kind icon by each component name indicates the kind of element.

Icon	Class Element
	Attribute
	Virtual attribute
⚙	Method
	Event
	Event handler
π	Constant

To turn off the kind icons, switch off the View > Kind Icon toggle.

Filter drop list To view a list of only one kind of class element, use the filter drop list for the Class Workshop. The filter drop list allows you to choose a single class element, such as "Attributes" or "Events," or "All Class Elements."

Note that you can set your filter preferences for the workshop by using the Workshop Preferences command. These preferences are saved as part of your current workspace. See "Setting Workshop Preferences" on page 329 for information.

Sorting By default the list of elements is in alphabetical order, sorting all class elements by name. The View menu provides the following commands for sorting the class elements:

Command	Description
Sort By Name	The default. Sorts the elements by name.
Sort by Kind	Sorts the elements by kind, attribute, event, method, and event handler.
Sort by Datatype	Sorts the elements by their data types.
Sort by Superclass	For inherited elements, sorts the elements by their superclass.

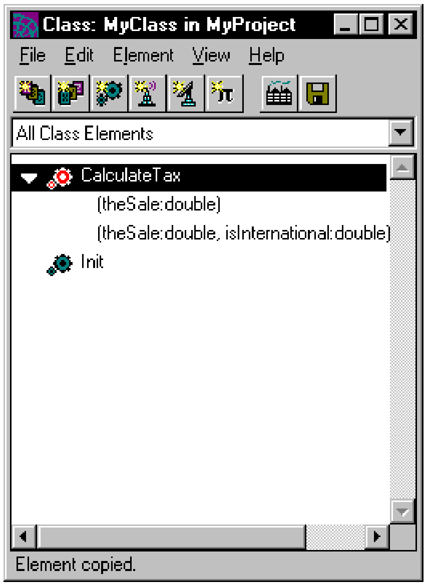
Inherited command By default the Class Elements browser shows only the elements defined specifically for the selected class. To view the elements inherited by the class from its superclasses, choose the View > Inherited command.

Examining Methods

To display only the methods for the class, choose "Methods" from the filter drop list in the Class Elements browser. For methods, the browser displays the name and return type for each method.

To display the complete definition of the method, double-click on the method name, or select the method name and choose the Element > Open command. This command opens the Method Workshop, which displays the original method definition, including the method source code. See Chapter 10, "Using the Method Workshop," for information about using the Method Workshop.

Overloaded methods If the method is overloaded, the Class Workshop displays the method name as a folder, and opens the folder to display the individual method signatures for all the methods that share the same name. To display the complete definition of an individual method, double-click the signature for the particular method you wish to examine.



CalculateTax
(theSale:double)
(theSale:double, isInternational:double)
Init

Element copied.

Converters By default, converter methods are not displayed in the Class Workshop. To display the converters in the current class, use the View > Converters command. You can also use the View Converters option in the Class Workshop Preferences dialog to specify that converters be displayed in the Class Workshop browser. See "Viewing Preferences" on page 137 for information.

To display the complete definition of an individual converter, double-click on the converter name or use the Element > Converter command as follows.

➤ To view converter methods

1. Select the method or event for which you wish to see converters.
2. Select Element > Converter.
3. If you have converters defined for the current method or event, you will see the Open for New Method and Open for Obsolete Method commands on the slide-off menu.

4. Choose either the Open for New Method or Open for Obsolete Method commands, depending on which converter you want to see. If you have no converters defined, the slide-off menu shows “Create” instead of “Open.”

Examining Attributes

To display only the attributes for the class, choose “Attributes” from the filter drop list in the Class Elements browser. This displays both nonwindow and window attributes. For attributes, the browser displays the name and data type for each attribute.

To display the complete definition of an individual attribute, double-click on the attribute name, or select the attribute name and choose the Element > Open command. For nonwindow attributes, this command opens the Attribute Properties dialog, which displays the original definition of the attribute. For window attributes, this command opens the Window or Menu Workshop, and highlights the selected attribute.

Window attributes To display only the window attributes for the class, choose “Window Attributes” from the filter drop list in the Class Elements browser.

Nonwindow attributes To display only the nonwindow attributes for the class, choose “Nonwindow Attributes” from the filter drop list in the Class Elements browser.

Examining Events

To display only the events for the class, choose “Events” from the filter drop list in the Class Elements browser. For events, the browser displays the name of each event.

To display the complete definition of an individual event, double-click on the event name, or select the event name and choose the Element > Open command. This command opens the Event Properties dialog, which displays the original definition of the event.

Examining Event Handlers

To display only the event handlers for the class, choose “Event Handlers” from the filter drop list in the Class Elements browser. For event handlers, the browser displays the name for each event handler.

To display the complete definition of the event handler, double-click on the event handler name, or select the event handler name and choose the Element > Open command. This command opens the Event Handler Workshop, which displays the original event handler definition, including the handler source code. See Chapter 11, “Using the Event Handler Workshop,” for information about using the Event Handler Workshop.

Examining Constants

To display only the constants for the class, choose “Constants” from the filter drop list in the Class Elements browser. For constants, the browser displays the name and value for each constant.

To display the complete definition of an individual constant, double-click on the constant name, or choose the constant name and give the Element > Open command. This command opens the Constant Properties dialog, which displays the original definition of the constant.

Examining Class Properties

To display the class properties, choose the File > Properties… command. This command opens the Class Properties dialog, which displays the current settings for the class properties.

The Class Properties dialog contains three tab pages: General, Runtime, and Interfaces Implemented. For information on the individual properties on these pages, see “Using the Class Properties Dialog” on page 306.

Examining Extended Properties for Class Elements

To display the extended properties for an individual class element, select the class element and choose the Element > Extended Properties... command. The Extended Properties dialog opens, displaying the current settings for the extended properties.

Examining Window Classes

For window classes, the Class Workshop allows you to view only the methods, attributes, events, event handlers and constants for the class. To view the window or menu for a window class, you must open the Window or Menu Workshops, respectively.

Opening the Window Workshop To open the Window Workshop for the class, choose the File > Window… command or click the Window Workshop button on the toolbar. See Chapter 7, "Using the Window Workshop," for information about using the Window Workshop.

Opening the Menu Workshop To open the Menu Workshop for the class, choose the File > Menu… command, or click the Menu Workshop button on the toolbar. See Chapter 9, "Using the Menu Workshop," for information about using the Menu Workshop.

Creating a Class

To create a class, you must start from the Project Workshop. The Project Workshop allows you to create three kinds of classes: nonwindow classes, window classes, and domain classes. The following sections provide detailed instructions for creating each kind of class.

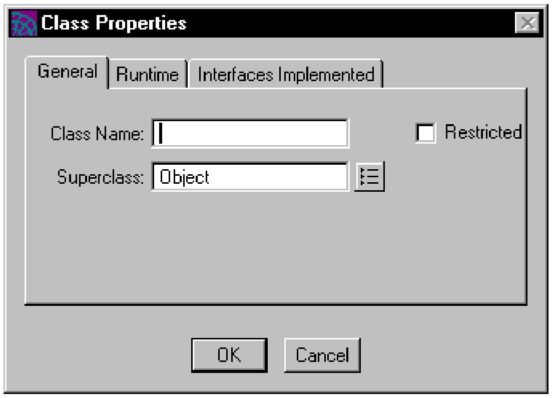
Creating a Nonwindow Class

The New > Nonwindow Class command creates a new standard class with the name and superclass you specify.

➤ **To create a nonwindow class**

1. In the Project Workshop, choose the Component > New > Nonwindow Class command, or click the New Nonwindow Class button on the toolbar.

   The Class Properties dialog opens.



Class Name:

Superclass: Object

2. On the General tab page, specify the name and superclass for the class.
3. Set any other properties for the class as described under "Using the Class Properties Dialog" on page 306.
4. Click the OK button to create the class.

After you create the class, iPlanet UDS opens the Class Workshop, where you can define the class elements.

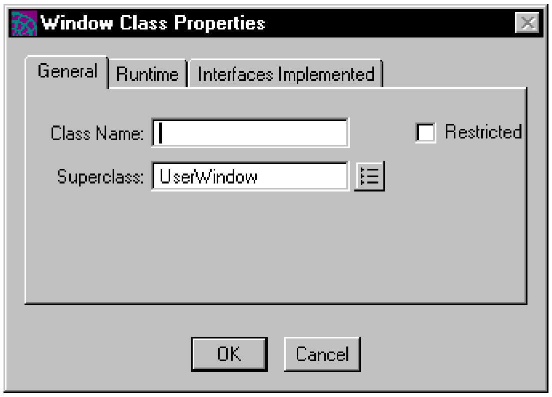
Creating a Window Class

The New > Window Class command creates a new window class, using the name you specify.

➤ **To create a window class**

1. In the Project Workshop, choose the Component > New > Window Class command, or click the New Window Class button on the toolbar.

   The Window Class Properties dialog opens.



Class Name:

Superclass: UserWindow

☐ Restricted

2. On the General tab page, enter the class name.

   The superclass "UserWindow" is filled in automatically. If you wish to create an inherited window, replace the value "UserWindow" with the name of the appropriate window superclass.

3. Set any other properties for the class as described under "Using the Class Properties Dialog" on page 306.

4. Click the OK button to create the class.

After you create the window class, iPlanet UDS opens the Class Workshop, where you can define the class elements.

Creating the window To create the window for the class, choose the File > Window… command or click the Window Workshop button on the toolbar. See Chapter 7, "Using the Window Workshop," for information about using the Window Workshop.

Creating the menu bar To create the menu bar for the class, choose the File > Menu… command or click the Menu Workshop button on the toolbar. See Chapter 9, "Using the Menu Workshop," for information about using the Menu Workshop.

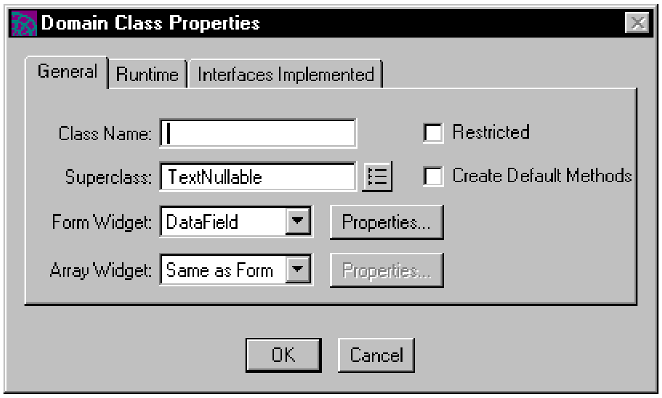
Creating a Domain Class

The New > Domain Class command creates a new domain with the name, superclass, and widget you specify.

➤ **To create a domain class**

1. In the Project Workshop, choose the Component > New > Domain Class command, or click the New Domain Class button on the toolbar.

The Domain Class Properties dialog opens.



Domain Class Properties

General | Runtime | Interfaces Implemented

Class Name:

Superclass: TextNullable

Form Widget: DataField

Array Widget: Same as Form

☐ Restricted

☐ Create Default Methods

2. On the General tab page, specify the name and superclass for the domain. The superclass can be any nullable DataValue subclass.

3. Choose the form widget for the domain and, if desired, set its properties by clicking the Properties button.

4. If you want the array widget to be different than the form widget, choose the array widget type and, if desired, set its properties by clicking the Properties button.
5. Set any other properties for the class as described under "Using the Class Properties Dialog" on page 306.
6. Click the OK to button create the class.

After you create the domain class, iPlanet UDS opens the Class Workshop, where you can define the class elements.

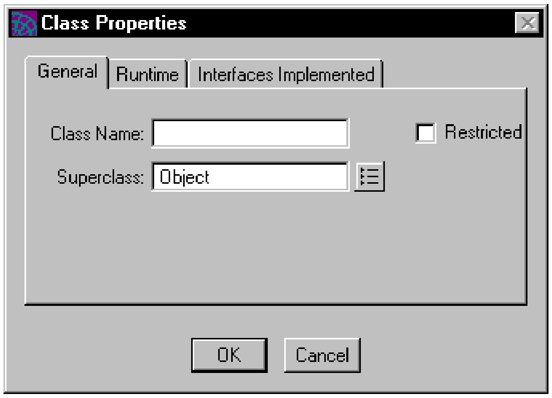
Using the Class Properties Dialog

The Class Properties dialog contains a tab folder with the three tab pages: General, Runtime, and Interfaces Implemented.

General Page

The General tab page contains the basic properties you must set in order to define the class. For domain classes, the General tab page allows you to set the form and array widgets associated with the domain.

Figure 5-4 General Tab Page on Class Properties Dialog



Class Name:
Superclass: Object
Restricted

Class Name Type the class name into this field. The class name can be any legal iPlanet UDS name that is unique for the project.

Superclass Type the name of an existing superclass, or use the browser button to display a list of classes from which you can make a selection.

Restricted To make the class restricted, set this toggle to on.

Create Default Methods For domain classes, you can request that default SetValue and FillString methods be created for the class by setting this toggle to on.

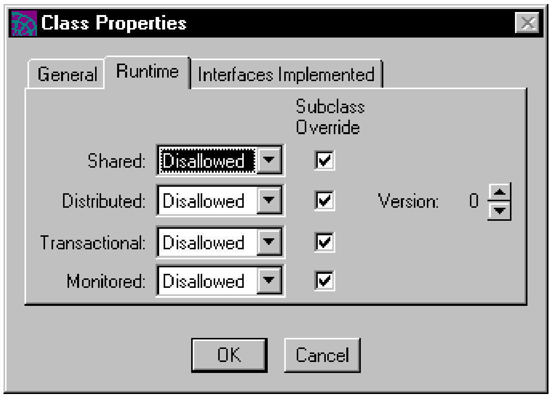
Form Widget For domain classes, choose the form widget type from the drop list. To set the form widget's properties, click the Properties button.

Array Widget For domain classes, choose the array widget type from the drop list. To set the array widget's properties, click the Properties button.

Runtime Page

The Runtime page contains the object runtime properties and the Version property. Because the defaults for the object runtime properties are suitable for most situations, most users do not need to set these properties at this point. See "Object Runtime Properties" on page 284 for details about using the object runtime properties most efficiently.

Figure 5-5 Runtime Page of Class Properties Dialog



Shared: Disallowed ☑
Distributed: Disallowed ☑
Transactional: Disallowed ☑
Monitored: Disallowed ☑
Version: 0

Shared To change the setting, use the drop list to select: Disallowed, Allowed, Is Default, or Subclass Override.

Distributed To change the setting, use the drop list to select: Disallowed, Allowed, Is Default, or Subclass Override.

Transactional To change the setting, use the drop list to select: Disallowed, Allowed, Is Default, or Subclass Override.

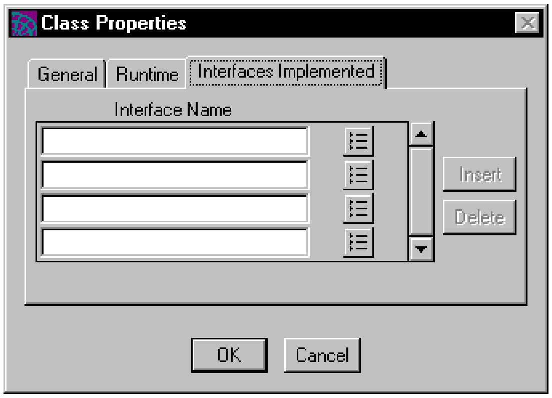
Monitored To change the setting, use the drop list to select: Disallowed, Allowed, or Subclass Override.

Version A version is zero (0) by default. You can set the version of a class to a higher number by clicking the up arrow to increment the version number. Note that you should use class version numbers only when you are performing a class version upgrade of a deployed iPlanet UDS application. Please see the *iPlanet UDS Programming Guide* for information about upgrading applications and using class versions.

Interfaces Implemented Page

The Interfaces Implemented page allows you to specify which interfaces the current class is implementing

Figure 5-6 Interfaces Implemented Page



Enter the name of the interface you wish to implement in the Interface Name field, or click the Browser button and choose the interface.

If you enter the name of the interface manually, use the following syntax:

project.interface

Once you implement an interface in a class, you must implement in your class all methods, event handlers, and events defined in the interface. See the *iPlanet UDS Programming Guide* for complete information on implementing an interface.

Defining Class Elements

To define the elements in the class, you can either create new elements or copy existing elements from another class or from an interface. You can copy these elements using the Edit > Copy, Cut, and Paste commands (described under "Using the Clipboard" on page 323), or you can copy them using drag and drop. This is especially useful for methods and event handlers; when you copy a method or event handler from one class to another, all the source code is copied along with the basic method or event handler definition.

Copying Interface Elements

Note that copying interface elements from an interface to a class is particular useful when the class is implementing the interface. By copying the methods, event handlers, and events defined in the interface to the class, you are automatically "implementing" all the necessary class elements in the interface. After that, all you have to do is write the method and event handler code.

Using drag and drop You can copy an existing interface or class element by dragging it from the class or interface that defined it and dropping it on the current class. Of course, if you drag a method or event handler from an interface and drop it onto an class, there will be no code associated with the method or event handler. You must write the method or event handler code within your class definition.

➤ **To drag and drop an class element**

1. In the Workshop where the existing element was originally defined, select the element you wish to copy.
2. Drag the element to the Class Elements browser for the new class you are creating.
3. Drop the element onto the Class Elements browser.

The following sections describe how to create new class elements using the Class Workshop. "Setting Extended Properties for Class Elements" on page 321 describes how to set the extended properties for an interface element.

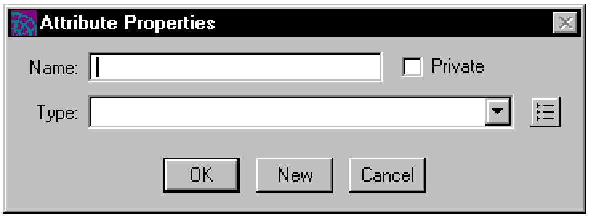
Defining Attributes

To define an attribute, use the New Attribute command, or click the New Attribute button on the toolbar.

➤ **To create an attribute**

1. Choose the Element > New Attribute command, or click the New Attribute button on the toolbar.

   The Attribute Properties dialog opens.

2. In the Attribute Properties dialog, specify the name and type for the attribute.
3. Click the OK button to add the attribute to the class and close the dialog, or click the New button to create the attribute and leave the dialog open so you can create another attribute.

Fill in the fields on the dialog as follows:

Property	How to specify
Name	Type the attribute name.
Type	Choose the data type from the drop list, or use the browser button to select a class name for the type.
Private	Click this toggle on to make the attribute private.

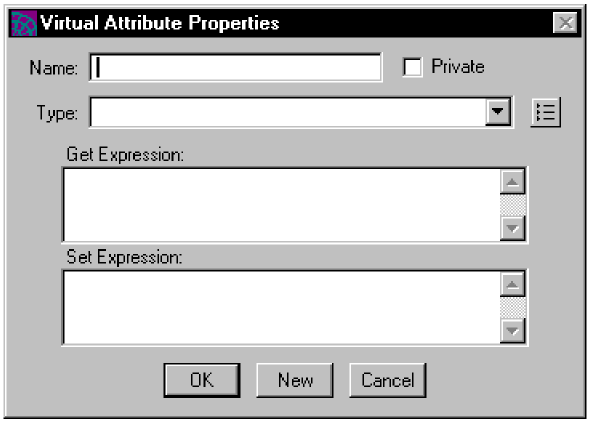
Defining Virtual Attributes

To define a virtual attribute, use the New Virtual Attribute command, or click the New Virtual Attribute button on the toolbar.

➤ **To create a virtual attribute**

1. Choose the Element > New Virtual Attribute command, or click the New Virtual Attribute button.

   The Virtual Attribute Properties dialog opens.



2. In the Virtual Attribute Properties dialog, specify the name and type for the virtual attribute, as well as the Get and Set expressions (described below).

3. Click the OK button to add the attribute to the class and close the dialog, or click the New button to create the attribute and leave the dialog open so you can create another virtual attribute.

Fill in the fields on the dialog as follows:

Property	How to specify
Name	Type the attribute name.
Type	Choose the data type from the drop list, or use the browser button to select a class name for the type.
Private	Turn on this toggle to make the attribute private.
Get	Enter a TOOL expression to be executed when the program accesses the value of the attribute (see below for further information). The Get expression is required.
Set	Enter a TOOL expression to be evaluated when the program assigns a value to the attribute (see below for further information). The Set expression is optional. A virtual attribute without a Set expression is a read-only attribute.

NOTE If the value of the virtual attribute is being displayed on a window, the value is calculated when the window is first opened and, unlike a spread sheet, is not automatically refreshed if the value of the Set expression changes. Instead, you must update the data on the window from your TOOL code using the UpdateDataFromField method (see the Display Library online Help).

To illustrate virtual attributes, we use the following example Weather class, which provides the estimated low temperature in both Fahrenheit and centigrade. Note that this example shows the class definition as you would write it using Fscript, rather than using the Class Workshop.

Code Example 5-2 Virtual Attribute Definition Example

```
class weather inherits from object
  has public
    method setctemp(temp : integer) :void;
    lowf: integer;
    virtual lowc : integer =
      (get = (5.0/9.0) * (lowf -32),
      (set = SetCTemp(lowc));
end class;

method weather.setctemp(temp : integer) : void
  begin
    self.lowf = (9.0/5.0)* temp + 32;
end;
```

Set Expression property In the Set Expression field, enter the TOOL expression to be evaluated when the program assigns a value to the attribute. (The Set expression is optional. A virtual attribute without a Set expression is a read-only attribute.)

The Set expression for a virtual attribute usually invokes a method to update some data. However, this can be any expression, as long as the expression data type is compatible with the virtual attribute's data type.

The expression can reference any elements defined in the current class and any of its superclasses, or any project components. iPlanet UDS assumes that all references to attributes, methods, and so on, are for the current object. Because this is just an expression and not a statement, do not use a semicolon at the end.

In the Set expression, you can use the virtual attribute name to represent the value that the user assigned to the attribute. Typically, the expression contains a method that uses the virtual attribute name as one of its parameters. For example, the following set expression uses the LowC virtual attribute as a parameter in the SetCTemp method.

```
SetCTemp(lowc)
```

(Note that you cannot update the value of an attribute by setting its value directly to the value of the virtual attribute; instead, you must invoke a method to update it.)

If the Set expression produces a value, such as a return value from a method, iPlanet UDS ignores it.

Get Expression property In the Get Expression field, enter a TOOL expression to be executed when the program accesses the value of the attribute. The value of the expression is the value of the virtual attribute. This can be any expression with a data type that is compatible with the attribute's data type.

For a private virtual attribute, the expression can reference any elements defined in the current class and its superclasses, or any globally defined components, such as service objects and project constants. For a public virtual attribute, the expression can reference only public attributes, methods, events, and event handlers. iPlanet UDS assumes that all references to attributes, methods, and so on are for the current object. Because this is just an expression and not a statement, do not use a semicolon at the end.

The following example illustrates the get expression from the Lowc virtual attribute:

```
(5.0/9.0) * (lowf - 32)
```

In the Get expression, you can use the virtual attribute name as an input-output or output parameter for a method (see *TOOL Reference Guide* for information about input-output and output parameters). iPlanet UDS uses this "return" value as the value for the virtual attribute. Any other use of the virtual attribute name is illegal.

See the *TOOL Reference Guide* for information about how to write TOOL expressions.

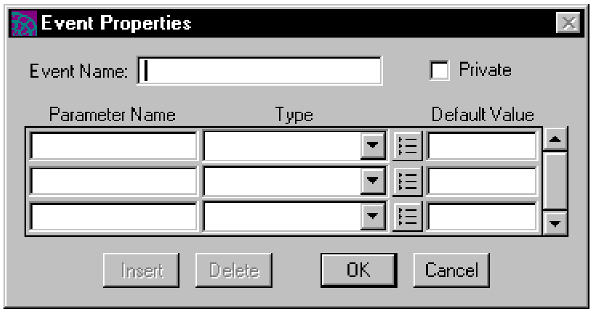
Defining Events

To define a new event, use the New Event command, or click the New Event button on the toolbar.

➤ To create an event

1. Choose the Element > New Event command, or click the New Event Button.

   The Event Properties dialog opens.



2. On the Event Properties dialog, enter the event name.

   The event parameters are optional. See below for information on specifying the event parameters.

3. Click the OK button to add the event to the class.

Event parameters The Event Properties dialog displays the event parameters in an array field. To add a parameter to the list, add a new row to the array field. Fill in the columns as follows:

Column	How to fill it in
Name	Type the parameter name.
Type	Choose a data type from the drop list or type in a class name.
Default Value	If desired, enter a value that is compatible with the parameter's data type. For parameters with a class data type, the default value must be NIL.

Delete and Insert buttons You can add as many parameters to the event as you wish. If you wish to add a parameter in the middle of the list, use the Insert button to insert a new row above the row you select. If you wish to delete a parameter from the list, use the Delete button to remove the currently selected row from the array field.

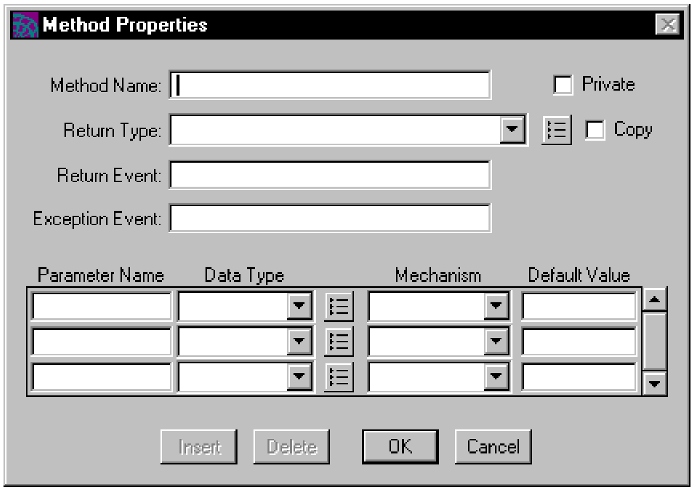
Defining Methods

To define a method, use the New Method command, or click the New Method button on the toolbar.

➤ To create a new method

1. Choose the Element > New Method command, or click the New Method button.

   The Method Properties dialog opens.



2. In the Method Properties dialog, enter the method's name in the Method field. The return type, return event, exception event, and parameters are optional.

3. Click the OK button to add the method to the class and open the Method Workshop.

See Chapter 10, "Using the Method Workshop," for information on the Method Properties dialog and the Method Workshop.

The following sections provide information about overloading methods and creating converter methods.

Overloading Methods

To overload a method, you simply create a new method with the same name as the method you wish to overload, but with a different parameter list.

When more than one method with the same name exists—with different parameter lists—iPlanet UDS automatically overloads the method.

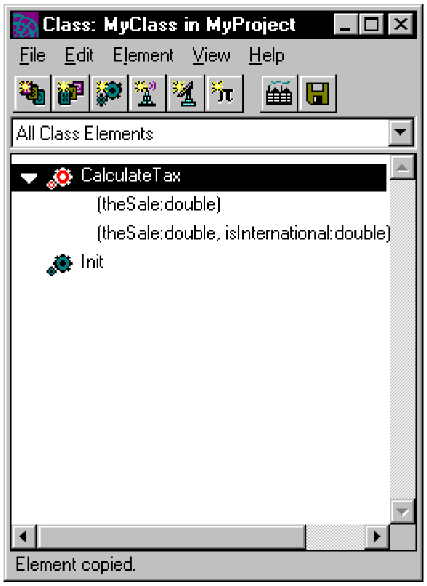
➤ To overload an existing method

1. Create a new method with the New Method button, or choose the Element > New Method command.

   The Method Properties dialog opens.

2. On the Method Properties dialog, enter the same method name as the original method, but specify a different parameter list (the return type and return events can also be different).

   The Class Workshop displays all the method signatures for the overloaded method.



```
CalculateTax
    (theSale:double)
    (theSale:double, isInternational:double)
Init
```

Element copied.

Deleting a single method signature To delete a single method signature for an overloaded method, simply highlight the individual signature and choose the Edit > Delete command.

Creating a Converter Method

You can create a converter method for any class whose Distributed property is set to Allowed. If the Distributed property is not set to Allowed, you will not be able to create a converter.

➤ **To create a method converter**

1. In the Class Workshop, select the method requiring a converter.
2. Choose the Element > Converter command. If the current method has no converters defined, you will see two options: Create for New Method and Create for Obsolete Method.
3. After you choose the appropriate type of converter, the Method Workshop opens. The name and parameters of the converter are automatically derived; you cannot change them.
4. Write the converter code using TOOL.

See the *iPlanet UDS Programming Guide* for details about writing converter code.

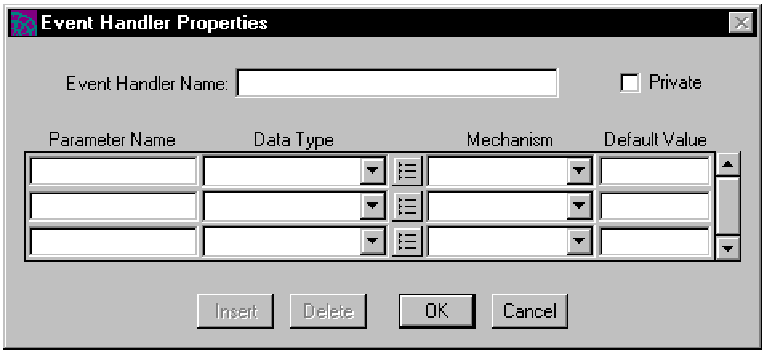
Defining Event Handlers

To define a handler, use the New Event Handler command, or click the New Event Handler button on the toolbar.

➤ To create a new event handler

1. Choose the Element > New Event Handler command, or click the New Event Handler button.

   The Event Handler Properties dialog opens.



2. In the Event Handler Properties dialog, enter the event handler's name in the Handler field. The parameters are optional.

3. Click the OK button to add the handler to the class and open the Event Handler Workshop.

See Chapter 11, "Using the Event Handler Workshop," for information on the Event Handler Properties dialog and the Event Handler Workshop.

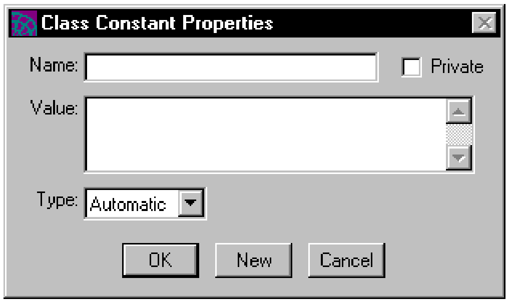
Defining Class Constants

To define a constant, use the Element > New Constant command, or click the New Constant button on the toolbar.

➤ To create a constant

1. Choose the Element > New Constant command.

2. The Constant Properties dialog opens.



3. In the Constant Properties dialog, specify the name, type, and value for the constant. The constant types are described below.

4. Click the OK button to add the constant to the class or the New button to create another constant.

Constant types The constant types are:

Constant Type	Description
Automatic	iPlanet UDS determines the type based on the constant value (see below). Use the Automatic type for strings.
Boolean	Allows a value of TRUE or FALSE.
Double	Allows a floating point number.
Integer	Allows a positive or negative whole number.
String	Allows an alphanumeric string.

Automatic type For an automatic type, the value you specify determines the type of the constant. The types are:

Value	Type
TRUE, FALSE	boolean
Positive or negative whole number	integer
Floating point number	float
Character data (without quotes)	string

Single quotes for strings Normally, you do not need to enclose a string value in single quotation marks. However, if you wish to create a string constant with a value of an integer or a floating point, you must use single quotes. In addition, because iPlanet UDS automatically truncates trailing spaces, you need to use single quotes to specify a string value with trailing spaces.

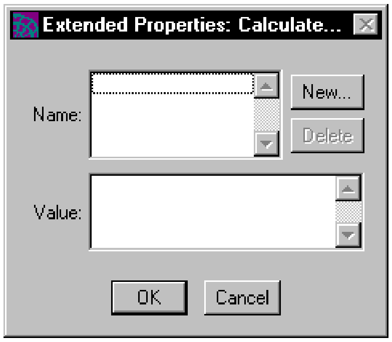
Setting Extended Properties for Class Elements

To set the extended properties on an individual class element, use the Element > Extended Properties... command.

➤ **To set extended properties for a class element**

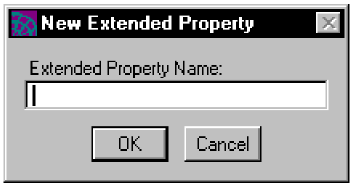
1. Select the element for which you wish to set extended properties.
2. Choose the Element > Extended Properties... command.

   The Extended Properties dialog opens.



3. Click the New... button.

   The New Extended Property dialog opens.



4. Enter the name of the property you wish to create and click OK.
5. In the Value field, enter the value of the extended property.

6. To enter additional extended properties, repeat Step 3 through Step 5.
7. Click OK.

Modifying a Class

Before modifying a class, you must have write access to it. You have write access to a class if you have just created it (and have not yet integrated your workspace), or if you have checked out or branched the class. In the Project Workshop, you can use the View > Writeable Icon command to see if you have write access to a class. If you do not have write access to the class, you must either check it out or branch it before you can modify it. See "Write Access to Project Components" on page 257 for information.

In the Class Workshop, you can modify a class by modifying the definition of an element, by adding an element, by deleting an element, or by changing the class properties. The Edit menu in the Class Workshop provides commands that let you use the clipboard to copy or move class elements from one class to another. The Find Text… and Replace Text… commands on the Edit menu let you find all occurrences of a specified string within the method source code for the class and to make global replacements.

Updating Class Elements

To update a class element, double-click the element name in the Class Elements browser, or select the element name and choose the Element > Open… command. In the case of attributes, events, and constants, this command opens the dialog in the Class Workshop where the item was originally defined. In the case of methods and event handlers, this command opens the appropriate workshop, which displays the original definition of the method or event handler.

Deleting Class Elements

The Edit > Delete command lets you delete class elements from the Class Elements browser.

➤ **To delete a class element**

1. Select the element you wish to delete by clicking on the name.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the class element.

Updating Class Properties

To modify the class properties, choose the File > Properties… command. This command opens the Class Properties dialog, where you can make your updates. See "Using the Class Properties Dialog" on page 306 for information about how use the Class Properties dialog.

Updating Extended Properties for Class Elements

To modify the extended properties for an individual class element, select the class element and choose the Element > Extended Properties... command. This command opens the Extended properties dialog, where you can add, delete, or change the extended properties.

Using the Clipboard

The Edit menu in the Class Workshop allows you to cut or copy a class element onto the clipboard the same way you cut or copy text in a text editor. Once the class element is in the clipboard, you can paste it into another class definition or an interface definition.

The commands on the Edit menu for using the clipboard are:

Command	**Description**
Cut	Removes the class element from the current class and copies to the clipboard.
Copy	Copies the selected element onto the clipboard.
Paste	Pastes the element in the clipboard into the current class.

Finding and Replacing Text

The Edit > Find Text… and Replace Text… commands in the Class Workshop let you search for all occurrences of a specified string within the method and event handler source code for the class.

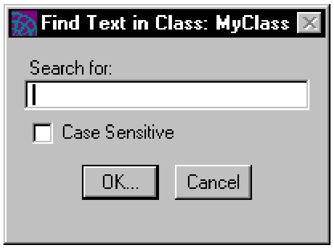
Finding Text

The Find Text… command finds all occurrences of the specified string in the method and event handler source code in the class.

➤ **To find a string**

1. Choose the Edit > Find Text… command.

   The Find Text in Class dialog opens.

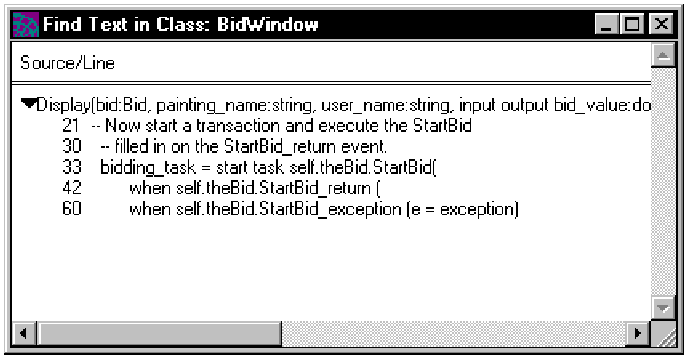
   

2. In the Find Text in Class dialog, specify the string you wish to search for.

   The default is a case-insensitive search. If you want a case-sensitive search, set the Case Sensitive toggle to on.

   Click the OK button to start the search.

3. When the search is complete, the Find Text in Class dialog displays the method or event handler name, and the line itself for each source code line that contains the string.



```
Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:do
    21  -- Now start a transaction and execute the StartBid
    30   -- filled in on the StartBid_return event.
    33   bidding_task = start task self.theBid.StartBid(
    42        when self.theBid.StartBid_return (
    60        when self.theBid.StartBid_exception (e = exception)
```

4. Double-click any line in the Find Text in Class dialog to open the Method or Event Handler Workshop to display the original source code.

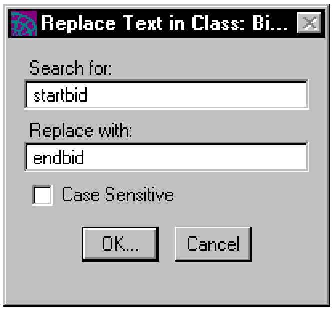
Replacing Text

The Replace Text… command finds all occurrences of the specified string in the method and event handler source code in the class.

➤ **To make a global replacement**

1. Choose the Edit > Replace Text… command.

   The Replace Text in Class dialog opens.

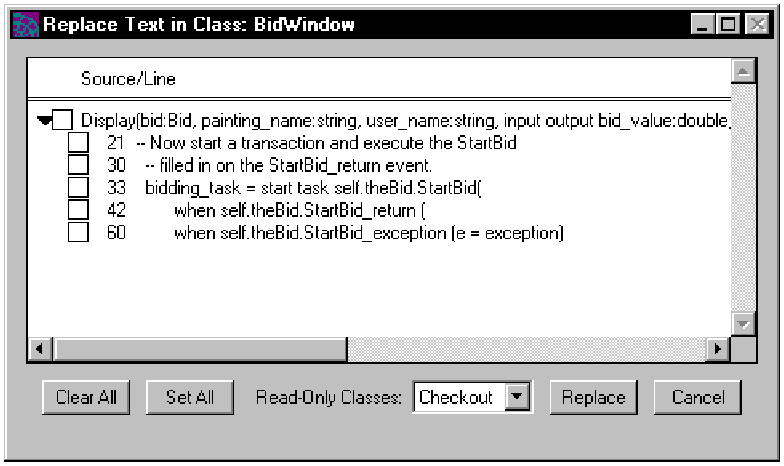


Search for: startbid
Replace with: endbid
Case Sensitive ☐

2. In the Replace Text in Class dialog, specify the string you wish to search for and the replacement string. Click the OK button to start the search.

   The default is a case-insensitive search. If you want a case-sensitive search, switch the Case Sensitive toggle to on.

3. When the search is complete, another Replace Text in Class dialog opens, displaying the method name or event handler name, and the line itself for each source code line that contains the string.



```
Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:double,
  21  -- Now start a transaction and execute the StartBid
  30   -- filled in on the StartBid_return event.
  33   bidding_task = start task self.theBid.StartBid(
  42        when self.theBid.StartBid_return (
  60        when self.theBid.StartBid_exception (e = exception)
```

4. Click the toggle next to the line where you wish to make the replacement.

   See below for further information about using the Replace Text in Class dialog.

5. In the Replace Text in Class dialog, click the Replace button to make the replacements.

Read-only classes By default, if a class is read only, the Replace Text… command automatically checks it out before making the replacement in any method or event handler source code. If it cannot be checked out, you will be prompted to indicate whether you wish to branch or ignore the read-only class. If you do not want a read-only class to be checked out, you can select one of the following options from the Read-Only Classes drop list on the Replace Text in Class dialog.

Replacement Option	Description
Ignore	Will not make the replacement in the class.
Branch	Will branch the class.
Checkout	The default. Will attempt to check out the class.

Selecting lines The Replace Text in Class dialog provides several features to make it easier to select the lines where you want to make the replacement. Use the Set All button to set the toggles for all the source code lines to on. Use the Clear All button to set the toggles for all the source code lines to off. If you set the toggle for a method or event handler name to on, this turns the toggles on for all relevant lines in the method or event handler.

Testing a Class

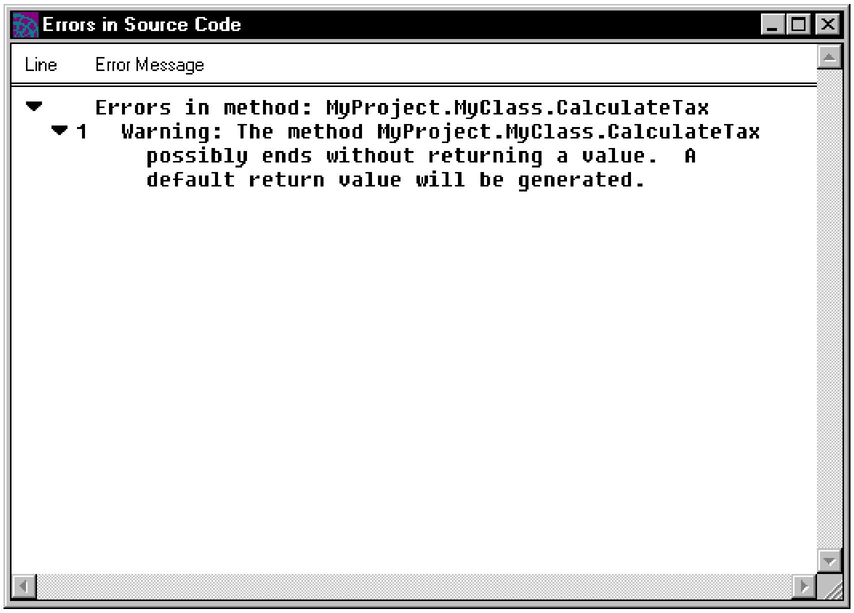
The File > Compile command (or the Compile button on the toolbar) compiles the class elements in your current class and reports the compilation errors. While it is not necessary to compile your class before you run the project, this command allows you to check for syntax errors without actually executing the code. Any compilation errors are reported in the Error window.

The Compile command compiles only those class elements that have changed since your last compilation. If you wish to compile all your class elements, regardless of whether or not they have changed, you can use the Utility > Force Compile command in the Repository Workshop. See "Compiling Plans" on page 191 for information about the Force Compile command.

Using the Error Window

The Error window displays the errors messages for the class in an outline field.

Figure 5-7 Error Window



```
▼     Errors in method: MyProject.MyClass.CalculateTax
  ▼ 1   Warning: The method MyProject.MyClass.CalculateTax
          possibly ends without returning a value.  A
          default return value will be generated.
```

You can jump directly from one of these messages to the code that caused the error. In the Error window, double-click the error you wish to find. The appropriate workshop will then open, with the cursor positioned at the beginning of the line that contains the code that caused the error.

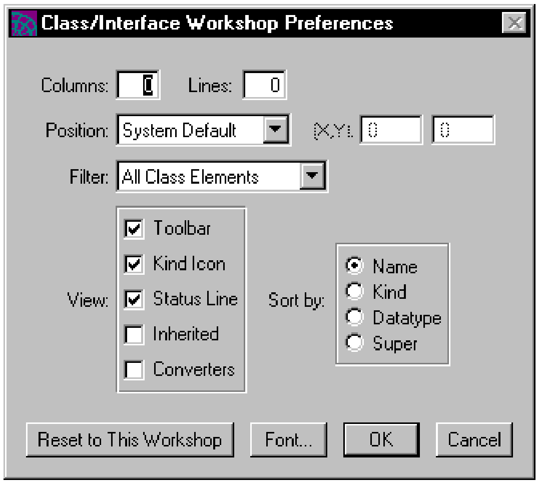
You can keep the Error window open as long as you need it. When you are finished using the window, click the Done button to close the window.

Setting Workshop Preferences

The Class Workshop allows you to set preferences that are saved as part of your current workspace.

To set the workshop preferences, choose the File > Workshop Preferences… command. This command opens the Class/Interface Workshop Preferences dialog, where you can set any number of preferences.

Figure 5-8 Class Workshop Preferences Dialog



The preferences you can set for the Class Workshop fall into the following general categories:

- workshop window size and position
- default filter
- viewing preferences
- sorting preference
- font preference

The workshop window size and position, filter, viewing, and font preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Class Workshop.

Sorting Preference

The sorting preference lets you specify the default sorting used for the Class Elements browser. The following table describes the sorting options:

Preference	**Description**
Name	Sorts the class elements by name.
Kind	Sorts the class elements by kind.
Datatype	Sorts the class elements by data type.
Superclass	When you are viewing inherited class elements, sorts the class elements by superclass.

Chapter 6

Using the Interface Workshop

This chapter provides conceptual information about interfaces and their elements, and describes how to use the Interface Workshop.

In this chapter, you will learn how to:

- examine an interface
- create an interface
- modify an interface
- set Interface Workshop preferences

About Interfaces

An interface defines a set of class elements, without providing the code that implements them. The interface provides the method and event handler signatures that define a standard "interface" to an object. The code for the methods and event handlers in the interface is provided by the classes that *implement* the interface.

For example, the AdaptableAuction sample application defines a TaxCalculationIFace interface, which provides a method signature and an event signature related to calculating taxes on a sale. The method is called CalculateTax and the event is called TaxCalculated. The code for CalculateTax is provided by the TaxCalculationImp class, which implements the TaxCalculationIFace interface.

For information about the benefits and uses of interfaces, see *iPlanet UDS Programming Guide*.

Interface elements An interface has the same elements as a class, except for attributes. These elements include: virtual attributes, methods (method signatures), events, event handlers (event handler signatures), and constants. A *method signature* is the method name, parameter list, and return value. Likewise, an *event handler signature* is the event handler name and parameter list. See "Interface Elements" on page 334 for complete information on these elements.

Implementing an Interface

Implementing an interface in a class means providing the code for all methods and event handlers defined in the interface. Any number of classes can implement a single interface, which provides multiple implementations for a single interface. In addition, a single class can implement multiple interfaces.

Using an Interface as a Data Type

You can use an interface as data type for any data item. The interface is the data item's declared type. The AdapatableAuction example uses the TaxCalculationIFace interface as the declared type of a local variable.

However, when you create the actual object associated with the data item, the object's runtime type must be one of the classes that implement the interface. In other words, the implementing class is the data item's runtime type. The AdaptableAuction example has a method that finds the library and class for the implementation, and creates an instance of that class. This instance is used to create the object. See *iPlanet UDS Programming Guide* for information and example of using interface as a type.

The classes that implement an interface can be included within your application code or you can load them at runtime. See *iPlanet UDS Programming Guide* for information about using interfaces for dynamic class loading or multiple inheritance, complete with examples.

Interface hierarchies Unlike a class, which always has a superclass, an interface does not need a super-interface. When you create an interface, you choose whether or not the interface has a super-interface. If the interface does have a super-interface, it inherits all the interface elements defined for its super-interface, just as a class inherits from its superclass. If the interface does not have a super-interface, you must define the entire interface from scratch—it does not inherit any interface elements.

Setting up an interface hierarchy is similar to setting up a class hierarchy. Each sub-interface inherits all the interface elements defined for its super-interface. The sub-interface can overload methods by defining different parameter types for the same method name. However, overriding is not allowed. Because methods and event handlers in interfaces have no code associated with them, there is no way to override them. See "Interface Elements" on page 334 for information about defining methods and event handlers in interfaces.

The following sections provide background information about creating an interface and providing implementations of the interface. "Interface Elements" on page 334 provides detailed information about the individual elements in an interface.

Creating an Interface

You define interfaces using the Interface Workshop. Creating an interface is similar to creating a class. You specify a name for the interface and, optionally, a super-interface. You then use the Interface Workshop to define each of the interface elements.

To create a new interface, you must start from the Project Workshop. The New Interface button or the Component > New > Interface command creates a new interface with the name you specify.

After the Interface Workshop opens, you can define the individual elements in the interface. See "Creating an Interface" on page 347 for information.

Implementing an Interface

To implement an interface, you declare one or more classes as "implementing" the interface. Then, in each class that implements the interface, you write the "implementation" code for all the methods and event handlers in the interface.

The implementing class must also define every event in the interface. You cannot exclude any of the methods, event handlers, or events, or you will get a compile error. Constants and virtual attributes do not need to be implemented in the implementing class.

When a class implements an interface, all of its subclasses also implement the interface. When you assign an object to the data item whose type is an interface, the object's type can be a class that implements the interface or a *subclass* of a class that implements the interface.

Remember, a single class can implement multiple interfaces. In addition, the class may also define functionality of its own. The multiple inheritance example described in the *iPlanet UDS Programming Guide* shows classes that implement the Sortable interface to complement their basic functionality.

To implement an interface in a class, you use the Class Workshop. The Class Properties dialog for an individual class allows you to list one or more interfaces which the given class will be implementing. Any number of interfaces can be implemented by a given class. And any number of classes can implement a given interface. See "Implementing Interfaces" on page 283 for information about implementing interfaces in the Class Workshop.

The classes that implement the interface can be included within your application code or you can load them at runtime. See *iPlanet UDS Programming Guide* for information about how to load the implementing classes at runtime.

Interface Elements

The following sections provide conceptual information about the elements common to all interfaces:

- virtual attributes
- methods
- events
- event handlers
- constants

Virtual Attributes

As described under "Virtual Attributes" on page 278, a virtual attribute does not store a value or point to an object—instead, a virtual attribute consists of two expressions, one that is evaluated when the program sets the value of the attribute and another that is evaluated when the program gets the value of the attribute. The Get expression for the virtual attribute is required, but the Set expression is optional. A virtual attribute without a Set expression is a read-only attribute.

Defining a virtual attribute in an interface allows you to make a method invocation look like an attribute. Typically, a virtual attribute provides a convenient way for getting and setting a complex value.

All methods used in the Get and Set expressions for the virtual attribute must be defined in the interface.

You do not need to "implement" the virtual attribute in the classes that implement the interface. When a data item's declared type is an interface, iPlanet UDS always uses the virtual attribute definition provided by the interface. However, you can "re-define" the virtual attribute in the implementing classes. In this case, when the *declared type* of the data item is an interface, iPlanet UDS uses the definition provided by the interface, and when the *declared type* of the data item is the implementing class, iPlanet UDS uses the definition provided by the class.

Methods

A method is a procedure that is specially written to operate on an object. Every method consists of a method signature (which specifies the method name, parameters, return type) and a statement block that contains the TOOL code that performs the operations on the object. In an interface, you define only the method signature; a method defined in an interface does *not* include source code. The class (or classes) that implements the interface provides the source code for the method.

The AdaptableAuction example provides two implementations of the TaxCalculationIFace interface. The interface defines the following method signature:

```
CalculateTax(theSale:Sale):double
```

The two implementations of this method signature are in separate projects, in classes with different names. One implementation of CalculateTax takes into consideration whether the buyer was a non-profit organization; the other doesn't.

For examples of the use of methods within interfaces, see the following methods in the AdaptableAuction example:

- CalculateTax (TaxCalculationImp class in AAImplementations project)
- CalculateTax (NewTaxCalculationImp class in AAImp2) project

Overloading You can overload a method in an interface. To overload a method, you simply add a new method signature using the same name with a different parameter list. The class (or classes) that implement the interface must provide source code for every method signature in the interface.

NOTE Because the interface defines only a method signature, overriding an inherited method has no effect. The inherited method and the overriding method have the exact same signature.

Events

An event is a signal that something has changed. Every event has a name and, optionally, one or more parameters. An event defined for an interface is exactly the same as an event defined for a class. You can use the `post` statement to post the event on any object whose class implements the interface.

All classes that implement the interface must implement all events defined in the interface. Implementing an event in a class consists of re-defining the event name and parameter list.

The TaxCalculationIFace interface in the AdaptableAuction example defines an event called TaxCalculated. Both implementations of this interface define the TaxCalculated event.

For further information on using events in TOOL, see the *TOOL Reference Guide*.

Event Handlers

An event handler is a named block of TOOL code that provides programming to be executed in response to one or more events. The event handler provides reusable, modular event handling code that you can include in any number of event statements.

In a class, an event handler consists of an event handler signature, which consists of the event handler name and parameters, and the event handler source code. In an interface, you define only the event handler signature; an event handler defined in any interface does *not* include source code. The class (or classes) that implements the interface provides the source code for the event handler.

No overloading Unlike methods, there is no overloading for event handlers. There can only be one event handler with a given name in the interface.

No overriding Because the interface defines only an event handler signature, overriding an inherited event handler has no effect. The inherited event handler and the overriding event handler have the exact same signature.

Chapter 11, "Using the Event Handler Workshop," contains detailed information about event handlers. For information on using event handlers in TOOL, see the *TOOL Reference Guide*.

Constants

A constant is a literal string or numeric value that has a name. When you declare the named constant, you specify a constant name and a value. You can then use the constant name in place of the value in the TOOL code that implements the interface's methods and event handlers.

The most common use for a constant within an interface is for specifying the values of a parameter.

When a constant is defined within an interface, the implementing code within classes that implement the interface can reference the constant directly. Other classes must reference the constant with the following syntax:

interface_name.constant_name.

Remember, although you can use constants to specify values in your TOOL code, you cannot use them to specify values in dialogs in the iPlanet UDS Workshops.

You do not need to "implement" the constant in the classes that implement the interface. When a data item's declared type is an interface, iPlanet UDS always uses the constant definition provided by the interface. However, you can "re-define" the constant in the implementing classes. In this case, when the *declared type* of the data item is an interface, iPlanet UDS uses the definition provided by the interface, and when the *declared type* of the data item is the implementing class, iPlanet UDS uses the definition provided by the class.

Extended Properties of Interface Elements

The extended properties feature allows you to assign arbitrary name-value pairs to the individual interface element. You can use the extended properties on a interface element for whatever purpose you choose. For example, you might wish to use them for comments.

Using the Interface Workshop

You enter the Interface Workshop from the Project Workshop either by opening an existing interface or by creating a new interface.

Opening an existing interface If you wish to examine or edit an existing interface, double-click the interface name, or click the interface name and choose the Component > Open command.

Creating a new interface If you wish to create a new interface, click the New Interface button or choose the Component > New > Interface command.

The Interface Workshop Window

The Interface Workshop window, shown in Figure 6-1, consists of three parts: the Interface Elements browser, the status line, and the toolbar, as shown in Figure 6-2.

Figure 6-1 Interface Workshop Window

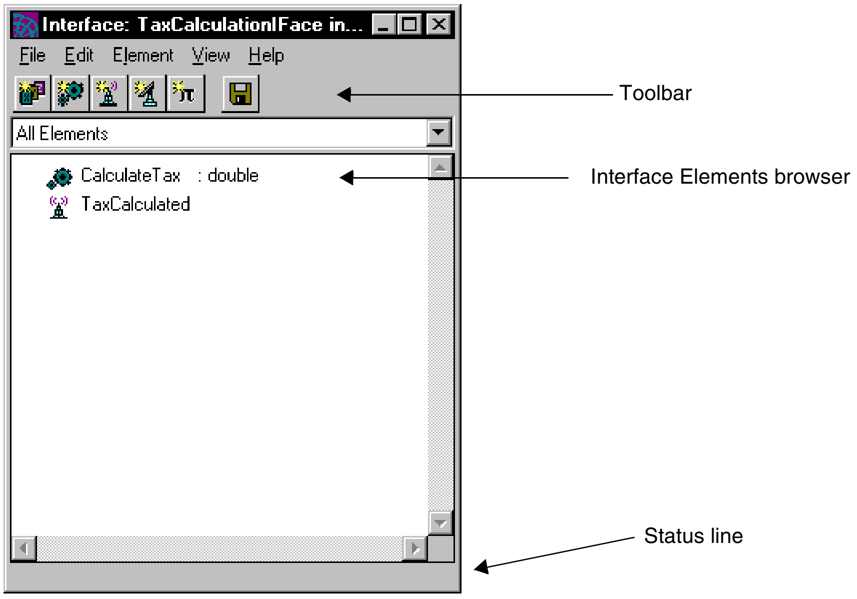
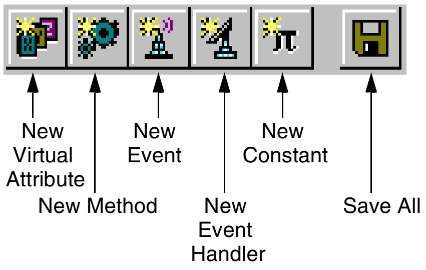


Figure 6-2 Interface Workshop Toolbar



View Menu

The View menu in the Interface Workshop provides the following toggles to control which parts of the workshop are displayed:

Command	Function
Toolbar	Makes the toolbar visible or invisible.
Kind Icon	When this toggle is turned on, the kind icons are displayed in the Interface Elements browser. When turned off, the kind icons are not displayed.
Status Line	Makes the status line visible or invisible.
Inherited	When this toggle is turned on, the Interface Elements browser displays the inherited interface elements. When turned off, the inherited interface elements are not displayed.

Note that you can set your viewing preferences for the workshop by using the Workshop Preferences... command. These preferences are saved as part of your current workspace. See "Setting Workshop Preferences" on page 360 for information on the Workshop Preferences... command.

Access to Other Workshops

From the Interface Workshop, you can access one other workshop:

Workshop	How to access it
Project Workshop	The File > Open Project… command opens the Project Workshop to display the definition of the project to which the current interface belongs. If the project is already being displayed, the Open Project… command moves the input focus to the appropriate Project Workshop window.

Leaving the Interface Workshop

To leave the Interface Workshop, use the File > Close command to close the workshop or use the system close box to close the window. This closes only the current workshop.

Examining an Interface

If the interface you wish to examine is not already displayed, you have two options. You can:

- use the Project Workshop to open an interface in the current project
- if the interface has a super-interface, use the Open Super-Interface command in the Interface Workshop to open one of the super-interfaces for the current interface

➤ **To examine an interface from the Project Workshop**

1. In the Project Components browser, double-click the interface name.

   You can also select the interface name, and choose the Component > Open command.

➤ **To use the Open Super-Interface command in the Interface Workshop**

1. In the Interface Workshop, choose the File > Open Super-Interface command.
2. On the Open Super-Interface submenu, select the super-interface you wish to open.

The Interface Workshop displays information about virtual attributes, methods, events, event handlers, and constants directly on the main workshop window.

The interface properties are not displayed on the main workshop window. To view them, you must use the File > Properties... command. The next sections provide detailed information about how to examine each of the interface elements and how to examine the interface properties.

Examining the Interface Elements

By default, the Interface Workshop displays all interface elements. A kind icon by each component name indicates the kind of element.

Icon	Interface Element
▣	Virtual attribute
⚙	Method
📡	Event
📡	Event handler
π	Constant

To turn off the kind icons, switch off the View > Kind Icon toggle.

Filter drop list To view a list of only one kind of interface element, use the filter drop list for the Interface Workshop. The filter drop list allows you to select a single interface element kind, such as “Methods” or “Virtual Attributes,” or “All Elements.”

Note that you can set your filter preferences for the workshop by using the Workshop Preferences... command. These preferences are saved as part of your current workspace. See “Setting Workshop Preferences” on page 360 for information.

Sorting By default the list of elements is in alphabetical order, sorting all interface elements by name. The View menu provides the following commands for sorting the interface elements:

Command	Description
Sort By Name	The default. Sorts the elements by name.
Sort by Kind	Sorts the elements by kind, virtual attribute, event, method, event handler, and constant.
Sort by Datatype	Sorts the elements by their data types.
Sort by Super-Interface	For inherited elements, sorts the elements by their super-interface.

By default the Interface Elements browser shows only the elements defined specifically for the selected interface. To view the elements inherited by the interface from its super-interfaces, choose the View > Inherited command.

Examining Methods

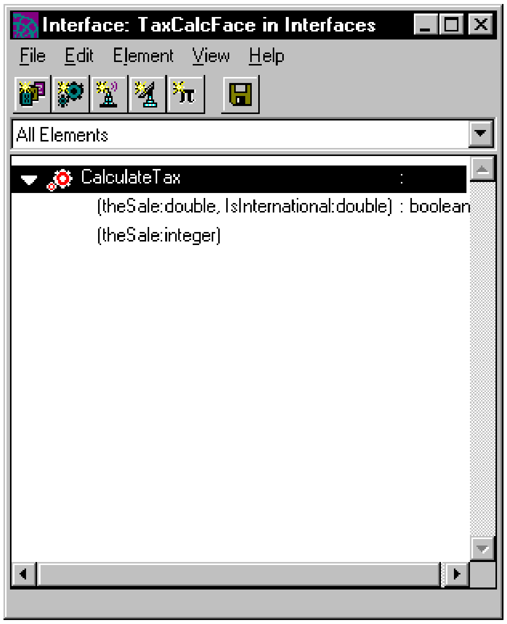
To display only the methods for the interface, choose "Methods" from the filter drop list in the Interface Elements browser. For methods, the browser displays the name and return type for each method.

To display the complete signature of the method, double-click on the method name, or select the method name and choose the Element > Open command. Opening the method opens the Method Properties dialog, which displays the complete method signature.

Overloaded methods If the method is overloaded, the Interface Workshop displays the method name as a folder, and opens the folder to display the individual method signatures for all the methods that share the same name. To display the complete definition of an individual method, double-click the signature for the particular method you wish to examine.

For example, Figure 6-3 shows the method signatures for all the CalculateTax methods:

Figure 6-3 Method Signatures for CalculateTax Method


Interface: TaxCalcFace in Interfaces
File Edit Element View Help
All Elements

CalculateTax :
(theSale:double, IsInternational:double) : boolean
(theSale:integer)

Examining Virtual Attributes

To display only the virtual attributes for the interface, choose "Virtual Attributes" from the filter drop list in the Interface Elements browser. For virtual attributes, the browser displays the name and data type for each attribute.

To display the complete definition of an individual virtual attribute, double-click on the attribute name, or select the attribute name and choose the Element > Open command. Opening the virtual attribute opens the Virtual Attribute Properties dialog, which displays the original definition of the attribute.

Examining Events

To display only the events for the interface, choose “Events” from the filter drop list in the Interface Elements browser. For events, the browser displays the name of each event.

To display the complete definition of an individual event, double-click on the event name, or select the event name and choose the Element > Open command. Opening the event opens the Event Properties dialog, which displays the original definition of the event.

Examining Event Handlers

To display only the event handlers for the interface, choose “Event Handlers” from the filter drop list in the Interface Elements browser. For event handlers, the browser displays the name for each event handler.

To display the complete signature of the event handler, double-click on the event handler name, or select the event handler name and choose the Element > Open command. Opening the event handler displays the Event Handler Properties dialog, which displays the complete event handler signature.

Examining Constants

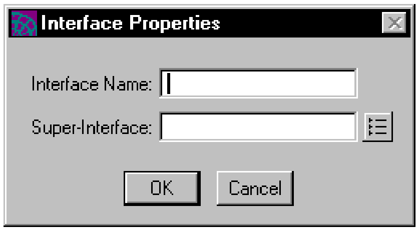
To display only the constants for the interface, choose “Constants” from the filter drop list in the Interface Elements browser. For constants, the browser displays the name and value for each constant.

To display the complete definition of an individual constant, double-click on the constant name, or select the constant name and choose the Element > Open command. Opening the constant opens the Constant Properties dialog, which displays the original definition of the constant.

Examining Interface Properties

To display the interface properties, choose the File > Properties… command. This command opens the Interface Properties dialog, which displays the interface name and super-interface, if there is one.

Figure 6-4 Interface Properties Dialog



Examining Extended Properties

To display the extended properties for an individual interface element, select the interface element and choose the Element > Extended Properties... command. The Extended Properties dialog opens, displaying the current settings for the extended properties.

Creating an Interface

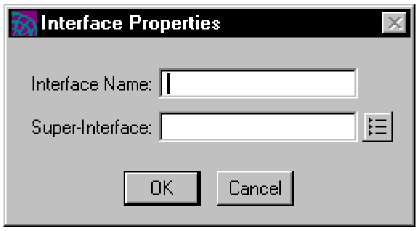
To create an interface, you must start from the Project Workshop.

The New Interface button or the Component > New > Interface command creates a new interface with the name you specify.

➤ **To create an interface**

1. In the Project Workshop, click the New Interface button or choose the Component > New > Interface command.

   The Interface Properties dialog opens.



2. In the Interface Properties dialog, specify the name and, if desired, the super-interface for the interface. Click the OK button to create the interface.

Fill in the fields on the dialog as follows:

Interface Property	**How to Specify It**
Interface Name	Type in the interface name.
Super-Interface	Type in the super-interface name or use the browser button to display a list of interfaces from which you can make a selection.

After the Interface Workshop opens, you can define the individual elements in the interface.

Defining Interface Elements

To define the elements in the interface, you can either create new elements or copy existing elements from another interface or from a class. You can copy these elements using the Edit > Copy, Cut, and Paste commands (described under "Using the Clipboard" on page 359), or you can copy them using drag and drop.

Using drag and drop You can copy an existing interface or class element by dragging it from the class or interface that defined it and dropping it on the current interface. If you drag a method or event handler from a class and drop it onto an interface, the code for the method or event handler is ignored. Only the method or event handler signature is added to the interface.

➤ **To drag and drop an interface element**

1. In the Workshop where the existing element was originally defined, select the element you wish to copy.
2. Drag the element to the Interface Elements browser for the new interface you are creating.
3. Drop the element onto the Interface Elements browser.

The following sections describe how to create new interface elements using the Interface Workshop. "Setting Extended Properties for Interface Elements" on page 356 describes how to set the extended properties for an interface element.

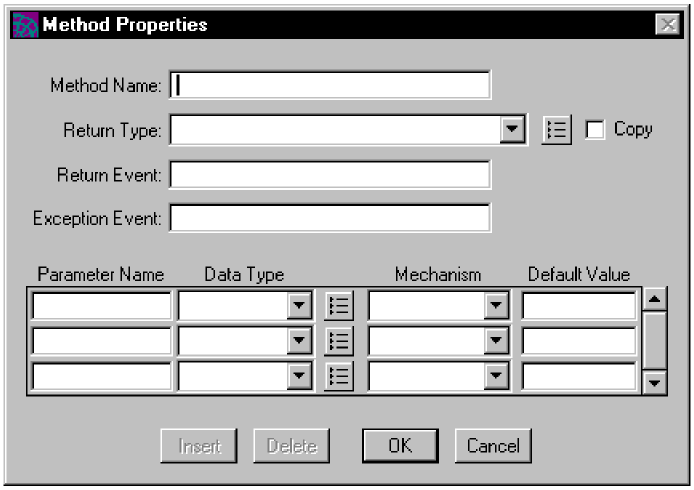
Defining Methods

To define a method, use the Element > New Method command, or click the New Method button on the toolbar.

➤ **To create a new method**

1. Choose the Element > New Method command, or click the New Method button.

   The Method Properties dialog opens.



2. In the Method Properties dialog, enter the method's name in the Method field. The return type, return event, exception event, and parameters are optional.

3. Click the OK button to add the method to the interface.

   When the Method Properties dialog closes, you will return to the Interface Workshop. Unlike when you create a method from the Class Workshop, when you create a method in the Interface Workshop, the Method Workshop does *not* open. You do not need to use the Method Workshop because there is no source code associated with a method in an interface.

See "Specifying Method Properties" on page 579 for information on the Method Properties dialog.

The following section provides information about overloading methods.

Overloading Methods

To overload a method, you simply create a new method with the same method as the method you wish to overload, but with a different parameter list.

When more than one method with the same name exists—with different parameter lists—iPlanet UDS automatically overloads the method.

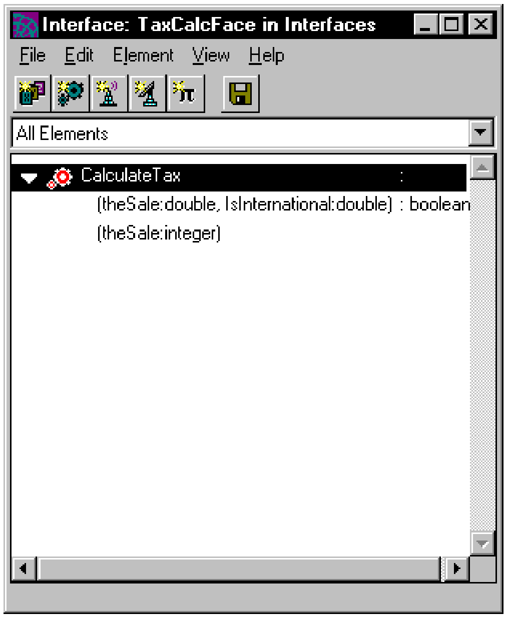
➤ **To overload an existing method**

1. Create a new method with the New Method button, or choose the Component > New Method command.

   The Method Properties dialog opens.

2. On the Method Properties dialog, enter the same method name, return type, and return events as the original method, but specify a different parameter list.

   The Interface Workshop displays all the method signatures for the overloaded method.



CalculateTax :
(theSale:double, IsInternational:double) : boolean
(theSale:integer)

Deleting a single method signature To delete a single method signature for an overloaded method, simply highlight the individual signature and choose the Edit > Delete command.

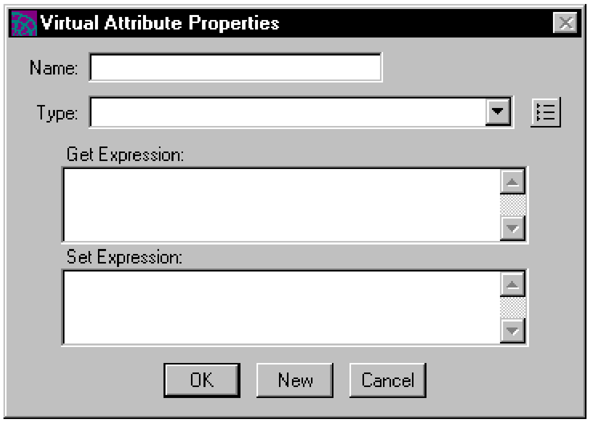
Defining Virtual Attributes

To define a virtual attribute, use the Element > New Virtual Attribute command, or click the New Virtual Attribute button on the toolbar.

➤ To create a virtual attribute

1. Choose the Element > New Virtual Attribute command, or click the New Virtual Attribute button.

   The Virtual Attribute Properties dialog opens.



2. In the Virtual Attribute Properties dialog, specify the name and type for the virtual attribute, as well as the Get and Set expressions.

3. Click the OK button to add the attribute to the interface and close the dialog, or click the New button to create the attribute and leave the dialog open so you can create another virtual attribute.

Fill in the fields on the dialog as follows:

Property	How to specify
Name	Type the attribute name.
Type	Choose the data type from the drop list, or use the browser button to select a class or interface name for the type.
Private	Turn on this toggle to make the attribute private.
Get	Enter a TOOL expression to be executed when the program accesses the value of the attribute (see "Defining Virtual Attributes" on page 311 for further information). The Get expression is required.
Set	Enter a TOOL expression to be evaluated when the program assigns a value to the attribute (see "Defining Virtual Attributes" on page 311 for further information). The Set expression is optional. A virtual attribute without a Set expression is a read only attribute.

Note that in order to write a virtual attribute definition for an interface, you must define the methods invoked by the virtual attribute *before* writing the virtual attribute definition.

For complete information about defining virtual attributes along with an example definition, see "Defining Virtual Attributes" on page 311.

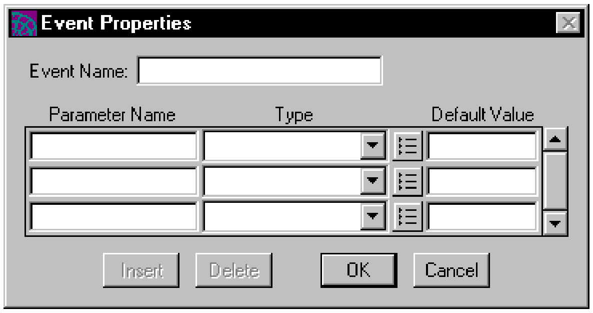
Defining Events

To define a new event, use the Element > New Event command, or click the New Event button on the toolbar.

➤ To create an event

1. Choose the Element > New Event command, or click the New Event Button.

   The Event Properties dialog opens.



2. On the Event Properties dialog, enter the event name.

   The event parameters are optional. See below for information on specifying the event parameters.

3. Click the OK button to add the event to the interface.

Event parameters The Event Properties dialog displays the event parameters in an array field. To add a parameter to the list, add a new row to the array field. Fill in the columns as follows:

Column	How to fill it in
Name	Type the parameter name.

Column	How to fill it in
Type	Choose a data type from the drop list or type in a class name.
Default Value	If desired, enter a value that is compatible with the parameter's data type. For parameters with a class data type, the default value must be NIL.

Delete and Insert buttons You can add as many parameters to the event as you wish. If you wish to add a parameter in the middle of the list, use the Insert button to insert a new row above the row you select. If you wish to delete a parameter from the list, use the Delete button to remove the currently selected row from the array field.

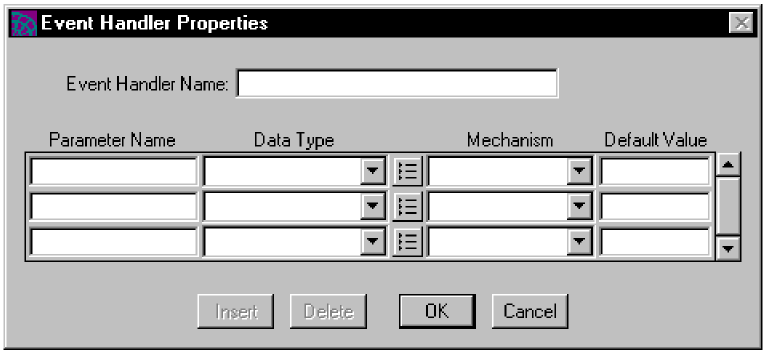
Defining Event Handlers

To define a handler, use the Element > New Event Handler command, or click the New Event Handler button on the toolbar.

➤ **To create a new event handler**

1. Choose the Element > New Event Handler command, or click the New Event Handler button.

   The Event Handler Properties dialog opens.



2. In the Event Handler Properties dialog, enter the event handler's name in the Handler field. The parameters are optional.

3. Click the OK button to add the handler to the interface.

When the Event Handler Properties dialog closes, you will return to the Interface Workshop. Unlike when you create an event handler from the Class Workshop, when you create an event handler in the Interface Workshop, the Method Workshop does *not* open. You do not need to use the Event Handler Workshop because there is no source code associated with an event handler in an interface.

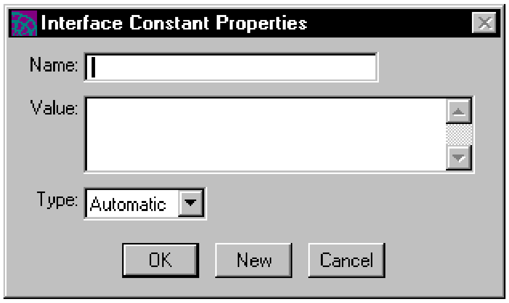
See “Specifying Event Handler Properties” on page 603 for information on the Event Handler Properties dialog.

Defining Interface Constants

To define a constant, use the Element > New Constant command, or click the New Constant button on the toolbar.

➤ To create a constant

1. Choose the Element > New Constant command.
2. The Constant Properties dialog opens.



3. In the Constant Properties dialog, specify the name, type, and value for the constant. The constant types are described below.
4. Click the OK button to add the constant to the interface or the New button to create another constant.

Constant types The constant types are:

Constant Type	Description
Automatic	iPlanet UDS determines the type based on the constant value (see below). Use the Automatic type for strings.
Boolean	Allows a value of TRUE or FALSE.
Double	Allows a floating point number.
Integer	Allows a positive or negative whole number.
String	Allows an alphanumeric string.

Automatic type For an automatic type, the value you specify determines the type of the constant. The types are:

Value	Type
TRUE, FALSE	boolean
Positive or negative whole number	integer
Floating point number	float
Character data (without quotes)	string

Single quotes for strings Normally, you do not need to enclose a string value in single quotation marks. However, if you wish to create a string constant with a value of an integer or a floating point, you must use single quotes. In addition, because iPlanet UDS automatically truncates trailing spaces, you need to use single quotes to specify a string value with trailing spaces.

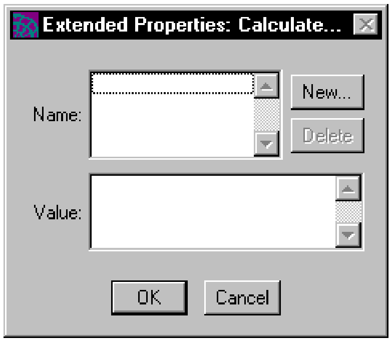
Setting Extended Properties for Interface Elements

To set the extended properties on an individual interface element, use the Element > Extended Properties... command.

➤ **To set extended properties for a interface element**

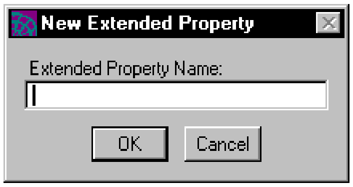
1. Select the element for which you wish to set extended properties.
2. Choose the Element > Extended Properties... command.

   The Extended Properties dialog opens.



3. Click the New... button.

   The New Extended Property dialog opens.



4. Enter the name of the property you wish to create and click OK.
5. In the Value field, enter the value of the extended property.
6. To enter additional extended properties, repeat steps 3 through 5.
7. Click OK.

Modifying an Interface

Before modifying an interface, you must have write access to it. You have write access to an interface if you have just created it (and have not yet integrated your workspace), or if you have checked out or branched the interface. In the Project Workshop, you can use the View > Writeable Icon command to see if you have write access to an interface. If you do not have write access to the interface, you must either check it out or branch it before you can modify it.

In the Interface Workshop, you can modify an interface by modifying the definition of an element, by deleting an element, or by changing the interface properties. The Edit menu in the Interface Workshop provides commands that let you use the clipboard to copy or move interface elements from one interface (or class) to another.

Updating Interface Elements

To update an interface element, double-click the element name in the Interface Elements browser, or select the element name and choose the Element > Open… command. Opening the interface element opens the dialog in the Interface Workshop where the item was originally defined.

Deleting Interface Elements

The Edit > Delete command lets you delete interface elements from the Interface Elements browser.

➤ **To delete an interface element**

1. Select the element you wish to delete by clicking on the name.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the interface element.

Updating Interface Properties

To modify the interface properties, choose the File > Properties… command. This command opens the Interface Properties dialog, where you can make your updates.

Updating Extended Properties for Interface Elements

To modify the extended properties for an individual interface element, select the interface element and choose the Element > Extended Properties... command. This command opens the Extended properties dialog, where you can add, delete, or change the extended properties.

Using the Clipboard

The Edit menu in the Interface Workshop allows you to cut or copy an interface element onto the clipboard the same way you cut or copy text in a text editor. Once the interface element is in the clipboard, you can paste it into another interface or into a class.

The commands on the Edit menu for using the clipboard are:

Command	Description
Cut	Removes the selected element from the current interface and copies to the clipboard.
Copy	Copies the selected element onto the clipboard.
Paste	Pastes the class or interface element in the clipboard into the current interface.

Copying class elements You can also copy an element from class to an interface using the Copy and Paste commands or using drag and drop. When you copy a method or event handler from a class and add it to an interface, the code associated with the method or event handler is ignored. Only the method or event handler signature is copied.

Setting Workshop Preferences

The Interface Workshop shares workshop preferences with the Class Workshop. Preferences that you set using the Class Workshop take effect for the Interface Workshop; likewise, preferences you set using the Interface Workshop take effect for the Class Workshop.

To set the workshop preferences in the Interface Workshop, give the File > Workshop Preferences… command. This command opens the Class/Interface Workshop Preferences dialog, where you can set any number of preferences. See "Setting Workshop Preferences" on page 329 for information about using the Workshop Preferences... command for the Class and Interface Workshops.

Chapter 7

Using the Window Workshop

This chapter provides conceptual information about windows and their components, and describes how to use the Window Workshop.

In this chapter, you will learn how to do the following:

- enter and leave the Window Workshop
- use the toolbar, widget palette, color palette, and status line
- create a window and set its properties
- create simple and compound widgets
- select and group widgets
- open a widget properties dialog
- work with inherited windows
- test a window
- import and export a window
- set Window Workshop preferences

Chapter 8, “Working with Widgets,” provides detailed information on creating and editing the individual widgets that you use in a window.

Chapter 9, “Using the Menu Workshop,” provides detailed information on building the menu bar for a window.

About Windows

This section provides background information about windows. The following topics are covered:

- "About Window Components" on page 362

  This section provides information about the components that make up a window: the window frame, the form, the menu bar, and the iPlanet UDS widgets.

- "About Window Style and Other Properties" on page 365

  This section provides information about the basic window styles and other properties that affect window appearance and behavior.

- "About Creating New Windows" on page 372

  This section provides information about the kinds of windows you can create using the Window Workshop.

- "About Internationalizing Windows" on page 382

  This section describes how you can create multilingual windows using the Window Workshop.

- "About Help for Windows" on page 383

  This section briefly describes the Help features available for windows.

About Window Components

A window consists of three basic components:

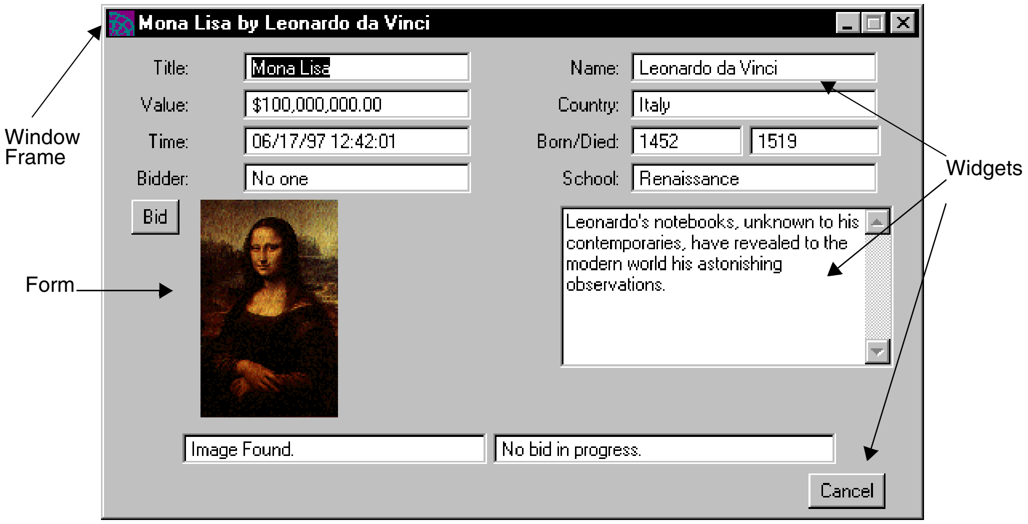
Window Frame A window's border, containing controls—menus and buttons—for manipulating the window, such as closing or minimizing the window. The window style determines whether or not the window has a frame.

Form The display area within the window frame where the user can interact with the application by examining and entering data, and by making selections and giving commands. All windows have forms.

Menu Bar A window's menu bar, which contains pull-down menus for giving application commands or changing application settings. Menu bars are optional.

Figure 7-1 illustrates a window's components.

Figure 7-1 Components of a Window



Mona Lisa by Leonardo da Vinci

Title:	Mona Lisa	Name:	Leonardo da Vinci
Value:	$100,000,000.00	Country:	Italy
Time:	06/17/97 12:42:01	Born/Died:	1452 1519
Bidder:	No one	School:	Renaissance

Leonardo's notebooks, unknown to his contemporaries, have revealed to the modern world his astonishing observations.

Image Found.

No bid in progress.

The windows that you build in the Window Workshop are completely portable. To provide a native look and feel, iPlanet UDS uses the current window system's window format. In this way, the iPlanet UDS window takes on both the familiar appearance and the window controls normally provided by the window system.

Note that if you are creating a user interface that will run on multiple window systems, there are some design issues you need to be take into consideration when planning your windows. See the *iPlanet UDS Programming Guide* for information.

About Forms

The form is a display area within the window frame where the user can interact with the application by examining and entering data, and by making selections and giving commands.

To create a window's form, you use widgets. Widgets are controls, such as buttons, list fields, picture fields, and array fields. These let the user select choices, give commands, enter data, and read information. When you first create a window, the form is completely blank. To format the form, you can place any number of widgets on it and arrange them as you wish. The widget palette in the Window Workshop provides a selection of widgets, which you can position directly on the form. The following section provides general information about working with widgets.

Form layers A window's form consists of two layers: the graphic layer and the field layer. The graphic layer contains all the form's graphic fields. The field layer contains all the form's field widgets. The graphic layer is always below the field layer. If you place a field on top of a graphic, it will cover the graphic. If you place a graphic on top of a field, it will move below the field. The following section provides background information on field and graphic widgets.

About Widgets

A widget is a graphical representation of a command or choice the user can select using the conventions of the host window system—either by clicking with a mouse or using the keyboard.

The widgets for formatting forms are divided into two groups: simple widgets and compound widgets.

Simple widgets Simple widgets are widgets that generally serve a single function, such as push buttons, text fields, and rectangles. Simple widgets are described in more detail below.

Compound widgets Compound widgets contain simple widgets, either to organize them or provide more complex functions for widgets as a group. For example, a grid field aligns a group of widgets in rows and columns, while a panel lets you provide preliminary data validation for a group of widgets.

There are also menu widgets, both simple and compound, for building menus for windows. For details on menu widgets and building menus, see Chapter 9, "Using the Menu Workshop."

About Simple Widgets

Simple widgets are divided into two groups: field widgets and graphic widgets.

Field widgets Field widgets are for displaying data, and accepting data and commands from the end user. For example, a radio list displays a list of choices, from which the end user can make one selection. With the exception of push buttons and picture buttons, all field widgets have data associated with them. For example, the value of a radio list is whatever value from the list is currently selected. End users set the values of these fields by selecting a value or entering data.

Graphic widgets Graphic widgets serve only as form decoration and visual organization cues. They do not accept commands or display or represent data. You use most graphic widgets, such as lines and rectangles, to organize a form visually. You use some graphic widgets, such as picture graphics, to decorate a form.

About Window Style and Other Properties

A window has a number of properties that affect the appearance of the window as well as its behavior. The following sections provide background information on these properties.

Window Style

The style of a window determines basic display characteristics for a window—whether the window is movable, resizeable, or has a title bar.

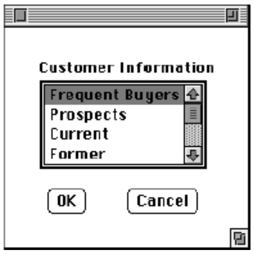
The following table briefly describes the values allowed for the Window Style property. These styles are described in further detail below.

Window Style	Definition
Resizeable	A resizeable, movable window with a title bar. Generally, a resizeable window also has a close box and a resize box.
Non-Resizeable	A fixed-sized, movable window with a title bar.
Frameless	A fixed-sized, immobile window without a title bar.

iPlanet UDS window styles correspond to the window styles provided by the host window systems. The appearance of a window in any iPlanet UDS window style is dependent on the host window system. For example, the resize box and its function vary from one window system to another.

Resizeable Window Style From an end user's perspective, a resizeable window is a fully functional window. A user may move the window, resize it, or close it without restriction. Figure 7-2 shows an example of a window in the resizeable style.

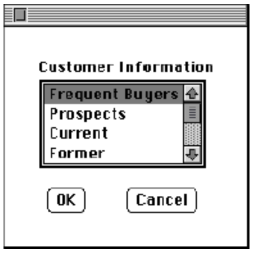
Figure 7-2 Resizeable Window



You normally use resizeable windows to represent an application at its base level. It is in an application's base window—a resizeable window—that end users conduct most of their work, branching to other windows in other window styles to perform specific application-dependent tasks.

Non-Resizeable Window Style A non-resizeable window behaves like a resizeable window in all respects but one: it is not resizeable. Figure 7-3 shows an example of a window in the non-resizeable style.

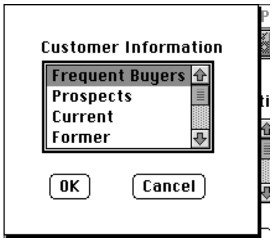
Figure 7-3 Non-Resizeable Window



You might use a non-resizeable window to enable a user to perform a specific task, such as spell-checking, while providing free access to the application's other windows.

Frameless Window Style A frameless window is neither resizeable nor moveable, and has no title bar. Figure 7-4 shows an example of a window in the frameless style.

Figure 7-4 Frameless WIndow



You use frameless windows to represent functions you want the user to complete without access to other application windows—functions whose timing is critical to an application's runtime operation. For example, you might use a frameless window for specifying environment settings.

NOTE To create a standard *dialog window*, complete with appropriate buttons, you can use one of the dialog methods defined by the Window class. See the Display Library online Help for details on these methods.

Initial Position Property

By setting a window's Initial Position Policy property, you determine the initial placement of the window when the window first runs.

You set a window's initial position either in relation to the screen or an application's primary window, or absolutely.

Relative initial window position To set a window's position relative to either the screen or to the primary window, you use the Initial Position Policy in conjunction with the Initial X and Initial Y specifications. The Initial X setting determines (in *mils*—thousandths of an inch) how far to horizontally offset the window relative to the screen or a primary window. The primary window is normally the window that opened the current window.

Absolute window position To set a window's position absolutely, you choose an initial position policy of Screen Centered or Primary Centered. These absolute policy settings ignore the Initial X and Initial Y settings, if any.

The following table summarizes the window position settings.

Initial Position Setting	Definition
System Default	The default position used by the particular window system on which the window is being displayed.
Screen Centered	Centers the window relative to the screen.
Primary Centered	Centers the window relative to the window specified as the primary window.
Screen Relative	Positions the window relative to the screen, using the Initial X and Initial Y setting to offset the window from the screen.

Initial Position Setting	Definition
Primary Relative	Positions the window relative to an application's primary window, using the Initial X and Initial Y settings to offset the window from the primary window.

Iconize Enabled Property

The Iconize Enabled property allows you to control whether a main window has an iconize symbol or iconize menu item in the window system menu.

When Iconize Enabled property is set to on (the default setting), the window has a close box or a close item in the window system menu.

This attribute has no effect on nested windows.

Initial Display State

The Initial Display State property allows you to specify the display state of a main window when it is first opened: whether it is maximized, hidden, iconized, or standard.

The following are the values for the Initial Display State property:

Value	Definition	Notes
Alternate	The window is maximized.	The window is resized to fit the entire screen. This is the same as the user clicking the Zoom box or Maximize button.
Hidden	The window runs, but is invisible.	Window can still receive and process events.
Iconized	The window is collapsed into an icon.	Window can still receive and process events. Window's IsIconizeEnabled attribute must be set to TRUE.
Normal	The window appears on the screen at its normal size.	

The window ignores Initial Display State settings that may conflict with Window Style settings.

This property has no effect on nested windows.

Maximize Enabled

The Maximize Enabled property lets you control whether or not a main window has an alternate (or maximize) symbol or a maximize menu item in the window system menu.

This property has no effect on nested windows.

Autosize Enabled

The Autosize Enabled property specifies whether or not a window is automatically resized when the widgets that the window contains are resized while the window is running. Initially, when the window is first displayed, iPlanet UDS *always* sizes the window to display all the widgets. After initially displaying the window, iPlanet UDS uses the Autosize Enabled property to determine whether or not to resize the window the next time the end user or the program resizes the widgets within the window. This ensures that the window is always large enough to display the entire contents of the form.

Window style and auto-sizing If the Window Style property is set to Non-Resizeable or Frameless, it is a good idea to set the Autosize Enabled property to on. Because the end user cannot resize the window, setting it to "autosize" ensures that the window's contents will always be displayed

If the Window Style property is set to Resizeable, the end user can resize the window himself. Therefore, it is not necessary for you to set Autosize Enabled to on.

Stay On Top Property

The Stay On Top property allows you to specify that the current window remain on top of the other windows on the screen, even when the end user activates another window. This feature is useful for toolbars and status windows that should always be displayed. The Stay On Top toggle is turned off by default.

NOTE The Stay On Top property is available only for Windows platforms. On other platforms, the property is ignored.

Tool Window Property

The Tool Window property allows you to specify that the current window not be displayed in the window system's task bar when the window is iconized. This feature allows you to create a secondary window for the application that is not displayed in the task bar when the main window for the application is iconized. The Tool Window toggle is turned off by default.

We strongly recommend that when you turn on the Tool Window toggle, you also turn off the Iconized Enabled property. Otherwise, once the end user iconizes the window, there will be no way for him to get it back.

Note that the "Tool Window" part of this property's name refers to the Windows 95 concept of a "tool window" and does not have any relationship to the iPlanet UDS TOOL language.

NOTE	This feature is available on Windows 95/NT only. On other platforms, the attribute is ignored.

System Close Policy

The System Close Policy property determines how the window responds to user action to close the window at the window system level. The settings are:

System Close Policy	Description
Disabled	Prevents end users from closing the window through the window system icon.
Post Shutdown	Invokes the PostShutdown method on the window and any child windows when the user closes the window.
Post Finalize	Invokes the RequestFinalize method on the window when the user attempts to close the window.
Post Shutdown Task Only	When the end user closes the window, iPlanet UDS invokes the PostShutdown method on the window being closed, but not on any of its child windows. The child windows continue to run.

This property has no effect on nested windows.

Usage Property

A window's *usage* provides a convenient way to control the states of all the widgets on the window. By defining window usages, you can reset all the widget states at once by simply switching to another window usage.

For each window usage, you can set every widget in the window to a particular state. The widget's *state* allows you to control how the end user interacts with the widget. For example, for an application that lets end users enter data into a form, you would want most widget states to be Update. However, for a read–only window, you would want the widget states to be View. For more information about widget states, see "About Widget States" on page 435 and the Display Library online Help.

Default window usage When you create a window in the Window Workshop, iPlanet UDS automatically sets the window usage to Update. The Update window usage sets the states for all the widgets in the window to let the end user "update" the form (that is, to set values or enter data). This default usage is appropriate for most applications.

However, if you are creating an application that runs in different modes or that lets the end user draw on the form or edit it (rather than letting the end user update it), you may want to change the window usage.

Setting the Usage property in the Window Workshop determines which usage is in effect when the window first opens. For example, imagine you want to provide a read-only form, where the user can view information but not update it. In the Window Workshop, you can change the window usage from Update to View.

The following table describes the six window usages:

Under This Usage	A Window Behaves This Way
View	The default states for widgets refuse keyboard and mouse input. In this usage, a user could only view a text field and would be unable to select the text field itself or its text. The default widget state for this usage is View Only.
Update	The default states for widgets accept mouse actions and keyboard input; the widgets themselves are changeable. In this usage, for example, a user could edit the text in a text field, move the text field itself, and change the font of the text field. The default widget state for this usage is Update.
Query	The default states for widgets are the same as Update.
User1	A user-defined usage. The default states for widgets are the same as Update.
User2	A user-defined usage. The default states for widgets are the same as Update.
User3	A user-defined usage. The default states for widgets are the same as Update.

iPlanet UDS provides default settings for all six usages, as shown in the table above. The View, Update, and Query usages are all prefabricated usages, defined for your convenience to set widgets to the corresponding states. You can override these settings to define these usages in any way you wish.

Defining a usage The User1, User2, and User3 usages are meant for you to define for your own use. They initially set all widgets the same as the Update usage.

Visual Style Property

For Motif and Windows, you can set the visual style for individual fields on the window. The visual style of a field is either three dimensional or two dimensional. For Windows 95 and Windows NT, you should always use the three-dimensional style (the default).

By default, the visual style for the *window* is determined by the current setting of the FORTE_VISUAL_STYLE environment variable. The default value of the FORTE_VISUAL_STYLE environment variable is “3D”.

About Creating New Windows

You use windows to create a graphical user interface for an iPlanet UDS application or as page templates for printing. When you are creating a window in the Window Workshop, you need to consider how the window is going to be used by your application. The way the window is used by the application influences how you design the window, for example, whether you include a menu bar on the window, whether you create a generic window that can be reused, and so on.

The window you create in the Window Workshop can be any of the following:

Standard window A standard window is a new, independent window that you create for your user interface. To create a standard window, simply create a window class that is a subclass of the iPlanet UDS UserWindow class. This window class provides a empty window for which you create the form and menu bar from scratch.

Inherited window An inherited window is a window that inherits part of its appearance and behavior from an existing window. If you want your new window to inherit from an existing window, you can create a new window class that is a subclass of another custom window class (rather than the UserWindow class). When you define a subclass of a custom window class, your new window class inherits the form and menu bar from its superclass in addition to its methods, attributes, event handlers, and events. You can then extend the inherited form and menu bar by adding new widgets to the subclass window.

Nested window A nested window is a window that is designed to be displayed as part of one or more other windows. When you nest one window in another window, iPlanet UDS displays the nested window inside the main window so that it appears to be part of the main window. Creating a nested window is a good way to reuse a standard form or "subwindow" that is needed by several other windows.

Page templates A page template is a window that is designed specifically for printing. In the Window Workshop, creating a page template is the same as creating a window for a user interface—the fact that you are going to print the window instead of display it affects only the way you design the windows. The difference between printing a window and displaying a window is the way you handle the window in your code; you use the iPlanet UDS printing classes to print the window rather than opening it for display.

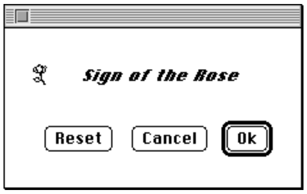
You can also set up an iPlanet UDS window so that it can be converted into a Web page.

This following sections provide information about inherited windows, nested windows, page templates, and windows as Web pages.

About Inherited Windows

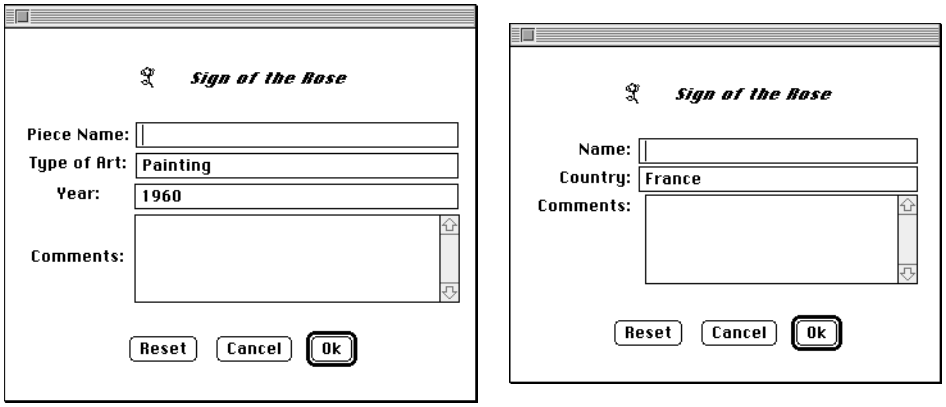
When you build a new window, you can create a *subclass window* that inherits its initial appearance and behavior from another window. The ability to create a subclass window enables you to define one window superclass that provides application or company-wide window formatting. You can then create subclasses of the generic window as many times as you wish, and customize each subclass window as appropriate for the application. The following example shows a simple data entry window skeleton with a logo and buttons.

Figure 7-5 Example Window Superclass



The following example shows two subclass windows that have inherited the superclass window's widgets, and that also have additional data fields:

Figure 7-6 Example Subclass Windows



Sign of the Rose

Piece Name:
Type of Art: Painting
Year: 1960
Comments:

Sign of the Rose

Name:
Country: France
Comments:

To see these windows in context, see the iPlanet UDS example program InheritedWindow.

To create an inherited window, you declare your window class as a subclass of any existing window class. The new subclass window automatically inherits the form and menu bar defined for the existing window class (as well as the methods, attributes, events, and event handlers in the superclass). When you open the Window Workshop, the inherited window will be displayed. You can use the Window Workshop to add widgets to the inherited window or even to modify the inherited widgets.

iPlanet UDS also allows you to modify the superclass window at any time, even after creating the subclass windows. Any changes you make to the superclass are automatically inherited by its subclasses—you will see the inherited changes the next time you open the subclass window. If any of the subclasses have modified the inherited widgets, iPlanet UDS automatically merges the superclass changes with the subclass changes. Of course, you need to carefully coordinate changes made to the superclass window to ensure that the subclass windows still work properly. See "Working with a Subclass Window" on page 424 for information about creating and modifying inherited windows.

Named event handlers and inherited windows iPlanet UDS named event handlers provide a very convenient way for you to provide event handling code in the superclass window that is inherited by the subclass windows. This works very effectively to allow both the widget and the widget's behavior to be defined in the superclass and then inherited by the subclasses. See Chapter 11, "Using the Event Handler Workshop," for information about named event handlers and the Event Handler Workshop. See the *iPlanet UDS Programming Guide* for an example of an event handler used for an inherited window.

About Nested Windows

A nested window is a window that is designed to be displayed inside another window. When the application is running, the nested window appears to be a compound field on the main window. You can nest the same window on any number of other windows throughout your application (or in multiple applications). Nesting windows provides a way to reuse a standard "subwindow" (and the event handling code associated with it) in more than one window.

Figure 7-7 shows the original window as it was designed in the Window Workshop.

Figure 7-7 Example Subwindow

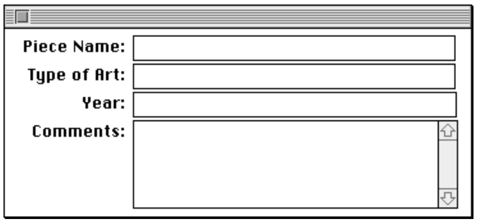
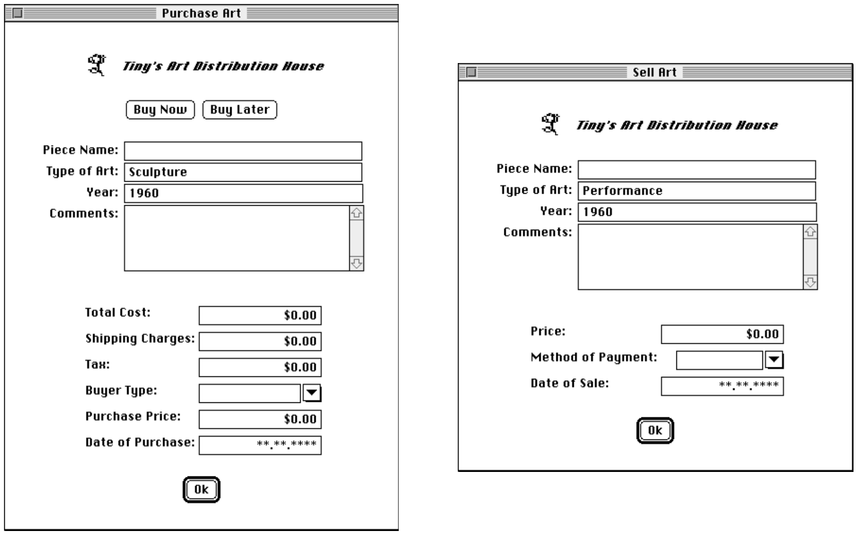


Figure 7-8 shows how the nested window looks when the application is running and the nested window is displayed on top of two different main windows.

Figure 7-8 Main Windows Displaying Nested Windows



Purchase Art

Tiny's Art Distribution House

Piece Name:
Type of Art: Sculpture
Year: 1960
Comments:

Total Cost: $0.00
Shipping Charges: $0.00
Tax: $0.00
Buyer Type:
Purchase Price: $0.00
Date of Purchase: **.**.****

Sell Art

Tiny's Art Distribution House

Piece Name:
Type of Art: Performance
Year: 1960
Comments:

Price: $0.00
Method of Payment:
Date of Sale: **.**.****

To see these windows in context, see the iPlanet UDS example program NestedWindow.

You can make any window into a nested window. The only restriction is that the nested window cannot have a menu bar. Typically, you display the nested window within a compound field in the outer window, for example, within an empty cell of a grid field. When you are designing the nested window, you should consider exactly where it will be displayed on the outer window. You should take this into account when planning the size and content of the outer window.

Set Parent attribute of nested window After creating the nested window in the Window Workshop, you can display it on any number of other windows at runtime. To nest a window in another window, you set the value of the nested window's Parent attribute to the compound field in the outer window where you wish to display the nested window. For example:

Code Example 7-1 Nesting a Window Example

```
-- This fragment shows how to nest a window.
artobjectWindow : ArtObjectWindow = new;

-- Nest the Art Object Window. First assign a row
-- and column position. Give it a compound field as
-- a parent. Remove its frame so it will look like a
-- seamless part of the parent window.

artObjectWindow.Window.Row = 3;
artObjectWindow.Window.Column = 1;
artObjectWindow.Window.Parent = <main_grid>;
artObjectWindow.Window.FrameWeight = W_NONE;

self.Open();
...
```

Project: NestedWindow • **Class:** PurchaseWindow • **Method:** Display

Whenever you specify a parent for a window, iPlanet UDS displays the nested window without its frame (the title bar, close box, and so on) so that it does not look like a separate window. For information on how to write the Display method for a nested window, see *iPlanet UDS Programming Guide*.

Named event handlers and nested windows iPlanet UDS named event handlers provide a very convenient way for you to define event handling code for the nested window that can be used by the window that displays the nested window. When the nested window definition includes named event handlers, the outer window can easily include the event handlers defined for the nested window as part of its own event handing code, making the two windows work together as one seamless window. See Chapter 11, "Using the Event Handler Workshop," for information about named event handlers and the Event Handler Workshop.

Note that a dialog box is not the same thing as a nested window. A dialog box is simply a standard window without a menu bar. The way your program opens the dialog box, the format of the dialog box (usually containing OK and Cancel buttons), and the behavior of the dialog box (it usually blocks the window that opens it) is what differentiates a dialog from other windows in your user interface.

Inherited Windows or Nested Windows?

Obviously, inherited windows duplicate some of the functionality of nested windows, that is, the ability to reuse window formatting and event handling code for more than one window. When should you use inherited windows? Inherited windows are most useful when you want to modify the form and menu inherited from the superclass. When should you use nested windows? Nested windows are most useful when you have a self-contained, modular form that you wish to reuse without modifications.

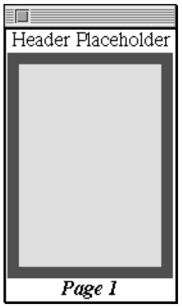
One advantage of inherited windows is the ability to view and customize the subclass windows in the Window Workshop. You cannot display a nested window on top of another window in the Window Workshop. Nested windows are displayed on their parent windows only when the application is running. To customize a single instance of a nested window, you must do so dynamically from your TOOL code.

About Windows as Page Templates

You can design a window to be used as a standard page in a report. The window functions as a form, which will be filled in with data at runtime. In your application, you can write TOOL code to fill in the data and print the report page. You can also prompt the end user to select printing options that control how the page is printed.

Figure 7-10 illustrates a window in the Window Workshop that is intended for printing a report with header and footer information.

Figure 7-9 Page Template Window



To see this window in context, see the ReportTemplateWindow class in the iPlanet UDS example program PrintSample.

A page template window can be any window class, either a standard subclass of UserWindow or any inherited window. There is nothing special you need to do when you create the page template window in the Window Workshop, except consider how the window layout will look when it is printed.

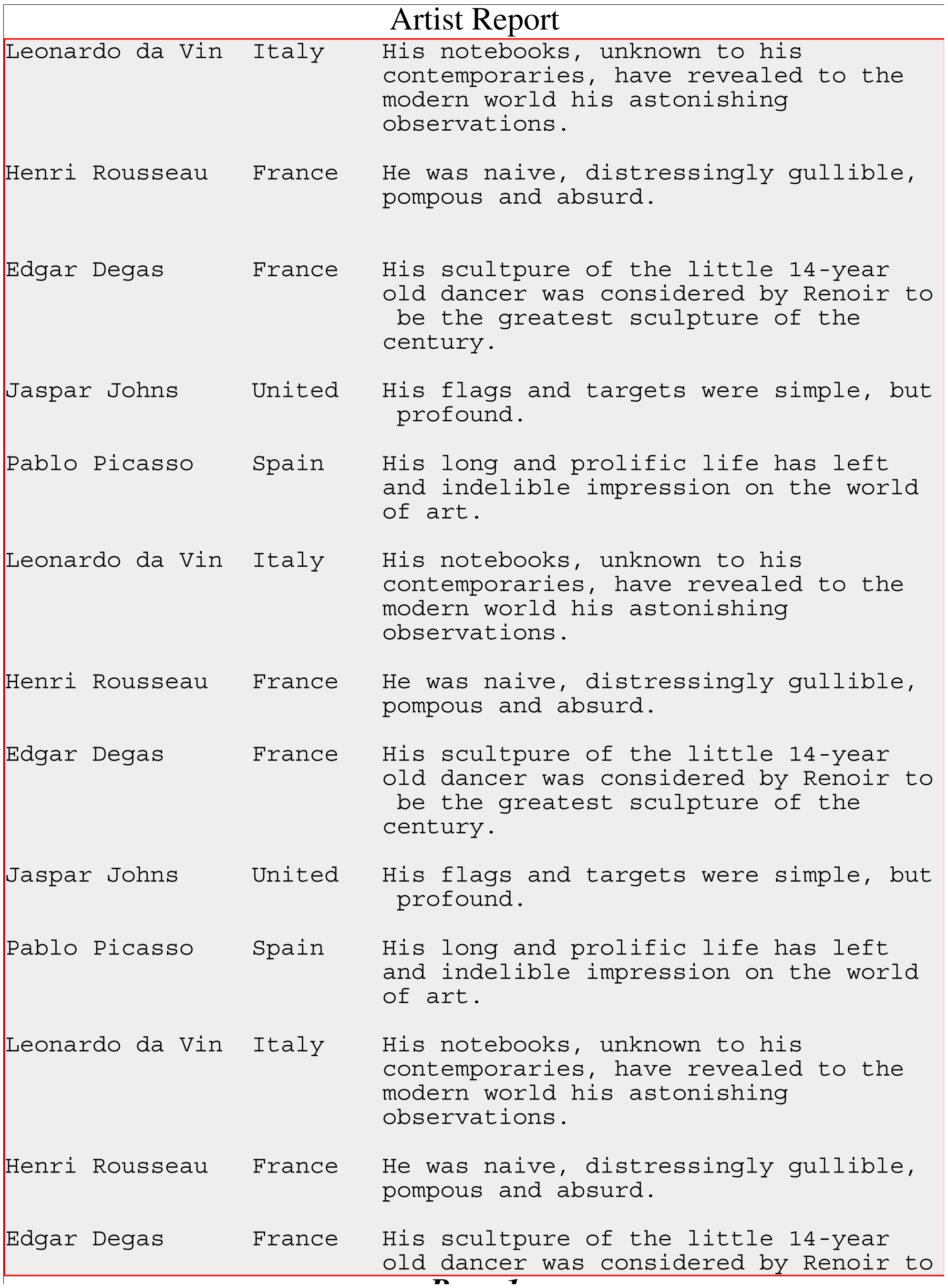
Page formatting The window you design for printing can be any size. The PrintPage class allows you to "tile" a large window onto multiple pieces of paper. The PrintPage class provides attributes that control the page format, as well as a set of methods that allow you to draw text and graphics on the working page (see PrintPage class in the Display Library online Help for information.) Also, you can include any widgets on a page template, even widgets, such as array fields, that do not display all their data at once. The printing classes provide the option of expanding these fields at print time.

When printing a window with a menu bar, iPlanet UDS ignores the menu bar and prints the window without it. Therefore, if you are designing a window specifically for printing, you should not include a menu bar on it.

After you have created your page template in the Window Workshop, you can print it by using the printing classes in the Display library. See PrintDocument class in the Display Library online Help for information about printing. Figure 7-10 illustrates how the window shown above will look when it is printed. (Note that data repeats in the sample page because the sample data set is so small.)

While this section provided information about creating windows especially for printing, remember that you can allow your end user to print any iPlanet UDS window by providing a print command that uses the Display library printing classes to print the current window.

Figure 7-10 Printed Version of Page Template



Artist Report

Leonardo da Vin	Italy	His notebooks, unknown to his contemporaries, have revealed to the modern world his astonishing observations.
Henri Rousseau	France	He was naive, distressingly gullible, pompous and absurd.
Edgar Degas	France	His scultpure of the little 14-year old dancer was considered by Renoir to be the greatest sculpture of the century.
Jaspar Johns	United	His flags and targets were simple, but profound.
Pablo Picasso	Spain	His long and prolific life has left and indelible impression on the world of art.
Leonardo da Vin	Italy	His notebooks, unknown to his contemporaries, have revealed to the modern world his astonishing observations.
Henri Rousseau	France	He was naive, distressingly gullible, pompous and absurd.
Edgar Degas	France	His scultpure of the little 14-year old dancer was considered by Renoir to be the greatest sculpture of the century.
Jaspar Johns	United	His flags and targets were simple, but profound.
Pablo Picasso	Spain	His long and prolific life has left and indelible impression on the world of art.
Leonardo da Vin	Italy	His notebooks, unknown to his contemporaries, have revealed to the modern world his astonishing observations.
Henri Rousseau	France	He was naive, distressingly gullible, pompous and absurd.
Edgar Degas	France	His scultpure of the little 14-year old dancer was considered by Renoir to

About Windows as Web Pages

iPlanet UDS allows you to convert any specially prepared window into a Web page. To prepare the window, you simply assign anchors and image URLs to the appropriate widgets on the window using the Widget > HTML Options... command.

The HTML Options… command enables you to set properties on the currently selected widget for use with the iPlanet UDS WindowConverter class. The WindowConverter class, in the iPlanet UDS Web Software Development Kit, allows you to convert an iPlanet UDS window into an HTML document (Web page).

Anchor URL property The Anchor URL property allows you to assign a hypertext link to the widget, so that when the window is converted into a Web page, the corresponding HTML element will be an anchor. The hypertext link must be in the form of a URL (Universal Resource Locator).

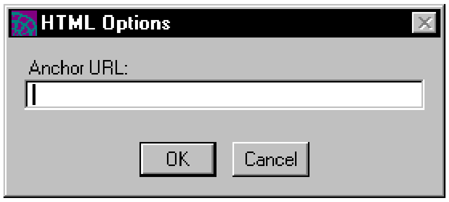
Push buttons that have anchors assigned to them are converted into HTML text strings with borders. The text string with a border simulates the appearance of a push button, with the HTML text string providing the button's label and the border providing the button's shape. The anchor for the push button provides the processing that takes effect when the button is "clicked."

Image Source URL property The Image Source URL property, available for picture fields, picture graphics, and picture buttons only, specifies the location of the image to be displayed on the Web page. The image file that you assign to a field must be located on the Web server. The format of the image file can be any format supported by the Web server. The Image Source URL must provide the address of the actual image file relative to the Web server.

If you do not assign an image file to provide the image for the picture on your window, or if the URL is incorrect, the Web browser displays a broken picture image in place of the field.

➤ To set the HTML options for a widget

1. Select the widget.
2. Select the Widget > HTML Options… command from the Widget menu.



3. In the HTML Options dialog, enter the Anchor URL and/or Image Source URL.

About Internationalizing Windows

You can set all text within windows and widgets displayed in a user interface through attributes and methods described in the Display Library online Help. Using iPlanet UDS international features, you can write code to set these attributes from messages read from a message catalog before the window is first displayed. If you design the windows using grid fields and other techniques for letting the interface adapt to the displayed contents, you can achieve excellent portability of interfaces across languages. See the *iPlanet UDS Programming Guide* for information about building a portable user interface. Note that you cannot change the base layout of a window using this technique.

Widget text and help All widgets that display static text, including menu items, have message numbers as properties in the Window Workshop so you can control the text for each widget individually. (Message number properties are not available on widgets that display text only at runtime, such as a data field.) In addition, the help topics for all fields and menu items also have message numbers, so you can create multilingual help for the window.

By default, all messages for a window are loaded from the same message set, which you specify using the Default Set Number property on the Window Properties dialog. If you do not specify a value for an individual widget's set number, iPlanet UDS will use the default set number specified for the window. If you do specify a set number for the individual widget, that set will be used. See "Setting Window Properties" on page 396 for information about specifying the default set number for the window properties dialog. See "About Internationalizing Widgets" on page 446 for information about specifying the message set and message number for an individual widget on a window.

Window title On the Window Properties dialog, the window's title has a message number and optional set number, so you can create a multilingual window title. If you not specify a set number for the tile, iPlanet UDS will use the default set number specified for the window. If you do specify a set number for the title, that set will be used.

See the *iPlanet UDS Programming Guide* for complete information on creating multilingual applications.

About Help for Windows

To provide help in your iPlanet UDS application, you can use iPlanet UDS's automatic help facilities, you can implement your own help facilities using the iPlanet UDS WinHelp method, or you can use any combination of the two.

iPlanet UDS automatic help facilities include the following:

Context-Sensitive Help for Individual Fields The Help Text command on the Widget menu allows you to assign a topic in the default help file to a field. iPlanet UDS displays this help topic when the input focus is on the field and the end user presses the Help key.

Float-Over and Status-Line Help for Individual Fields The HelpText command on the Widget menu allows you to specify both float-over and status-line help for the field. Float-over help is displayed alongside the field when the mouse pauses over the field. Status-line help is displayed in the window's status line when the mouse pauses over the field.

Help Menu The Menu Workshop provides three prefabricated commands for inclusion on the Help menu: Help Contents, Help Search, and Help on Help. The Help Contents and Help Search commands open the appropriate Help dialog, using the default help file you have specified for the window, or if none exists for the window, for the partition. The Help on Help command displays Windows standard Help on Help window.

iPlanet UDS also provides an About command, for which you can provide the appropriate processing. See "Using the Prefabricated Help Commands" on page 561 for information about implementing a Help menu for your window.

To implement your own help facilities, you can use the following iPlanet UDS features:

WinHelp Method The WinHelp method defined on the WindowSystem class provides an interface to the Windows Help API. You can use this method to access any Windows Help documents. See WinHelp method on Window class in the Display Library online Help for further information.

HelpRequest Event The HelpRequest event defined on the Window class is posted when the end user presses the Help key. You can provide any processing you wish in response to this event. See HelpRequest event on Window class in the Display Library online Help for further information.

Help Menu You can override the automatic behavior of the three iPlanet UDS Help commands by providing your own TOOL code to handle the Activate events on these commands. See "Using the Prefabricated Help Commands" on page 561 for information about implementing a Help menu for your window.

For complete details on implementing help for your window, see the *iPlanet UDS Programming Guide*.

Using the Window Workshop

This section discusses the following aspects of using the Window Workshop:

- entering the Window Workshop
- using the toolbar
- using the widget palette
- using the color palette
- using the widget status line
- leaving the Window Workshop

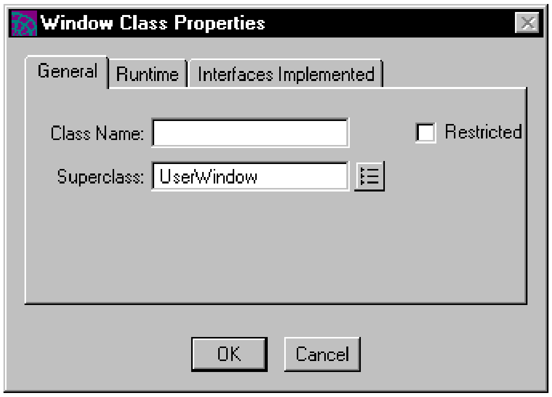
Entering the Window Workshop

You can enter the Window Workshop from the Class Workshop after creating a new window class or selecting an existing window class to modify.

➤ To create a new window class

1. In the Project Workshop, click the New Window Class tool or choose the Component > New Window Class command.

   The Window Class Properties dialog opens.



Class Name:

Superclass: UserWindow

☐ Restricted

2. On the General tab page, enter the name of a new class.

3. Select the superclass for the new window, either the iPlanet UDS UserWindow class or a custom window class of your own, and click the OK button. (See "Using the Class Properties Dialog" on page 306 for information on setting the other class properties.)

   iPlanet UDS creates the new window class and opens the Class Workshop.

4. In the Class Workshop, click the Window Workshop tool or choose the File > Window… command.

   You can now create a new form for your window class, or, if the window class has inherited a form, modify the existing form.

➤ To modify the window for an existing class

1. In the Project Workshop, double-click the window class you wish to modify.

   The Class Workshop opens.

2. In the Class Workshop, click the Window Workshop tool or choose the File > Window… command.

   You can now modify your existing window class.

The Window Workshop Window

The Window Workshop provides all the tools you need to create the form for your window. The workshop consists of:

Toolbar The toolbar provides a set of tools for creating compound widgets and editing your form.

Widget Palette The widget palette allows you to add simple widgets to your window by dragging them onto position on the form.

Color Palette The color palette lets you choose colors for your widgets by displaying the colors themselves (rather than selecting a color name).

Widget Status line The widget status line provides information about the currently selected widget and provides shortcuts for changing certain widget properties.

Form The form is an empty canvas where you layout the widgets you want to display on your window.

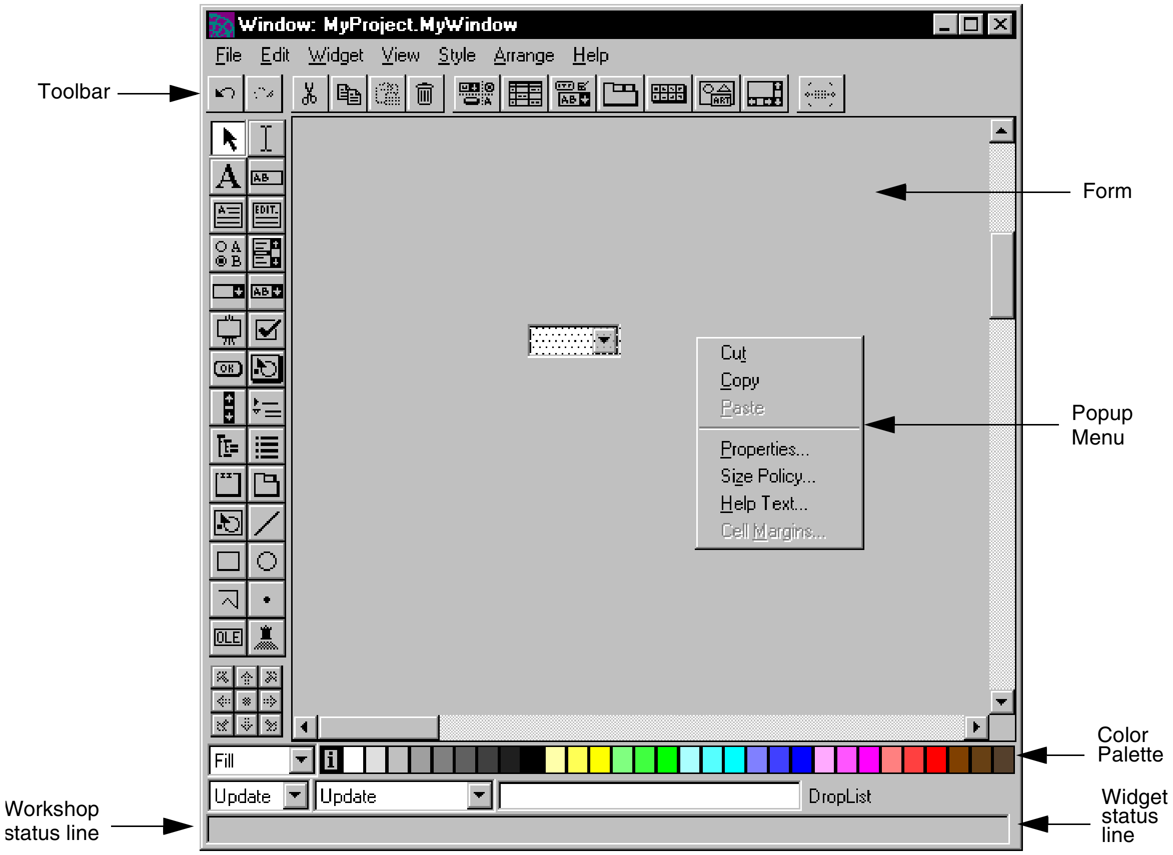
Menus The menus in the Window Workshop provide commands that duplicate the functions provided by the toolbar and palettes as well as special commands for formatting and arranging the widgets.

Popup Menu The Window Workshop provides a popup menu containing commonly used commands for building your window. To access the popup menu, use your window system's popup activation mouse button, key, or key combination. The following commands are available:

- Cut
- Copy
- Paste
- Properties...
- Size Policy...
- Help Text...
- Cell Margins...

Figure 7-11 illustrates the Window Workshop.

Figure 7-11 The Window Workshop



Viewing Window Workshop Tools

Because the menus in the Window Workshop provide equivalents to the toolbar, the status lines, and the two palettes, you do not need to display the entire workshop while you are working on your form. The View menu also allows you to add visual guidelines while you develop your window.

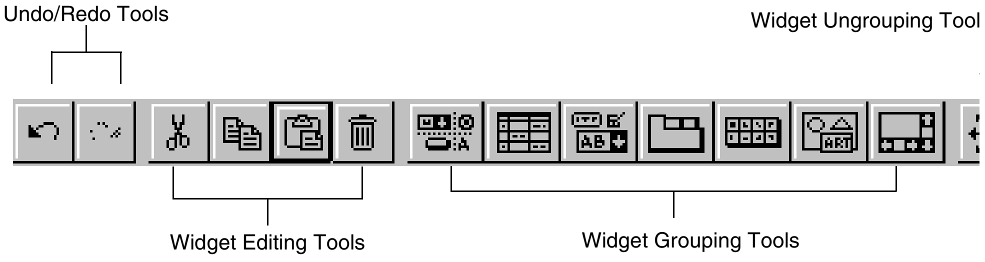
View menu The View menu in the Window Workshop provides the following set of toggles for displaying or hiding different parts of the workshop:

Command	Description
Grid	Displays or hides grid lines in the window form. When the grid is turned on, widgets placed on the form will snap to the nearest grid point.
Compound Field Lines	Displays or hides border lines around fields in a compound widget.
Toolbar	Displays or hides the toolbar.
Widget Palette	Displays or hides the widget palette.
Color Palette	Displays or hides the color palette.
Widget Status	Displays or hides the widget status line.
Status Line	Displays or hide the status line.

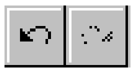
Using the Toolbar

The Window Workshop toolbar, shown in Figure 7-12, provides a selection of tools for editing your form and for creating compound widgets.

Figure 7-12 Window Workshop Toolbar



The Undo and Redo Tools



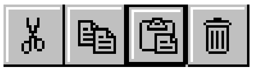
You use the Undo and Redo tools to undo or redo your most recent action in the Window Workshop.

The Undo tool undoes the last action you performed. You can repeat this as many times as you wish, as far back as your first action in the workshop, even removing information you recorded in properties dialogs.

The Redo tool re-does the last action you reversed by using the Undo tool.

The menu-based equivalents of the Undo and Redo tools are the Edit> Undo and Redo commands.

The Widget Editing Tools



The four editing tools, Cut, Copy, Paste, and Delete, allow you to manipulate the widgets on the form the same way you can cut, copy, paste, and delete text in a text editor.

➤ **To cut, copy, or delete a widget**

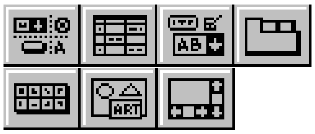
1. Select the widget.
2. Click the Cut, Copy, or Delete tool.

➤ **To paste a widget**

1. Click the Paste tool.
2. Select the position on the form where you want to paste the widget.

The menu-based equivalents of the widget editing tools are on the Edit menu.

The Widget Grouping Tools



You use the widget grouping tools to create a compound widget out of groups of widgets. To create a compound widget, you select a group of widgets on the form, and then click one of the widget grouping tools. Each tool represents a different compound widget. For information on creating compound widgets, see “Creating Compound Widgets” on page 405. The menu-based equivalent of the widget grouping tools is the Widget > Group Into command.

Using the Widget Palette

The widget palette provides tools for selecting widgets, entering and editing text, and creating simple widgets.

The Widget Selection Tool

The widget selection tool is for selecting widgets on the form. You can use it to select a single widget, multiple widgets, or compound widgets:

➤ **To select a single widget**

1. Activate the selection tool.
2. Click on the widget.

➤ **To select multiple widgets**

1. Activate the selection tool.
2. Enclose the widgets in a ghost box.

After you have selected one widget, you can use the Shift key to select or deselect multiple widgets. To select another widget, hold down the Shift key and click on the unselected widget. To deselect a selected widget, hold down the Shift key and click on the selected widget.

➤ **To select a compound widget**

1. Select one of the compound field's child widgets.
2. Hold down the Ctrl key and click on the child widget again.

   The parent widget will become selected.
3. To select the parent of the parent widget, hold down the Ctrl key and click the already selected parent.

   The parent's parent is now selected. You can repeat this process as many times as necessary to select each successive parent widget.

NOTE There is no menu-based equivalent for the selection tool. When the toolbar is hidden, the selection tool is always active.

See “Selecting Widgets” on page 410 for further information about selecting simple, compound, and multiple widgets.

The Text Graphic Tools

There are two text graphic tools: the Text Graphic Creation tool and the Text Graphic Edit tool. You use the Text Graphic Creation tool to place the text graphic on the form. You use the Text Graphic Edit tool to enter or edit the content of the text graphic.

➤ **To create a text graphic**

1. Click the Text Graphic Creation tool.

   The cursor turns into an upward pointing arrow.

2. Click the point on the form where you wish place the text graphic.

   iPlanet UDS adds the text graphic to the form using the phrase “Text Graphic.”

   iPlanet UDS automatically switches into edit mode so you can edit the content of the text graphic.

➤ **To edit a text graphic**

1. Activate the Text Edit tool.

   The arrow cursor appears.

2. Position the cursor over an existing text graphic on the form.

   The cursor turns into an I-beam.

3. You can enter, delete, or select text as you would in a word processing system.

For further information on text graphics, see “Creating a Text Graphic” on page 493.

The Widget Creation Tools

You use the widget creation tools to create simple widgets. The widget creation tools represent all the simple widgets you can create in the Window Workshop.

To create a simple widget, you simply click the tool whose widget you wish to create.

If the widget is fixed size, the cursor turns into a upward pointing arrow. In this case, move the cursor to the appropriate point on the form and click to indicate the widget's position.

If the widget is resizeable, the cursor turns into a cross hair. In this case, draw a ghost box on the form to indicate the widget size and position.

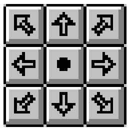
For further information about placing simple widgets on the form, see "Creating Widgets" on page 401.

Once you create a widget, the widget itself remains selected, its tool is deselected, and the widget selection tool is selected.

Sticky palette If you wish to create several widgets without having to reselect the appropriate widget tool, you can hold down the Control key when you select the new widget tool. The key sequence Ctrl-New Widget Tool keeps the tool selected until you select another tool.

The menu-based equivalent for the widget creation tools is the Widget > New submenu.

Using the Cell Gravity Tool



Each field within a grid field is enclosed in a cell (see "Creating a Grid Field" on page 517). The gravity of a cell determines how a child field in a compound field is aligned within its cell. You can use the Cell Gravity tool to realign a selected field within its cell.

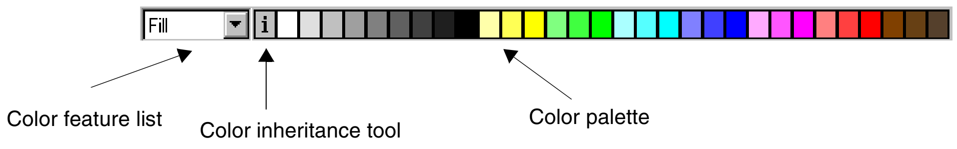
➤ **To align a field**

1. Select the field to be aligned.
2. Click the appropriate arrow in the Cell Gravity tool.

Using the Color Bar

The Window Workshop provides two color tools: the color feature list and the color palette. The color feature list, as shown in Figure 7-13, allows you to select a color feature for a widget: fill, pen, frame, or contrast pattern. The color palette offers an array of colors to use for color features.

Figure 7-13 Color Bar



You use the color palette to color the fill, pen frame, or contrast of a widget. For example, if you want to set the color of a rectangle's fill, you select the rectangle, select the color feature (Fill), and select a color for that feature. The table below describes the color features:

Color Feature	**Description**
Fill	Determines the background color. Used in conjunction with the contrast color on widgets with a fill pattern.
Pen	Determines the perimeter color for shapes and the text color for widgets with text.
Frame	Determines the frame color.
Contrast	Determines the pattern color of widgets with a fill pattern; used in conjunction with the fill color.

Dropping Colors

You can use the color selection tools to drop colors onto widgets.

➤ **To drop a color onto a widget**

1. Click on a color in the color palette.
2. Drag the package icon to the widget you want to color.
3. Release the mouse.

   The selected color feature for the widget changes color.

Setting color for a window You can set the background color for the window itself by dragging and dropping a color selection onto the window.

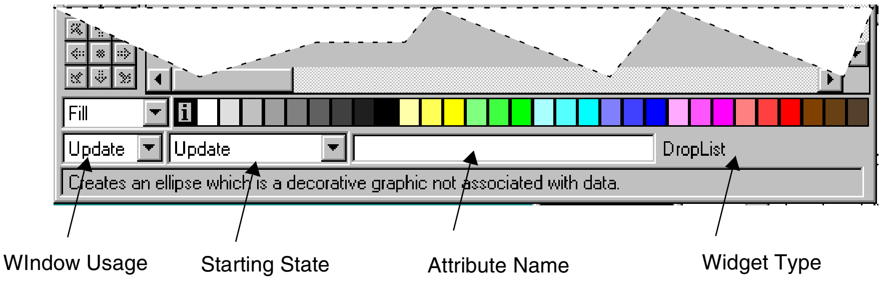
Making a widget inherit color from its parent You can make a child widget inherit the color scheme of its *parent* widget by using the Color Inheritance tool, shown in Figure 7-13. To change the color scheme of a child field to be the same as that for its immediate parent, drag the Color Inheritance tool and drop it on top of the child widget.

The menu-based equivalents of the color setting tools are the Style > Color Feature and Color commands.

Using the Widget Status Line

The widget status line includes four status fields that show the state of the window and of the widget you are creating. Figure 7-14 illustrates the status fields.

Figure 7-14 The Widget Status Line



Widget Status Field	Description
Window Usage	Determines the states for all the window's widgets. The menu equivalent is the Widget > For Usage command. There are six usages: Update—Mouse actions and keyboard input accepted; widgets themselves are changeable. View—View usage only; no keyboard input is accepted. Query—Query usage only; widget data is editable, but not widgets themselves. User1, User2, User3—User-defined usages. See "Setting Window Usages and Widget States" on page 398 for more information about window usages.

Widget Status Field	Description
Starting State	Controls how a widget reacts to user actions and how it displays itself. The menu equivalent is the Widget > Starting State command. See "About Widget States" on page 435 for more information about widget states.
Attribute Name	Displays the name for the currently selected widget. You can change the attribute directly in this field, as an alternative to setting it in the widget's properties dialog.
Widget Type	Identifies the type of the currently selected widget.

Access to Other Workshops

From the Window Workshop, you can access one other workshop:

Workshop	How to access it
Class Workshop	The File > Open Class… command opens the Class Workshop to display the definition of the class to which the current window belongs. If the class is already being displayed, the Open Class… command moves the input focus to the appropriate Class Workshop window.

Leaving the Window Workshop

To leave the Window Workshop, choose the File > Close command. When you leave the Window Workshop, iPlanet UDS creates the widget attributes for all widgets you have added to the form.

If you wish to remove all changes you have made to the form since you opened the Window Workshop, you can use the File > Cancel command. The Cancel command closes the workshop and ignores all work done since the workshop was opened, or since your last Save All command.

Designing a Window

To design a window, you set the window properties and use Window Workshop commands to define overall window appearance and behavior. You then use the Window Workshop to create the form for your window.

This section provides detailed information about the following topics:

- setting window properties
- setting window usages and widget states
- creating a form

To create a menu bar for a window, you use the Menu Workshop, described in Chapter 9, "Using the Menu Workshop." To open the Menu Workshop, choose the File > Menu… command.

Setting Window Properties

To set the window properties, choose the File > Window Properties… command.

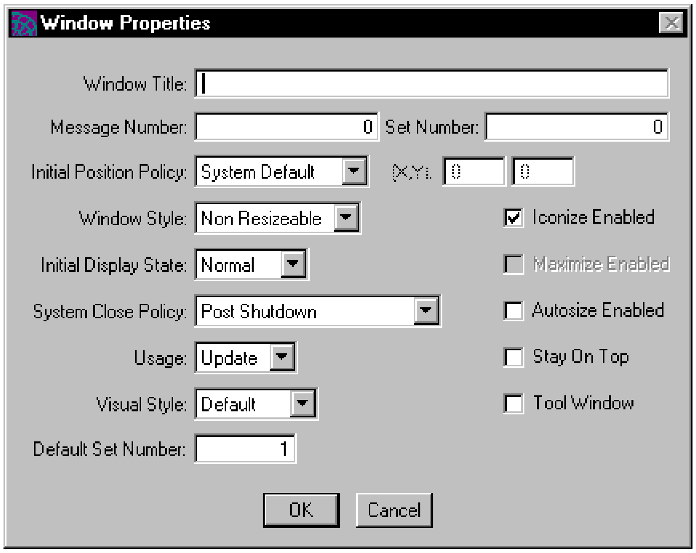
The Window Properties dialog, shown in Figure 7-15, allows you to set the properties defined in the following table.

Property	**Description**
Window Title	Sets the window's title, to be displayed in its title bar.
Message Number	Specifies the message number for the window's title, used for creating a multilingual window.
Set Number	Specifies the set number for the window title's message number. If unspecified, iPlanet UDS uses the default message set for the window.
Initial Position Policy	Sets the initial position a window should take in relation to other windows or to the screen. Window position policies are: System Default, Screen Centered, Primary Centered, Screen Relative, and Primary Relative.

Property	Description
X,Y	Sets the precise positioning for a window, according to the initial position policy. For example, a window using a position policy of Screen Relative and X,Y Position specifications of 10,10 would place itself at an offset of 10 mils horizontally and vertically from the upper left screen boundary. (Only the Screen Relative and Primary Relative position policies use X and Y Position information.)
Window Style	Sets the window’s style. A window can assume one of three styles: Resizeable, Non-Resizeable, or Frameless. See “Window Style” on page 365 for information on the window styles.
Iconize Enabled	Specifies whether or not to use a minimize box and corresponding window system menu item.
Initial Display State	Sets the window's initial display state—how it displays itself when it first runs. A window can assume one of the following states: Iconized, Hidden, Normal, Alternate.
Maximize Enabled	Specifies whether or not to use a maximize box and/or corresponding window system menu item.
Autosize Enabled	Specifies whether or not the Window and its form will automatically snap to a size which appropriately “frames” the form's children
Stay On Top	Specify that the current window remain on top of the other windows on the screen, even when the end user activates another window. This feature is useful for toolbars and status windows that should always be displayed. The Stay On Top toggle is turned off by default. The Stay On Top property is available only for Windows platforms. On other platforms, the property is ignored.
Tool Window	Specifies that the current window not be displayed in the window system’s task bar when the window is iconized. This feature allows you to create a secondary window for the application that is not displayed in the task bar when the main window for the application is iconized. The Tool Window toggle is turned off by default. This feature is available on Windows 95/NT only. On other platforms, the attribute is ignored.
System Close Policy	Sets the window’s close policy, which determines how the window responds to user action to close the window at the window system level. Choices are: Disabled, Post Shutdown, Post Finalize, and Post Shutdown Task Only.

Property	Description
Usage	Sets the initial window usage. A window can assume one of the following usages: View, Update, Query, User1, User2, User3.
Visual Style	Specifies the default field style for the window, either two dimensional or three dimensional. This takes effect for Windows and Motif only.

Figure 7-15 Window Properties Dialog



Window Title:			
Message Number:	0	Set Number:	0
Initial Position Policy:	System Default	(X,Y):	0 0
Window Style:	Non Resizeable	[✓] Iconize Enabled	
Initial Display State:	Normal	[ ] Maximize Enabled	
System Close Policy:	Post Shutdown	[ ] Autosize Enabled	
Usage:	Update	[ ] Stay On Top	
Visual Style:	Default	[ ] Tool Window	
Default Set Number:	1		

Setting Window Usages and Widget States

As described under "Usage Property" on page 370, window usages allow you to control how the end user interacts with the window. For example, imagine you want to provide a read-only form, where the user can view information but not update it. In the Window Workshop, you can change the window usage from Update to View.

The following instructions tell you how to define a window usage, set the starting state for a widget within the particular usage, and test a given usage for a window.

➤ **To define a window usage**

1. Choose the Widget > For Usage… command, and choose the usage you want to define from the submenu.

   A check appears beside the selected usage menu item.

2. Select the widgets whose states you want set for the usage.

3. Set the states for the widgets in the usage by choosing the states from the Widget > Starting State… submenu.

You can use the Starting State widget status field as a shortcut for setting the widget state:

➤ **To use the Starting State status field**

1. Select a widget.

2. In the Starting State status field, choose a starting state.

Testing a usage When you are finished setting the widget states for the usage, you can test the usage.

➤ **To test a window usage**

1. On the File > Test Usage submenu, select the usage you want to test.

   A check appears beside the selected usage menu item.

2. Choose the File > Test Window command.

   This opens the test window, where the window and its widgets behave as they would if the window were running in an actual application.

Changing window usages dynamically In the Window Workshop, you set only a window's initial usage. If you want to reset the usage based on its use in an application, you can write methods in TOOL to set the window's usage dynamically.

For example, imagine you have a data retrieval application, *Data Fetcher*. Data Fetcher offers its users a window to view or update database information shown in a variety of fields. The window must determine whether to allow a user to update the information in the fields, or only view it, depending on the identity of the user.

From user to user, Data Fetcher changes the accessibility of the widgets in the window by alternating between two usage settings for the window.

When a user qualified for data updating opens the window, Data Fetcher sets the window usage to Update. This sets the state for all the widgets in the window to update, meaning that all the fields are updateable.

When a user restricted to data viewing opens the window, Data Fetcher sets the window's usage to View Only. This sets the state for all the widgets in the window to View Only, meaning that all the widgets are visible, but not changeable.

For full information about changing a window's usage, or individual widget states dynamically, see the Display Library online Help.

Creating a Form

To design your window, you can place any number of simple fields or graphics on the window's form. Once the simple widgets are on the form, you can combine them into compound widgets.

After you place a widget on the form, you can open its properties dialog to customize it for your particular window. For example, you can place a push button on the form and then open its properties dialog to specify the label you want on the button.

When the basic form is laid out, you can refine it. You can resize and move the widgets, and align them with each other. You can also change widgets' format by setting their line width and style, and specifying color for individual widgets and sections of the form.

Automatic window sizing When your application opens the window, iPlanet UDS automatically sizes the window to accommodate the form. The top and left margins of the widget in the top-left-most position are also used as the bottom and right margins, providing equal spacing around the entire form.

See "Creating Widgets" on page 401 for detailed instructions on creating the individual widgets on the form.

Using a Placement Grid

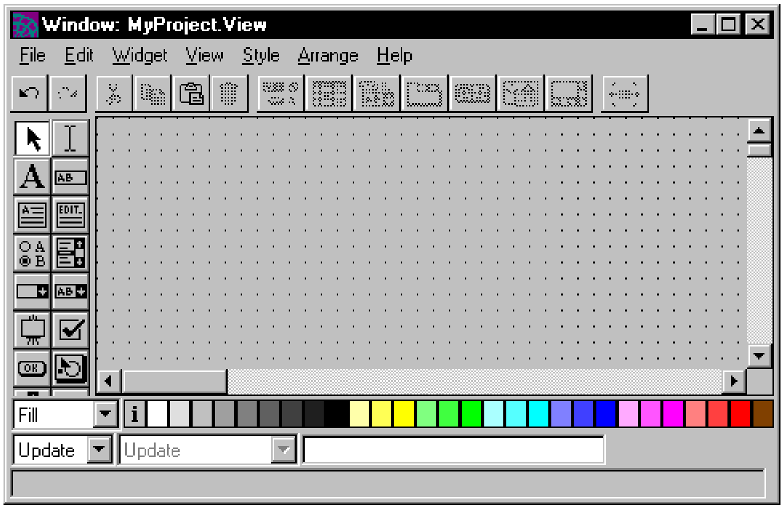
The Window Workshop provides an optional placement grid to help you align the widgets that you place on the form. When the grid is on, any widget that you place on the form automatically aligns to a matrix of displayed grid points.

You can use the grid to draw straight lines and to align groups of widgets quickly and easily.

➤ **To turn on the placement grid**

1. Choose the View > Grid command.

   iPlanet UDS displays the placement grid on the form.



Creating Widgets

All iPlanet UDS widgets are either fixed-sized or sizeable. The basic way you create and modify the widget depends on whether it is fixed-size or sizeable.

Fixed-size widgets A fixed-size widget is created in one standard size. When you add a fixed-size widget to the form, iPlanet UDS uses the widget's default contents and size. You then "resize" the widget by changing its contents on its properties dialog. For example, a radio list will automatically resize after you specify the exact number of buttons along with their labels.

Sizeable widgets A sizeable widget is created according to the size that you indicate on the form. When you add a sizeable widget to the form, you draw a rectangle to indicate the widget's basic size. You can then resize the widget as you wish by using resize handles.

The following sections describe how to add the following widgets to your form:

- simple widgets
- compound fields
- domain widgets

These sections are followed by general information about setting the properties of an individual widget and creating the attributes associated with widgets.

Creating Simple Widgets

For create a simple widget, the procedure you use depends on whether the widget is fixed-size or sizeable.

➤ **To create a fixed-size widget**

1. Select a widget, either from the widget palette or from the Widget > New menu.
2. Move the mouse onto the form.

   The cursor becomes an upward-pointing arrow.
3. Click the point for the upper-left corner of the widget to position the widget on the form.

   The widget appears on the form in its default dimensions at the point you clicked.

➤ **To create a sizeable widget**

1. Select a widget, either from the widget palette or from the Widget > New menu.
2. Move the mouse onto the form.

   The cursor becomes a cross hair.
3. Click on the form, and drag a ghost box to the desired dimensions.

   When you release the mouse button, the widget appears on the form.

   The widget palette provides tools for creating the following simple widgets.

Tool	Widget	Sizeable?	Description
	Text Graphic	Yes	A text graphic is an image of text—a decorative graphic not associated with data.
	Data Field	Yes	A data field is a one-line text field that can impose a formatting template upon the text—date, time, number, or text data—it contains.
	Text Field	Yes	A text field displays multiple lines of text in a scrollable field. A text field may use either word-wrapping or a horizontal scroll bar.
	Text Edit Field	Yes	A text edit field is a simple text editor, which can provide line numbers and/or icons along with the text as well as undo and redo utilities. A text edit field does not provide word wrapping.
	Radio List	No	A radio list displays a group of options as radio buttons, from which a user can choose one option.
	Scroll List	No	A scroll list presents a fixed group of choices in a scrolling vertical display, from which a user makes one selection.
	Drop List	No	A drop list presents a fixed group of choices in a menu-like display, from which a user makes one selection.
	Fillin Field	No	A fillin field provides a fixed element list for selection, but also permits text entry as an addition to the list. In a fillin field, a user can choose a value, or type his own.
	Picture Field	Yes	A picture field displays image data; if you change the image data associated with the field, the picture field displays a different image.
	Toggle Field	No	A toggle field is a widget that toggles a choice on and off. It consists of some text indicating the item or function to toggle, and a small button to toggle the choice.

Tool	Widget	Sizeable?	Description
	Push Button	No	A push button is a button designed to accept a mouse click and perform functions in response to the click. Every push button has a label, consisting of a single line of text, such as “Save” or “Cancel.”
	Picture Button	No	A picture button is a button that uses a bitmap image as a label. The button’s size is determined by the size of the image.
	Scroll Bar	Yes	A scroll bar is a widget you can program to work as a scroll bar or as a slider that lets the end user specify a numeric value within a range by dragging or clicking on the bar.
	Outline Field	Yes	Displays multi-column information, either as a flat list or as hierarchical information in an indented outline.
	List View Field	Yes	Displays a set of items, each consisting of an icon with a label, from which the end user can make selections.
	Tree View Field	Yes	Displays two-column hierarchical information in an indented outline, providing controls that let the end user expand and collapse the outline.
	Picture Graphic	Yes	A picture graphic displays an image directly and statically. Unlike a picture field, it does not map to image data.
	Line	Yes	A line is a decorative graphic not associated with data.
	Rectangle	Yes	A rectangle is a decorative graphic not associated with data.
	Ellipse	Yes	An ellipse is a decorative graphic not associated with data.
	Polyline	Yes	A polyline is a decorative graphic not associated with data. It is a multi-segment line, consisting of points and segments.
	Point	No	A point is a decorative graphic not associated with data. Points define lines and polylines. Points are fixed-size widgets.

Tool	Widget	Sizeable?	Description
♜	Domain	n/a	A domain widget is the form widget associated with the specified domain. If you place the domain widget inside an array field, its array widget is used. Otherwise, its form widget is used.
OLE	OLE Field	Yes	An OLE field displays OLE linked or embedded objects.

For details on creating any of these widgets, see the sections on creating specific widgets in Chapter 8, "Working with Widgets."

Using Repeat Mode

When you are creating simple widgets, you can use Repeat Mode to add any number of the same widgets to the form by simply clicking repeatedly on the form. In Repeat Mode, the last widget you added to the form continues to be selected on the palette. To add another widget of that type to the form, simply click on the form (or, for resizeable widgets, drag the ghost box). To change the type of widget you wish to add to the form, select a new item from the palette.

To turn on Repeat Mode, you can use the Widget > Repeat New command or the Repeat New preference for the Window Workshop. The Repeat New command overrides the Repeat New preference set for the workshop with the Workshop Preferences… command. See "Repeat New Preference" on page 430 for information on how to set Repeat Mode as the default for the Window Workshop.

Creating Compound Widgets

You create compound widgets by grouping together other widgets, both simple and compound.

➤ **To create a compound widget**

1. Select one or more widgets on the form.

   You can select several widgets either by dragging a ghost box around the desired widgets, or by clicking on single widgets successively while pressing the Shift key.

2. Click a compound widget tool, or choose a compound widget from the Widget > Group Into submenu.

The Window Workshop combines the widgets into the compound widget you selected.

NOTE Some compound widgets contain only certain widgets as components, or child widgets. A compound graphic, for example, contains only graphic widgets as child widgets. An array field can contain any widget except another array field. For details on compound widget components, see "Working with Compound Widgets" on page 501. The toolbar provides buttons for creating the following compound widgets.

Tool	Compound Widget	Description
	Grid Field	A grid field organizes widgets into rows and columns for consistent alignment across window systems.
	Array Field	An array field displays an array of widgets in uniform rows, where each row displays the same collection of widgets, but the widgets' data differ.
	Panel	A panel groups widgets together into a defined space on the form.
	Tab Folder	A tab folder displays a set of tab pages, which the end user can view one at a time. The end user selects a particular page to view by clicking on its tab.
	Palette List	A palette list acts as a collection of graphic buttons. A palette list is technically a simple widget, because it does not consist of other widgets. In the Window Workshop, however, you create a palette list as you would a compound widget.
	Compound Graphic	A compound graphic groups graphic widgets for easy manipulation as a single graphic.
	Viewport	A viewport offers a scrolling view of a single larger widget, such as a picture or a panel, which may be too large to display in its entirety on a form.

Two compound widgets, tab folder and panel, also appear on the widget palette in the Window Workshop. The tab folder palette item creates a default tab folder with three empty tab pages. The panel palette item creates an empty panel, into which add or move the widgets you want the panel to contain. See "Creating a Tab Folder" on page 504 for information about creating a tab folder. See "Creating a Panel" on page 502 for information about creating a panel.

Creating Domain Widgets

As described under "Domain Classes" on page 292, a domain is a special kind of class that combines a DataValue subclass with an iPlanet UDS widget. The domain's *form widget* specifies the class of widget, such as DataField or RadioList, that is used to display the data when the domain is added to a form. The domain can also have an *array widget*, which specifies the widget that is used when the domain is displayed within an array field.

To add a domain widget to a window, you can either drag the domain from the Project Workshop's browser and drop it onto the form in the Window Workshop, or you can use the domain tool on the widget palette.

➤ To create a domain widget with drag and drop

1. In the Project Workshop, select the domain you want to add to the form from the Project Component browser.
2. Drag the domain from the Project Component browser to the Window Workshop.
3. Drop the domain onto the form in the position where you want the widget.

   If you drop the domain onto an array field, the array widget is used. Otherwise, the form widget is used.

➤ To create a domain widget with the palette

1. In the Window Workshop, click the New Domain tool on the widget palette.
2. Click the starting point for the domain widget.

   The Choose a Domain dialog appears.

3. On the Choose a Domain dialog, choose the appropriate domain name.

   If you selected a starting point within an array field, the array widget for the domain will appear within the array. Otherwise, the form widget will appear on the form.

Note that you can use a domain as the mapped type for any appropriate widget that is already on the form. In this case, iPlanet UDS uses the widget that is already on the form, rather than the domain's form or array widget.

Setting Widget Properties

Every widget has a properties dialog, which allows you to define the appearance and behavior of a widget. For information on the individual properties available for specific widgets, see Chapter 8, "Working with Widgets."

➤ **To open the properties dialog for a simple field or graphic**

1. Double-click the widget.

or

1. Select the widget.
2. Choose the Widget > Properties…command.

➤ **To open the properties dialog for a compound widget**

1. Double-click the widget's frame or background

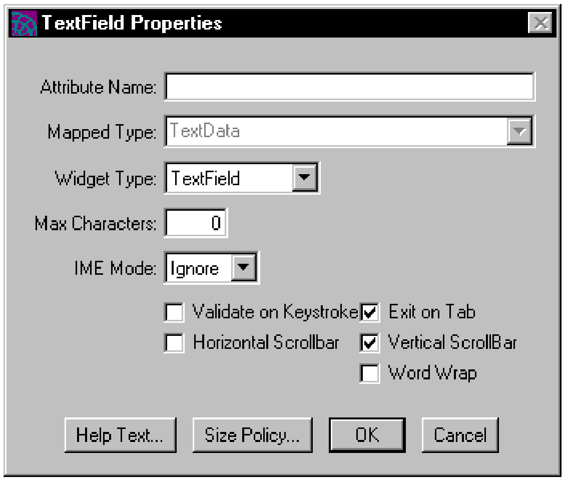
   A compound widget's background is any point within its frame that does not belong to its child widgets.

or

1. Select the compound widget.

2. Choose the Widget > Properties… command.

The figure below shows a properties dialog for a text field.


TextField Properties

Attribute Name:
Mapped Type: TextData
Widget Type: TextField
Max Characters: 0
IME Mode: Ignore

☐ Validate on Keystroke
☑ Exit on Tab
☐ Horizontal Scrollbar
☑ Vertical ScrollBar
☐ Word Wrap

When you want to resume working in the Window Workshop, you must close the properties dialog.

Setting the Properties for a Window

To set the properties for your window, choose the File > Window Properties… command. For more information on setting a window's properties, see "Setting Window Properties" on page 396.

Note that every widget also has a Help Properties dialog (described under "Multilingual Help for Widgets" on page 448), which allows you to specify help for the individual widget and a Size Properties dialog (described under "Size Policies" on page 439), which allows you to specify size policy properties for the widget.

Creating Attributes

Generally, iPlanet UDS does not create the attributes for the widgets you add to the form until you close the Window Workshop. However, if you wish to create them while you are still working on the form, you can use the File > Compile command.

Note that you never need to use the Compile command. The Compile command is provided for your convenience, so you can keep the list of attributes in the Class Workshop synchronized with the attributes you are adding to the window class when you add widgets to the form.

Selecting Widgets

To modify, copy, or delete a widget, you must first select it—a selected widget is highlighted. You can select a simple widget, a compound widget, or a group of unrelated widgets as described below.

Selecting Single Widgets

To select a single widget, you click the widget.

Selecting compound widgets To select a compound widget, you click the widget's frame or background. You can also select a compound field by selecting the "parent" of the current field.

➤ **To select a parent widget**

1. Select a child field of the compound widget.
2. Press Ctrl-Click or choose the Edit > Select Parent command.

If the compound widget belongs to another compound widget, you can select the compound widget's parent in the same way.

Selecting a Group of Widgets

There are three ways to select a group of widgets. You can:

- use a ghost box to enclose a group of adjacent widgets
- use Shift-Click to select the widgets one at a time—you can pick and choose from any widgets on the form
- use Edit > Select Siblings or Select Children commands—these commands let you select all the widgets within a compound field

Using a Ghost Box

You can select a group of widgets all at once by using the mouse to drag a ghost box around the group of widgets.

➤ **To use a ghost box**

1. Move the cursor outside the group of widgets that you wish to select.
2. Hold down the mouse button. A ghost box is anchored at that position.

3. Drag the ghost box to completely enclose the widgets you wish to select.

The selection group includes only the widgets that are completely enclosed by the ghost box. If a portion of a widget is outside of the box, that widget will not be selected.

Using Shift-Click

To select a group of widgets one at a time, select the first widget as usual and then use Shift-Click to add widgets to the group.

➤ **To use Shift-Click**

1. Select the first widget.
2. Position the cursor over the second widget and press the Shift key and click the widget.
3. Repeat Step 2 as many times as desired to add widgets to the selection group.

You can also use Shift-Click to deselect a selected widget. Simply position the cursor over the selected widget and press the Shift key and click the widget.

Using the Select Commands

The Select Siblings and Select Children commands let you select all the widgets within a compound field with one simple command. The Select Siblings command selects all the widgets that are contained within the same compound field as the selected child widget. The Select Children command selects all widgets contained within the selected compound widget.

Modifying Widgets

After you place a widget on the form, you can make any of the following changes to it:

- remove it from the form (cut it or delete it)
- make a copy of it (duplicate it or copy it onto the clipboard)
- move it from one position to another
- resize it
- convert it to another widget type
- sets its state

Removing Widgets

You can remove a simple widget, a compound widget, or a group of widgets.

➤ **To remove widgets**

1. Select the widget or group of widgets to be removed.
2. Choose the Edit > Delete or Cut command, or click the Delete tool.

   The Delete command removes the widget from the form. The Cut command moves the widget to the clipboard.

Copying Widgets

You can copy a simple widget, a compound widget, or a group of widgets. To copy widgets, you can either use the Duplicate command (which makes a copy of the widget on the form) or the Copy and Paste command sequence which copies the widget onto the form through the clipboard.

➤ **To use the Duplicate command**

1. Select a widget or group of widgets to duplicate.
2. Choose the Edit > Duplicate command.

iPlanet UDS creates a duplicate of the selected widget or group on the form, slightly offsetting the duplicate from the original.

➤ **To use the Copy and Paste commands**

1. Select the widget or group of widgets to be copied.
2. Choose the Edit > Copy command.
3. Choose the Edit > Paste command.
4. Click the starting point for the new widget or group of widgets.

Moving Widgets

You can move a simple or compound widget from one part of the form to another. You can also move an independent widget into or out of a compound widget or rearrange the positions of widgets within a compound widget.

Moving simple widgets To move a simple widget, you simply drag the widget to its new location.

➤ **To move a simple widget**

1. Position the cursor on the widget you want to move.
2. Click on the widget, and, holding down the mouse button, drag the widget to its new position.
3. Release the mouse button.

Moving compound widgets To move a compound widget, you must be sure to select the compound widget's frame or background for dragging.

➤ **To move a compound widget**

1. Position the cursor on the compound widget's frame or background.
2. Hold down the mouse button and drag the widget to its new position.
3. Release the mouse button.

Moving components of compound widgets Compound widgets offer differing options for moving their child widgets. For details, see the specific compound widget description in Chapter 8, "Working with Widgets."

Resizing Widgets

You can resize any simple or compound widget that displays resize handles when you select it. When selected, resizeable widgets display resize handles, while fixed-size widgets simply reverse their display.

➤ **To resize a widget**

1. Select the widget so the resize handles are displayed.
2. Position the cursor on the handle that is on the side or corner that you wish to expand or shrink.
3. Drag the widget's boundary to its new position.
4. Release the mouse button.

Converting Widgets

You can convert a widget you created to different kind of widget by using the Widget > Convert To command. Only a subset of widgets can be converted; if this command is not available for a particular widget, there will be no submenu for the Convert To command. If the submenu is available, it will list the types of widgets to which you can convert the selected widget.

➤ **To convert a widget**

1. Select the widget so the resize handles are displayed.
2. Choose the Widget > Convert To command.
3. Select the appropriate widget type to which to convert the selected widget.

Setting a Widget State

As described under "Usage Property" on page 370, the state of an individual widget determines how the end user interacts with the widget.

The Starting State status field in the Window Workshop contains a drop list of available states for the current window usage. Remember, a widget state is always associated with a window usage, which you can set in the Window Usage status field. For example, a widget can have a different state for each window usage setting—if the window usage is set to Update, you can specify that a certain widget is active, but cannot be modified.

The Widget > Starting State and For Usage commands are menu equivalents of the Starting State and Window Usage status fields.

➤ **To set a widget state**

1. Select a widget.
2. In the Window Usage status field, choose the window usage for which you wish to set the widget's state.
3. In the Starting State status field, choose a starting state for the selected widget.

See "About Widget States" on page 435 for further information about widget states. See "Setting Window Usages and Widget States" on page 398 for information about window usages.

Formatting Widgets

The Style menu contains commands that enable you to set the style of a widget, including its frame and fill colors, its fill pattern, and, if appropriate, its font.

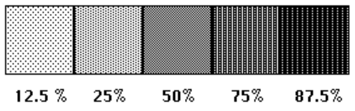
Transparency for Graphic Fields

The Style > Transparent command determines whether objects beneath a graphic field are visible through the graphic field. When the Transparent command toggled on for a graphic field (simple or compound), a graphic field allows objects in the layers underneath it to show through its background.

Creating Fill Patterns

The Style > Fill Pattern command determines a widget’s contrast pattern. A contrast pattern is a pattern of lines overlaid upon a shape’s fill color, serving as a contrast to the fill color.

Figure 7-16 Fill Patterns



Fill patterns have their own colors, known as contrast colors, which are defined by the Contrast color feature.

The Fill Pattern command has the following options:

Value	Definition
None	No contrast pattern.
Vertical Lines	A pattern of vertical lines.
Horizontal Lines	A pattern of horizontal lines.
Box	A pattern of lines crossing one another to form a box lattice.
Diamond	A pattern of lines slashing across one another to form a diamond lattice.
Slash Up	A pattern of lines slashing at a 45 degree angle from the lower left to the upper right.
Slash Down	A pattern of lines slashing at a 45 degree angle from the upper left to the lower right.
Contrast 12.5%	12% grey (shown above).
Contrast 25%	25% grey (shown above).
Contrast 50%	50% grey (shown above).
Contrast 75%	75% grey (shown above).
Contrast 87.5%	87.5 grey (shown above).

➤ **To select a fill pattern**

1. Select the widget.
2. Choose the Style > Fill Pattern command, and then choose the appropriate pattern from its submenu.

➤ **To select a contrast color**

1. Select the widget that is filled with a pattern.
2. Choose the Style > Color Feature > Contrast command.
3. Choose the Style > Color command and choose a color from the submenu.

➤ **To select a background color**

1. Select the widget that is filled with a pattern.
2. Choose the Style > Color Feature > Fill command.
3. Choose the Style > Color command and choose a color from the submenu.

Controlling Line Style and Weight

You can control the weight of the frame surrounding a widget. For Windows and Motif, you can also specify the visual style of the widget, either two dimensional or three dimensional. This overrides the visual style setting for the window (see "Visual Style Property" on page 372 for information).

You can also set different styles and line weights for line widgets.

➤ **To modify the frame weight of a widget**

1. Select the frame.
2. Choose the Style > Frame Weight command.
3. Choose the appropriate weight.

➤ **To set the visual style of a widget**

1. Select the widget.
2. Choose the Style > Visual Style command.
3. Choose the visual style.

➤ **To modify the weight of a line widget**

1. Select the line widget.
2. Choose the Style > Line Weight command.
3. Choose the appropriate line weight.

You can use the Line Weight command in conjunction with the Line Style command.

➤ **To modify the style of a line widget**

1. Select the line widget.
2. Choose the Style > Line Style command.
3. Choose the appropriate line style.

You can use the Line Style command in conjunction with the Line Weight command.

Setting Widget Fonts

Unless you explicitly set a widget to use a particular font, it will use the host window system's default display font. If you do want to specify the font to use for a widget, you have a choice between using a portable font, system font, or extended font.

Portable fonts A portable font is consistent across window systems and has specific characteristics that you can control. Use a portable font when you want to specify the size and style of the font, rather than just its face. A portable font consists of three elements:

- face—the actual typeface, such as Helvetica or Courier
- size—the point size of the font
- style—determines whether the font is bold or italic

➤ **To set the font for a widget to a portable font**

1. Select the widget or widgets whose font you wish to change.
2. Choose the Style > Portable Font command.
3. On the Portable Font command's submenus, choose the typeface, type size, and style for the font.

System fonts A system font is the particular window system's version of the font. The window system uses the system font to display window titles, dialogs, menus, button labels, and other widgets for which it makes standard representations. System fonts differ from one window system to another.

➤ **To set the font for a widget to a system font**

1. Select the widget or widgets whose font you wish to change.
2. Choose the Style > System Font command.
3. On the System Font command's submenu, choose the typeface for the system font.

Extended fonts An extended font is a font that might vary across codesets and display systems. Unlike the portable or system fonts, an extended font is set up as part of the installation to represent fonts that are used only in applications that do not require portability of user interfaces across codesets. See the *iPlanet UDS Programming Guide* for further information about extended fonts.

➤ **To set the font for a widget to an extended font**

1. Select the widget or widgets whose font you wish to change.
2. Choose the Style > Extended Font command.
3. On the Extended Font command's submenus, choose the typeface, type size, and style for the font.

Arranging Widgets

The Arrange menu contains commands that enable you to align widgets with each other, automatically resize widgets to uniform sizes, and control how widgets are stacked in layers on top of one another.

Sizing and Aligning Widgets

You can choose two or more widgets and automatically make them a uniform size using the Arrange > Size To command. The Size To submenu gives you the options of making each widget the same size as the tallest, shortest, widest, or narrowest widget. In addition, you can align a group of widgets using the Align To command. The Align To submenu lets you choose to align two or more widgets along the top, bottom, left, or right.

➤ **To make a group of widgets uniform size**

1. Select the widgets you wish to resize.
2. Choose the Arrange > Size To command.
3. Pick the appropriate sizing option from the submenu.

➤ **To align a group of widgets**

1. Select the widgets you wish to align.
2. Choose the Arrange > Align To command.
3. Pick the appropriate alignment option for the widgets.

Widget Partnership Commands

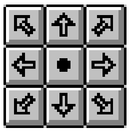
iPlanet UDS allows you to link any number of resizeable fields together in a height or width partnership. When two or more fields are in a height partnership, iPlanet UDS uses the height of the field with the largest minimum height in the partnership to determine the height for all the fields in the partnership. When two or more fields are in a width partnership, iPlanet UDS uses the width of the field with the largest minimum width in the partnership to determine the width for all the fields in the partnership.

You can also link any number of grid fields in a row or column partnership. Column and row partnerships are very useful when your form contains two grid fields that are separated by other widgets. When one grid field is above another, you can make their corresponding columns exactly the same width by joining the two grid fields into a column partnership. When the grid fields are side by side, you can make their corresponding rows the same height by joining the two grid fields into a row partnership.

The table below briefly describes the partnership commands. For details about using these commands, see the *iPlanet UDS Programming Guide*.

Command	**Description**
Fields Into Height Partnership	Links selected fields into a height partnership.
Fields Into Width Partnership	Links selected fields into a width partnership.
GridFields Into Row Partnership	Links selected grid fields into a row partnership.
GridFields Into Column Partnership	Links selected grid fields into a column partnership.
Remove Field From Height Partnership	Removes selected fields from height partnership.
Remove Field From Width Partnership	Removes selected fields from width partnership.
Remove GridField From Row Partnership	Removes selected grid field from a row partnership with other grid fields.
Remove GridField From Column Partnership	Removes selected grid field from a column partnership with other grid fields.

Aligning Cells in a Grid Field



Each field within a grid field is enclosed in a cell (see "Creating a Grid Field" on page 517). The gravity of a cell determines how the field is aligned within the cell. You can use the Arrange > Cell Gravity command to specify how to align a field. You can also use the Cell Gravity tool, shown at left, to perform the same functions.

The Cell Gravity command has the following options:

Value	Definition
Bottom Center	Bottom of the cell, centered horizontally.
Bottom Left	Bottom, left corner of the cell.
Bottom Right	Bottom, right corner of the cell.
Center	Center of the cell.
Default	Center of the cell.
Middle Left	Left side of the cell, centered vertically.
Middle Right	Right side of the cell, centered vertically.
Top Center	Top of the cell, centered horizontally.
Top Left	Top, left corner of the cell.
Top Right	Top, right corner of the cell.

You can also use the Cell Gravity tool on the Window Workshop widget palette to align fields within a cell. See "Using the Cell Gravity Tool" on page 392 for information.

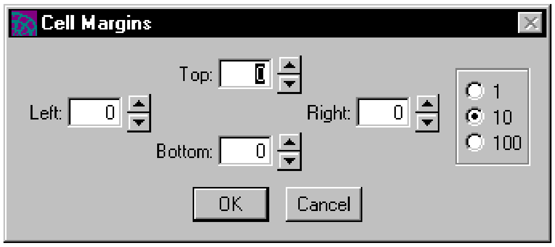
Cell Margins

You can add margins around widgets within the cells of a grid field. You can also control the margins between the widget borders and the cell borders.

➤ **To set margins**

1. Select the cell.
2. Choose the Arrange > Cell Margins command.

   The Cell Margins dialog opens.



3. Type in values or use the arrow to increment or decrement each setting. Set the increment amount with the radio buttons to 1, 10, or 100 mils.
4. Click the OK button.

The table below describes the area to which each setting applies.

Margin Setting	Description
Left Margin	The left setting determines in mils the margin between the widget's left edge and the border of the left cell. The actual margin is the greater of two settings: the left margin for the grid field, and left margin for the widget in the cell.
Right Margin	The right setting determines in mils the margin between the grid field's right cell dividers and the widgets the cells contain. The actual margin is the greater of two settings: the right margin for the grid field, and right margin for the widget in the cell.
Top Margin	The top setting determines in mils the margin between the grid field's top cell dividers and the widgets the cells contain. The actual margin is the greater of two settings: the top margin for the grid field, and top margin for the widget in the cell.

Margin Setting	Description
Bottom Margin	The bottom setting determines in mils the margin between the grid field's bottom cell dividers and the widgets the cells contain. The actual margin is the greater of two settings: the bottom margin for the grid field, and bottom margin for the widget in the cell.

Stacking Widgets in Layers

The Arrange > Send To Back and Bring to Front commands enable you to move fields on top of or underneath other fields, and to move graphics on top of or underneath other graphics. Remember the form consists of two layers, the graphic layer and the field layer. The graphic layer is *always* below the field layer. Therefore, if you place a field on top of a graphic, it will cover the graphic. And if you place a graphic on top of a field, it will move below the field.

➤ **To send a widget behind another widget**

1. Select the widget to send to the back.
2. Choose the Arrange > Send to Back command.

➤ **To send a widget to the front**

1. Select the widget to send to the front.
2. Choose the Arrange > Bring to Front command.

Working with Inherited Windows

iPlanet UDS allows you to create a window subclass that inherits its form and menu, as well as the methods, attributes, events, and event handlers, from a window superclass.The window superclass can define standard company-wide or application-wide features, such as menus and basic window formatting. The window subclass can extend the inherited window by adding new widgets to the form and menu, and by modifying the inherited widgets. There can be any number of window subclasses for a single window superclass.

Like any subclasses, the window subclasses inherit the attributes, methods, events, and event handlers defined for their window superclass. However, this section provides information only about how to work with the inherited form and menu using the Window Workshop. For information about writing the TOOL code for inherited windows, see the *iPlanet UDS Programming Guide*.

iPlanet UDS allows you to modify the window superclass at any time, even after the window subclasses have been created. Any changes you make to the superclass are automatically inherited by its subclasses. These changes take effect when the subclass is compiled.

If any of the subclasses have modified the inherited widgets, iPlanet UDS automatically merges the superclass changes with the subclass changes. However, if there are conflicts that cannot be resolved, the subclass windows will not compile. Modifying a subclass window is described in further detail below.

If a window subclass is open and being modified at the same time the superclass window is being modified, the changes in the superclass window will not show up until you compile your project, compile both the window classes, or close both instances of the Window Workshops.

Working with a Subclass Window

If your window class is the subclass of another custom window class (not the UserWindow class), your new window inherits the form and menu bar from the superclass.

You can make the following changes to the subclass window:

- add new widgets
- move inherited widgets
- change any widget properties, except the name and mapped type (if mapped)
- delete inherited widgets

The following sections provide information about making these changes.

Adding New Widgets

You can add new widgets anywhere on the inherited form or menu. The new widgets you add become new attributes for the subclass.

What if the subclass adds new widgets and later on its superclass adds new widgets? When this happens, iPlanet UDS merges all the changes so that the widgets added both by the superclass and the subclass will appear in the subclass window. However, you should always check the subclass window after changes have been made to its superclass to ensure that the subclass window is properly formatted.

Of course, not all modifications can be merged. For example, if the superclass adds a widget with the same name as a newly added widget in one of the subclasses and the two widgets do not have compatible types, there is no way to merge the changes. In this case, the change to the superclass takes precedent—the subclass will get a compile error.

Moving Inherited Widgets

You can move inherited widgets anywhere you wish. You can even move an inherited widget from one compound field (or menu) to another. This move changes the widget's parent, but has no effect on its relationship to the superclass that defined it.

Note that if you change the position of an inherited widget in the subclass and the superclass later changes the position of the same widget, the subclass will not inherit the position change from the superclass. The only way to synchronize the positioning of the two widgets is to change the position of the widget in the subclass by hand to the same position it has in the superclass. When the positions of the two widgets are synchronized, the subclass widget will inherit future position changes from the superclass.

Changing Widget Properties

You can change any of the widget properties for an inherited widget, except the widget's name and mapped type.

Note that if you change the property of an inherited widget in the subclass and the superclass later changes the same property of the same widget, the subclass will not inherit the property change from the superclass. The only way to synchronize the properties of the two widgets is to change the property of the widget in the subclass by hand to the same value it has in the superclass. When the properties of the two widgets are synchronized, the subclass widget will inherit future property changes from the superclass.

If the superclass originally defined the widget with an abstract widget type (see "Creating a Superclass Window" on page 426), the subclass can convert the widget type to any of the subclasses of the abstract type. For example, if the superclass defined a radio field as having the abstract widget type of ListField, the subclass can convert the radio field to any of the other ListField subclasses: DropList, PaletteList, or ScrollList.

➤ **To change the widget type**

1. Select the widget whose type you wish to change.
2. Choose the Widget > Convert To command.
3. Select the new type for the widget.

Deleting Inherited Widgets

You can remove any inherited widget. Deleting an inherited widget from the form or menu removes the widget from the window, but does not remove the associated attribute from the class. Therefore, you can add the same widget back to the window by placing it again on the form or menu and giving it the same name.

Creating a Superclass Window

In the Window Workshop, there is no difference between creating a window that will be used as a superclass and creating an ordinary window, with one small exception. If a window is going to be used as a superclass window, you must name *all* the widgets in the superclass.

iPlanet UDS provides one special feature that you may wish to use when creating a superclass window: the ability to include "abstract" widgets on the superclass window so that subclass windows can convert the widgets to any appropriate concrete widget.

Normally, the widgets on a window are one of the concrete widget class from the Display library, such as RadioList or TextField. The type of each widget corresponds to widget palette item you used to create that widget.

However, for some fields, iPlanet UDS provides the option of changing the widget's type to an abstract class, such as CharacterField or ListField. The window subclass can then convert the generic widget to any of its subclasses. For example, if you add a radio list to the superclass window, you could change its type to the generic ListField class. The subclass window could then convert the generic ListField class to any of its subclasses: DropList, RadioList, ScrollList, or PaletteList.

To change the type of a widget on the superclass window, use the Widget Type property on the widget's properties dialog. To change the type of the widget on the subclass window, select the appropriate type from the Widget Type drop list. Note that changing the widget's type to an abstract type does not change the appearance of the widget; it simply makes it possible for the subclass windows to convert that widget to another type.

Testing a Window

You can test a window in the Window Workshop by putting the workshop into Test mode. In Test mode, the workshop opens the user window that you have created. This window looks exactly as it will to end user and behaves as it would in actual use. If you have defined a menu bar for the window, this will be displayed as part of the window.

While the Window Workshop is in Test Mode, you can only test the function of the window; you cannot conduct window-building activities such as adding, removing, or altering widgets.

Entering test mode To enter test mode, choose the File > Test Window command, switching the toggle on.

Leaving test mode To leave test mode, and return to editing your window, choose the File > Test Window toggle again, switching the toggle off.

Testing Window Usages

When you test a window, you are always testing the window in a certain usage. A window usage determines the states of its widgets on a collective basis. The default window usage is Update. For more information about creating and implementing window usages, see Chapter 7, "Using the Window Workshop." You can test a window in any of six usages, as shown in the File > Test Usage… submenu.

➤ **To test a window in a specific usage**

1. Choose the usage you want to test in the File > Test Usage… submenu.
2. Choose the File > Test Window command, switching the toggle on.

Importing and Exporting a Window

To copy a window from one window class to another, you use the Window Workshop Import… and Export… commands. These commands let you export a window from one class into a file and then import the window stored in that file into another class.

The Export… command in the Window Workshop allows you to export a window definition to a standard file (portable across operating systems), which you can then import into another iPlanet UDS window class using the Import… command. The Import… command in the Window Workshop replaces the current window in the window class with the window in the specified text file.

Importing a Window

The Import… command replaces the current window with the window in the file you specify. If there is a menu bar, this will also be imported.

The window that you import must have been created either by the Export… command in the Window Workshop or by the iPlanet UDS Fscript utility.

➤ **To import a window**

1. Choose the File > Import… command.
2. Choose a window file (a file with an .fsw suffix) to load.

After the window has been successfully imported, iPlanet UDS displays the window in the Window Workshop. If there is an error, such as a bad file, iPlanet UDS displays an error message.

Exporting a Window

The Export… command writes the definition of the current window into a standard file (portable across operating systems). iPlanet UDS stores the window in a file with the name you specify and an .fsw suffix.

➤ **To export a window**

1. Choose the File > Export… command.

   If the window has an associated menu, you are prompted for whether you want the export to include or exclude the menu bar.

2. In the file selection dialog, specify the name of the file to contain the window definition. If you give the name of an existing file, the Export… command overwrites the file.

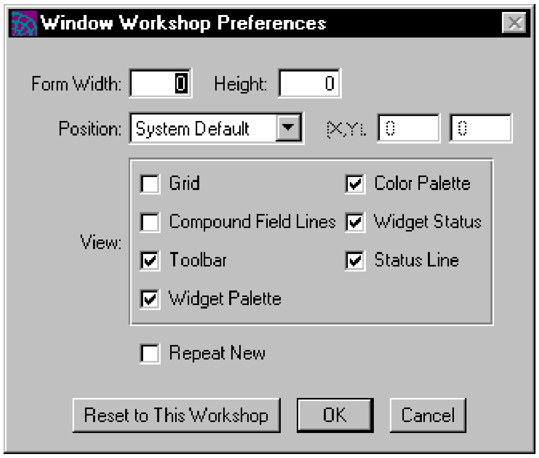
While the window is being exported, iPlanet UDS displays a message indicating that the class is being written to the specified file and prevents all input until the export is complete.

Setting Window Workshop Preferences

The Window Workshop allows you to set preferences that are saved as part of your current workspace. The preferences you set take effect for the current Window Workshop and any future Window Workshops you open in the current workspace. However, if any other Window Workshops are already open, these will not be changed.

To set the workshop preferences, choose the File > Workshop Preferences… command. This command opens the Window Workshop Preferences dialog, where you can set any number of preferences.

Figure 7-17 Window Workshop Preferences Dialog



Form Width: 0 Height: 0

Position: System Default (X,Y): 0 0

View:
- [ ] Grid
- [ ] Compound Field Lines
- [✓] Toolbar
- [✓] Widget Palette
- [✓] Color Palette
- [✓] Widget Status
- [✓] Status Line

[ ] Repeat New

The preferences you can set for the Window Workshop fall into the following general categories:

- workshop window size and position
- viewing preferences
- Repeat New preference

The workshop window size and position and viewing preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Window Workshop.

Repeat New Preference

The Repeat New preference specifies whether or not Repeat Mode is used for the Window Workshop.

When Repeat Mode is on, you can add any number of the same widget type to the form by simply clicking repeatedly on the form. In Repeat Mode, the last widget you added to the form continues to be selected on the palette. To add another widget of that type to the form, simply click on the form (or, for resizeable widgets, drag the ghost box). To change the type of widget you wish to add to the form, select a new item on the palette.

When Repeat Mode is off, you must always follow two steps to add a widget to the menu:

1. Select the widget type from the palette.
2. Place the widget on the form.

You can override the Repeat New preference at any time with the Widget > Repeat New command.

Chapter 8

Working with Widgets

This chapter provides general information about working with widgets in the iPlanet UDS Window Workshop.

In this chapter, you will learn how to:

- set widget properties
- control widget sizing
- use widget states
- provide help for widgets
- create multilingual widgets

The chapter also provides detailed information about the individual widgets and their properties. The information on widgets is organized under the following general topics:

- simple widgets
- list widgets
- compound widgets

About iPlanet UDS Widgets

iPlanet UDS widgets are divided into three groups:

- simple widgets
  (including field widgets, list widgets, and graphic widgets)
- compound widgets
- menu widgets

This chapter describes the simple and compound widgets you use to create the form for your window. For information on the menu widgets you use to create the menu bar for your window, see Chapter 9, "Using the Menu Workshop."

Classes for widgets Widgets are objects that are displayed on windows. Every widget provided by the Window Workshop corresponds to an iPlanet UDS class in the Display library. This is true for all the widgets in a window, including simple, compound, and menu widgets.

When you add a widget to a form or menu, you are creating an attribute in the associated UserWindow subclass. For example, when you add a radio list to your form, you are creating an attribute of the RadioList class. All the widgets on a window's form and menu are attributes of the window's class.

Widgets that display data For widgets that display data, such as data fields, toggles, and menu lists, iPlanet UDS creates a second attribute when you add the widget to the user window. This attribute contains or points to the data that the widget displays. The data type of the data attribute is appropriate for data to be displayed, such as string for a data field and boolean for a toggle field. For widgets that display data, you specify the data type of the data attribute in the widget's properties dialog. You can then access and set its value from your TOOL code by referencing the data attribute.

Mapping attributes to widgets Rather than having iPlanet UDS create a second attribute to contain the data for a widget, you can map a widget to an existing attribute in the class. Simply use the existing attribute's name when you specify the widget name; sharing the same name maps the two attributes to each other. For example, if you want your text field to display the artist's name for a painting, you name your text field widget "Artist" because you already have a TextData attribute called "Artist" that stores the artist's name. iPlanet UDS then displays the current value of the Artist attribute in the Artist text field widget.

Referencing widgets When you reference widget attributes in your TOOL code, you distinguish between the visual representation of a widget and the data it displays. You do this by enclosing the widget name in brackets to refer to the visual widget alone.

For example, suppose your form uses a data field widget named "Data1," and that you need to write a method to change the alignment of the characters in the field, and to update the value in the field. In this case, when you change the character alignment, you refer to the visual representation of the data field using the bracket syntax, and when you update the value in the field, you refer to its data attribute without the bracket syntax.

The following TOOL code fragment demonstrates this principle. The fragment defines an event loop to react to clicks on two buttons. The first button, <alignleft_button>, sets the alignment of the data field, and the other button, <update_button>, updates the data itself.

```
event loop
  when <alignleft_button>.click do
    --Refer to the DataField widget
    <data1>.alignment=fa_left
  when <update_button>.click do
    --Refer to the data (the current value of the field)
    data1 = data1 +1;
end event;
```

Visual widgets Some widgets, such as graphic widgets and push buttons, do not represent data. They are visual widgets only. When you refer to visual widgets in your TOOL code, you always use the widget name with brackets.

After creating the widgets, you can manipulate them in your code by setting their attributes or invoking methods on them. For example, you might want to change the background color of a panel when a user clicks on it. You do this by setting the panel's FillColor attribute.

For information on how to set attributes and invoke methods on widgets dynamically, see the *TOOL Reference Guide*. For information on the attributes and methods of individual widget classes, see the Display Library online Help.

The remainder of this section provides general information about working with widgets. This is followed by detailed information on each of the widgets, including descriptions of the properties you can set on each widget's properties dialog.

Naming Widgets

Every widget has an optional name. If you plan to manipulate a widget from your TOOL code, or reference the data it displays, you should name the widget. To name the widget, enter the name into the Attribute Name field on the widget's properties dialog.

Distinguishing a widget from its data The widget name serves two functions. First, the name identifies the attribute that points to the widget. To manipulate the widget from your TOOL code, you must use the widget name enclosed in brackets.

Second, for widgets that display data, such as data fields, the name identifies the attribute that contains the data. For example, if there is a radio list widget called <biglist>, the attribute called “biglist” contains the current value of the radio list. To manipulate the data from your TOOL code, you must use the widget name without brackets.

For more information on referring to widgets and their data in TOOL code, see the *TOOL Reference Guide*.

About Widgets and Data Types

Each widget that displays data has a default data type. This is the data type of the attribute that contains the data that is displayed by the widget. For example, the default data type for a text field is TextData, which stores character data.

Mapped type For most widgets, you have the option of choosing a different data type for a widget. This determines the type of data that the widget can display and/or accept as input from the end user. To change the data type of the data attribute associated with a widget, you use the Mapped Type field on the widget’s properties dialog. If the attribute already exists in the user window class, changing the data type for the attribute on the widget’s properties dialog will change the existing attribute’s data type.

The following list shows the mapped data types available for each of the widgets. The data types in lowercase are simple data types and the types in initial caps are iPlanet UDS classes.

For This Widget	You Can Use These Data Types
Text field	string, TextData, TextNullable
Data field	string, TextData, integer, double, float, IntegerData, long, DateTimeData, IntervalData, DecimalData, DecimalNullable, DoubleData, DoubleNullable, IntegerNullable, DateTimeNullable, IntervalNullable
Toggle field	boolean, BooleanData
Radio list	string, TextData, TextNullable, integer, IntegerData, IntegerNullable
Scroll list	string, TextData, TextNullable, integer, IntegerData, IntegerNullable
Drop list	integer, string, IntegerData, TextData
Fillin field	string, TextData
List view field	DisplayNode class or subclass

For This Widget	You Can Use These Data Types
Picture field	ImageData, ImageNullable
Palette list	string, TextData, TextNullable, integer, IntegerData, IntegerNullable
Outline field	DisplayNode subclass
Scrollbar	integer, IntegerData
Tree view field	DisplayNode class
Menu toggle	boolean, BooleanData, integer, IntegerData
Array field	Array, LargeArray

About Widget States

A widget is always in a particular state. The state of the widget determines how the user can interact with it. For example, when a data field is set to the Update state, the end user can type data into the field. When it is set to the Moveable state, the user can drag it from one part of the window to another. When it is set to the Invisible state, the field is not visible and the user cannot interact with it at all.

States and events The state of the widget also determines which events it posts in response to user actions. For example, when a data field is set to the Update state, it can post an AfterValueChange event. When it is set to Moveable state, it can post the AfterMove event but cannot post the AfterValueChange event. When it is set to the Invisible state, it cannot post any events.

The widget states allow you to control how the end user interacts with the window. For example, for a data entry application, you would want the widget states to allow updating. However, for a drawing application, you would want the widget states to allow moving and resizing.

The following table lists and defines widget states:

	In This State	A Widget…
Operational States	Update	Is visible and operational. For widgets that display data, the data is updateable.
	View	Is visible and operational, but not updateable.

	In This State	A Widget…
Passive States	Disable	Is visible, but dimmed, ignoring mouse clicks and selections.
	Invisible	Is invisible, ignoring mouse clicks and selections.
	Inactive	Is fully visible, but ignores mouse clicks and selections.
Selectable States	Select Only	Is visible and selectable.
	Stretch	Is selectable, and stretchable in any direction from an anchored point.
	Horz Stretch	Is selectable, and stretchable along a horizontal path only.
	Vert Stretch	Is selectable, and stretchable along a vertical path only.
	Move	Is selectable, and moveable in any direction.
	Horz Move	Is selectable, and moveable along a horizontal path only.
	Vert Move	Is selectable, and moveable along a vertical path only.
	Expand	Is selectable, moveable in any direction, stretchable in any direction.
	Route	Is itself selectable, its points selectable and moveable. (Applies to lines and polylines only).
	Move and Route	Is itself selectable and moveable, its points selectable and moveable. (Applies to lines and polylines only).
	Drag	Is selectable, moveable, droppable.
	Mark Rectangle	Shows a ghost box on click and drag.
	Mark Line	Shows a ghost line on click and drag.
	Mark Polyline	Shows a ghost line from point to point in a polyline during a click/drag/click sequence. Sequence ends on a double-click.
	Mark Point	Shows a ghost point on click and drag.

Using Widget States with Window Usages

As described under "Usage Property" on page 370, a window's usage determines which state is in effect for a widget. When you create a widget in the Window Workshop, iPlanet UDS uses a default state for the widget for each window usage. For example, in the Update usage, the default state for a data field is the Update state. In most cases, these default widget states are the best settings and you should not change them. However, if you are writing an application that runs in different modes or if you want one of the widgets to be in an unusual state (for example, setting a widget to View state while the rest of the widgets are in the Update state), you can change the widget state for a particular usage.

Changing a Widget's State Dynamically

You can also change a widget's state dynamically from your TOOL code. The State attribute of the FieldWidget class lets you specify the current state of the widget, regardless of what usage is in effect for the window. See the Display Library online Help for information on this technique.

About Tabbing

The end user of your application can move between fields on the form in two ways:

- move the cursor to the field and click on the mouse to activate the field
- press the Tab key to move from the current field to the next field in the tabbing sequence

The tabbing sequence moves from left to right, from the top of the form to the bottom. Your window system determines which types of widgets are included in the tabbing sequence and which are not. For example, a data field is included in the tabbing sequence on Windows. See your window system documentation for information on which widgets are included in the tabbing sequence. If an individual field on your form is included in the tabbing sequence and you wish to exclude it, you can choose the Widget > Skip on Tab command. The Skip on Tab menu toggle excludes the currently selected field from the tabbing sequence so that the Tab key will skip over the field.

About Input Focus

Some window systems provide mouseless support. Mouseless support means that the end user can use the keyboard (that is, the tab key or arrow keys) to move between the fields on the form.

Using the keyboard to move between fields means moving the input focus from one field to another. The field that has the input focus is the one field on the form that will currently accept the input from the keyboard. Your window system indicates which field has the input focus by displaying the text insertion cursor or by using another indicator. In addition, iPlanet UDS posts BeforeFocusLoss and AfterFocusGain events when the end user moves the input focus from one field to another.

Only certain fields can have the input focus—this differs among window systems. For example, all window systems give the input focus to text fields. However, only some window systems give the input focus to push buttons. If you are creating a portable application that uses input focus events, you need to be aware of the differences between the window systems. See the Display Library online Help for more information on input focus.

About Sizing

When you add a sizeable field to a form, you determine its initial size when you drop it on the form. After this, there are several options for adjusting its size relative to the other fields on the form.

If you are planning to deploy your application on more than one window system, you should use the iPlanet UDS geometry management features to ensure that the fields on your form will be correctly aligned on all the window systems. For detailed information about using geometry management features for aligning fields, see the *iPlanet UDS Programming Guide*.

However, even if you do not plan to deploy your application on multiple window systems, you may still wish to use the iPlanet UDS basic sizing features to arrange the fields on your form.

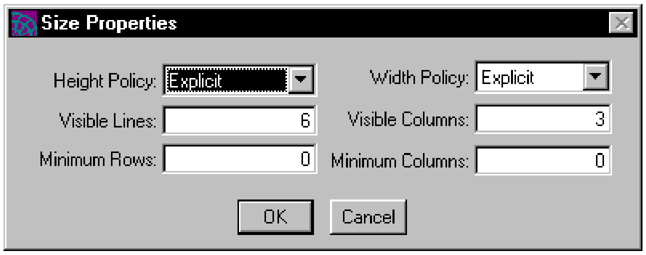
Size policies for widgets All iPlanet UDS fields have two properties that specify how the field size is determined: the height policy and width policy. The size policies for fields allow you to control whether the field size is static, whether iPlanet UDS adjusts the field size to accommodate its content, or whether the field size is determined by the grid that contains it. See “Size Policies” below for further information about height and width policies.

Size partnerships for widgets iPlanet UDS allows you to link any number of resizeable fields together in a height or width partnership. When fields are in a size partnership, their height or width is determined by the largest minimum height or width of the fields in the partnership. See "Size Partnerships" on page 442 for further information about height and width partnerships.

Size Policies

The Widget > Size Policy command lets you set the height and width policy for any widget. The size policies for a field allow you to control whether the field size is static, whether iPlanet UDS adjusts the field size to accommodate its content, or whether the field size is determined by the grid that contains it. For resizeable widgets, you can also specify the minimum height and width for the field. For grid fields, you can set column and row justify weights. Figure 8-1 shows the Size Properties dialog for a text field:

Figure 8-1 Size Properties Dialog



Size Properties

Height Policy: Explicit
Width Policy: Explicit
Visible Lines: 6
Visible Columns: 3
Minimum Rows: 0
Minimum Columns: 0

(Note that the Size Properties dialog for the widget is also available through the Size Property button on the widget's properties dialog.)

Height and Width Policy properties The Height Policy property specifies the policy iPlanet UDS uses for setting the field widget's height. The Width Policy property specifies the policy iPlanet UDS uses for setting the field widget's width. The policy you choose depends on whether the field is included within a grid field or a size partnership. The default height and width policies for a field depend on the type of field.

The values for Height and Width Policy properties are:

Size Policy Setting	Definition
Natural	iPlanet UDS automatically adjusts the height or width of the field to accommodate the field's content. When the height of a field is Natural, you cannot resize it on the screen (resize handles will not be displayed).
Explicit	You can specify an explicit height and width for the field by resizing it in the Window Workshop.
Matrix Partner	For a column in a grid field, the width of the column is determined by the width of the column with the largest minimum width in the set of column partners. For a row in a grid field, the height of the row is determined by the height of the row with the largest minimum height in the set of row partners. Row and column partnerships are determined by the Arrange > GridFields into Column Partnership and GridFields into Row Partnership commands. See the *iPlanet UDS Programming Guide* for further information.
Parent	The height or width of the field is determined by the height or width of its parent. If the field is in a grid field, this setting uses the grid field cell size to determine the field's height or width. When the grid field changes size, the height or width of the child field will automatically be resized to fit the cell. When the height or width of a field is Parent, you cannot resize the field on the screen. Note that you should be careful not to set the height or width policy of all fields in a grid to Parent if all of the fields have a very small minimum size. See the *iPlanet UDS Programming Guide* for further information.
Size Partner	The height or width of the field is determined by the height or width of the field with the largest minimum height or width in the set of partners for the current field. When you include a field in a size partnership, iPlanet UDS automatically sets the size policy to Size Partner.

Minimum height and width If a field has a height or width policy of Parent and the field is resizeable, you can set a minimum height and width for the field. iPlanet UDS uses these minimum dimensions when calculating the size of the field. Setting a minimum height and width of a field ensures that iPlanet UDS will never resize the field below these dimensions when the grid field is resized. In addition, the grid field size will always be large enough to display the entire field at its specified minimum size.

For text fields, text edit fields, and data fields, you specify the minimum dimensions in rows and columns. For all other resizeable fields, you specify the minimum dimensions in mils.

Grid field justify weights The Size Policies dialog for grid fields lets you set the justify weights for the columns and rows in the grid fields. See "Setting Grid Field Properties" on page 519 for information on justify weights.

Image size policies The Size Policies dialog for picture fields and picture graphics allows you to ensure that the entire image is always displayed by scaling the image to fit the field. Or, you can ensure that the field does not change size by clipping the image to fit into the field.

The image size policy properties are the following:

Value	Definition
Natural	The image size never changes (the true image size is retained). If the field is too small for the image, the image is cropped. If the field is larger than the image, the setting of the Image Gravity property is used to position the image within the field.
Field	The image is scaled to fit into the field, possibly distorting the image.
Field Height	The image height is scaled to fit the field height. The width will be adjusted to preserve the aspect ratio.
Field Width	The image width is scaled to fit the field width. The height will be adjusted to preserve the aspect ratio.

Image gravity The Size Properties dialog for picture fields and picture graphics also contains the Image Gravity property, which allows you to determine the alignment of the image within the field. The options are: top left, top center, top right, middle left, center, middle right, and bottom left, bottom center, and bottom right.

Size Partnerships

In the Window Workshop, the Arrange > Fields Into Height Partnership and Fields Into Width Partnership commands let you select a group of fields to include in a height or width partnership.

The fields in the partnership can be simple or compound, resizeable or fixed size. If you include fixed-size widgets in a size partnership, the fixed-size field itself is not resized, but its minimum size affects the resizeable widgets. For example, if a radio list, which has a fixed size, is in a width partnership with a push button, which is resizeable, the radio list will never be resized but the push button will be sized in relation to the radio field. If all are fixed, the partnership is legal, but meaningless. All compound fields are sizeable, and can be included in size partnerships. See "Creating Widgets" on page 401 for a list of the sizeable and fixed-sized simple fields.

Height partnership When two or more fields are in a height partnership, iPlanet UDS uses the height of the field with the largest minimum height in the partnership to determine the height for all the fields in the partnership. You specify the minimum height by using the Widget > Size Policy command.

When you add a field to a height partnership, its Height Policy property is automatically set to Size Partner. The height partnership is completely independent of the width partnership.

Width partnership When two or more fields are in a width partnership, iPlanet UDS uses the width of the field with the largest minimum width in the partnership to determine the width for all the fields in the partnership. You specify the minimum width for a field by using the Widget > Size Policy command.

When you add a field to a width partnership, its Width Policy property is automatically set to Size Partner. The width partnership is completely independent of the height partnership.

The Window Workshop provides the following commands on the Arrange menu for setting size partnerships:

Command	**Description**
Fields Into Height Partnership	All selected fields are grouped into a height partnership.
Fields Into Width Partnership	All selected fields are grouped into a width partnership.
Remove Field From Height Partnership	All selected fields are removed from the height partnership.

Command	Description
Remove Field From Width Partnership	All selected fields are removed from the width partnership.

About Help for Widgets

There three kinds of help you can provide for an individual widget on a window:

- context-sensitive help
- float-over help
- status-line help

The following sections provide background information about these kinds of help. See the *iPlanet UDS Programming Guide* for complete information on implementing help.

About Context-Sensitive Help

Context-sensitive help is a help topic that is displayed in response to the Help key. When the end user presses the Help key, iPlanet UDS checks the field that has the input focus to see if a value was provided for its help topic (stored in its HelpTopic attribute). If there is a help topic associated with the field that has the input focus, iPlanet UDS displays the field's help topic. If there is no help topic associated the field that has the input focus, iPlanet UDS checks the field's parent, the parent's parent, and so on, all the way up the containment hierarchy to the window, to find a help topic. If there is no help topic associated with the window, iPlanet UDS displays the Contents page for the Help document.

Only certain fields can have the input focus—this differs among window systems. For example, all window systems give the input focus to text fields. However, only some window systems give the input focus to push buttons. If you are creating context-sensitive help that uses input focus events, you need to be aware of the differences between the window systems. See "About Input Focus" on page 438 and the Display Library online Help for more information on input focus.

You provide the context-sensitive help for an individual field by using the Widget > Help Text command. On the Help Text dialog for the widget, set the widget's Help Topic property to the topic ID or key word for its help topic in the default help file. (Note that the Help Text dialog for the widget is also available through the Help Text button on the widget's properties dialog.)

See the *iPlanet UDS Programming Guide* for complete information about creating the help file that contains the help messages, linking the help file to the window, and associating specific topics in the help file with individual widgets.

About Float-Over Help

Float-over help is a help message that is displayed next to the field whenever the mouse pauses over the field. To provide float-over help for the fields on your window, you must ensure that float-over help is turned on for the window system, and you must provide the float-over help text for the individual fields on the window.

IsFloatOverEnabled attribute The IsFloatOverEnabled attribute of the WindowSystem class specifies whether or not the float-over help for the window system is displayed. The default value of the IsFloatOverEnabled attribute is determined by the FORTE_ISFLOATOVERENABLED environment variable, whose default value is TRUE. You should use the IsFloatOverEnabled attribute in your TOOL code to ensure that float-over help is turned on when appropriate, and to give the end user the option of turning it off and on. See the Display Library online Help for information on the IsFloatOverEnabled attribute.

➤ **To provide float-over help for a field**

1. Select the field and choose the Widget > Help Text command.
2. In the Float-over Text field on the Help Text dialog, enter the float-over help message.
3. In your TOOL code, use the WindowSystem.IsFloatOverEnabled attribute to ensure that float-over help is enabled for the application and to provide the end user with the ability to turn the float-over help on and off.

Note that the float-over text for an individual widget is stored in the FloatOverText attribute defined on the FieldWidget class. See the Display Library online Help for information.

For complete information on implementing help for your window, see the *iPlanet UDS Programming Guide*.

Float-over help for palette lists To provide float-over help for individual regions (or icons) in a palette, you must use the Float-Over Help property on the palette's properties dialog for each list item in the palette.

The ShowRegionFloatOver attribute of the PaletteList class determines whether the float-over help for the individual regions (or list items) in the palette list are actually displayed. By default, this attribute is set to TRUE. When you set the ShowRegionFloatOver attribute to FALSE, the float-over messages for individual regions in the palette are not displayed.

See the *iPlanet UDS Programming Guide* for further information about providing float-over help for palette lists.

See "Creating a Palette List" on page 489 for information about setting these properties for palettes.

About Status-Line Help

Status-line help is a help message that is displayed in the window's status line when the mouse pauses over the field.

Creating a status line To provide status-line help for a window, you must add a status line field to the window and map this field to a TextData attribute. Then, you must map the TextData attribute associated with the status line field to the StatusText attribute of the Window object. Finally, you must provide the status-line help text for individual fields by using the Widget > Help Text command. See the *iPlanet UDS Programming Guide* for complete information on implementing status-line help for your window.

WindowSystem.StatusText attribute If you have set up the status line field correctly, each time the mouse moves onto a new field, iPlanet UDS automatically sets the value of the WindowSystem.StatusText attribute to the status-line help text value that was specified for the current field. The window is automatically refreshed, and the new value of the StatusText attribute is displayed in your window's status line field. If the Window.StatusText is NIL, there will be no status-line help for the window. See the Display Library online Help for information about the StatusText attribute on the WindowSystem class.

Status-line help for palette lists To provide status-line help for individual regions (or icons) in the palette, you must use the Status Line property on the Palette List's properties dialog for each picture graphic or picture button in the palette. See "Creating a Palette List" on page 489 for information about setting the Status Line property for the palette regions.

About Internationalizing Widgets

The section "About Internationalizing Windows" on page 382 described how you can use the Window Workshop to create a multilingual window. All widgets that display static text have Message Number and Set Number properties in the Window Workshop that you can use to specify a message number and optional set number for the individual widget. (These properties are not available on widgets that display text only at runtime, such as a data field.)

Message set used for widgets By default, all messages for a window are loaded from the same message set, which you specify using the Default Set Number property on the Window Properties dialog (see "Setting Window Properties" on page 396). If you do not specify a value for an individual widget's set number, iPlanet UDS will use the default set number for the window. If you do specify a set number for the individual widget, that set will be used.

IME mode For widgets that allow data entry, including data field, text field, text edit field, and fillin field, you can turn on the double-byte manager (IME mode) for the field to allow entry of multi-byte characters, such as Kanji. By default, this property is off. To request IME Mode for an individual widget, simple turn on its IME Mode property.

Multilingual Help You can also create multilingual help for individual widgets by using the Message Number and Set Number properties on the widget's Help Properties dialog. Multilingual help. The message number for an individual widget's help text should point to a message that contains both the float-over and status-line messages in the following format:

*float_over_help***n** *status_line_help*

The following sections provide information about using the Window Workshop to specify the message and set numbers for the text displayed by the widget as well as for its help text. See the *iPlanet UDS Programming Guide* for complete information on creating multilingual windows.

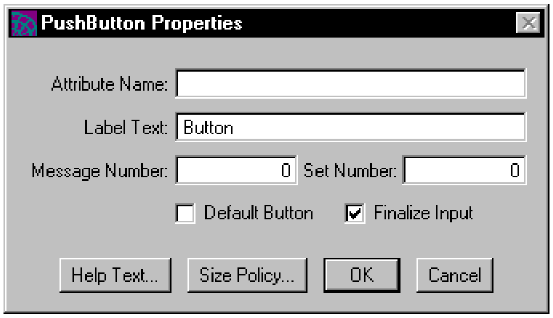
Specifying Message and Set Numbers for Widgets

For each widget that has a single text value that you can internationalize, there is a Message Number and Set Number property. For lists (radio list, drop list, scroll list, fillin field, menu list), there is a Text Value Set Number property that specifies the set number for all the list elements, as well as Message Number properties for individual list elements.

For outline fields and list view fields, there is a Column Title Set Number that specifies the set number for all the column titles.

➤ **To specify message and set numbers in the Window Workshop**

1. Double-click the widget to display its properties dialog.



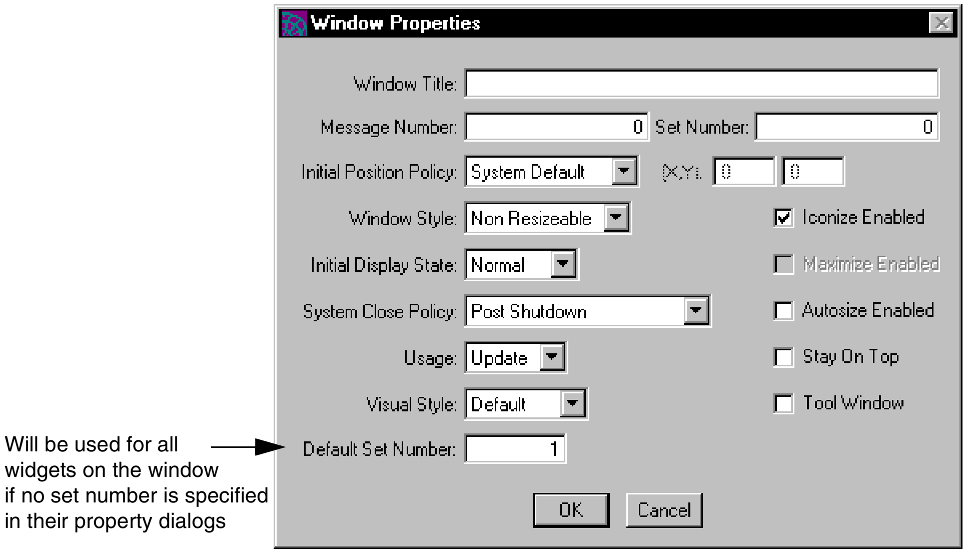
PushButton Properties

Attribute Name:
Label Text: Button
Message Number: 0 Set Number: 0
☐ Default Button ☑ Finalize Input

2. Enter a message number in the Message Number field and a set number, if necessary, in the Set Number field.

   Note that you can use the set number for the window as the default set number for every widget on the window. In this case, you do not need to enter a set number in the widget's properties dialog, but you must enter a set number in the Window Properties dialog. However, if you do enter a set number in the widget's properties dialog, it will override the set number in the Window Properties dialog.

   The Window Properties dialog is shown in the following figure:



Window Properties

Window Title:
Message Number: 0 Set Number: 0
Initial Position Policy: System Default (X,Y): 0 0
Window Style: Non Resizeable
Initial Display State: Normal
System Close Policy: Post Shutdown
Usage: Update
Visual Style: Default
Default Set Number: 1
☑ Iconize Enabled
☐ Maximize Enabled
☐ Autosize Enabled
☐ Stay On Top
☐ Tool Window

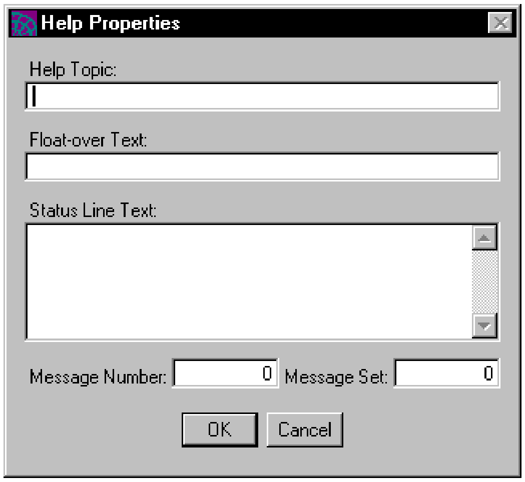
Will be used for all widgets on the window if no set number is specified in their property dialogs
OK
Cancel

3. In the message catalog, enter the message number and the appropriate text according to the syntax described in the *iPlanet UDS Programming Guide*.

Multilingual Help for Widgets

All widgets have a Help Properties dialog in the Window Workshop, shown in Figure 8-2, where you can enter float-over text and status-line help text. Each dialog has a Message Number and Set Number property in which you specify the message number and, optionally, the set number. If you do not specify a set number, the help text will use the set number specified for the widget's window.

Figure 8-2 Help Properties Dialog



NOTE There is only one message and (optional) set number for the two types of help—float-over and status line. When you enter the help text in the message catalog, you must use the syntax described in the Display Library online Help, which is the following:

---

float_over_help **\n** *status_line_help*

This syntax places both help strings on one line in the catalog, and thus only one message number is required.

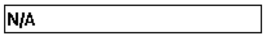
Working with Simple Widgets

Simple widgets are basic controls that allow the end user to invoke commands, make selections, and enter and display information. iPlanet UDS provides the following simple widgets:

Simple Widget	Definition
Data Field	Displays a single line of data—text or numbers. A data field can enforce a specified format on data entry, or display existing data according to a specified format.
Text Field	Displays text in a scrollable field, with optional word wrap.
Text Edit Field	Provides a simple text editor for text, without word wrap.
Push Button	A button that a user clicks to enact a command. The button features a text label describing its function.
Picture Button	A button that displays an image that the end user can click to enact a command.
Toggle Field	A labeled check box. The check box has two possible values—on and off (checked and unchecked).
Picture Field	A frame where you can display image data that changes at runtime.
Scrollbar	An independent scrollbar that you can program to work as a scrollbar or as a slider that lets the end user specify a numeric value within a range by dragging or clicking on the bar.
List View Field	Displays a set of items, each consisting of an icon with a label, from which the end user can make selections.
Tree View Field	Displays two-column hierarchical information in an indented outline, providing controls that let the end user expand and collapse the outline.
Outline Field	Provides a browser for a hierarchy of data, such as a file directory structure, or a simple list with icons.
OLE Field	Displays OLE linked or embedded objects. When your iPlanet UDS application is running, the user can work directly with the linked or embedded object. The OLE field is only available for Windows platforms.
ActiveX Field	Displays an ActiveX control in your iPlanet UDS window. The ActiveX field is only available for Windows platforms.

Simple Widget	Definition
List Widgets	List widgets are a special form of simple widget that allow you to provide the end user with a set of choices from which she can make one, and sometimes more than one, selection. For information on list widgets, see "Working with List Widgets" on page 480.
Graphic Widgets	Graphic widgets are a special form of simple widget. You use graphic widgets as form decorations and organizers. Graphic widgets include lines, rectangles, and picture graphics. For information on graphic widgets, see "Working with Graphic Widgets" on page 493.

Creating a Data Field



A data field displays a single line of data of a specific data type, such as a date, money value, or ID number. Use the data field to display information to the end user or to provide a data entry field where the end user can type in or modify information.

Specifying Data for a Data Field

To restrict the content of a data field, you specify a data type. This allows you to create, for example, an ID entry field where only integers are allowed. You can also specify a format to control the appearance of the data.

The data types you can choose from include both simple data types and class data types. The simple data types store the data, but do not provide the functions available in the class data types. The classes store the data as objects, and provide a wide range of methods for manipulating the data. To create a data field for date or time data, you can use only class data types (DateTimeData and IntervalData).

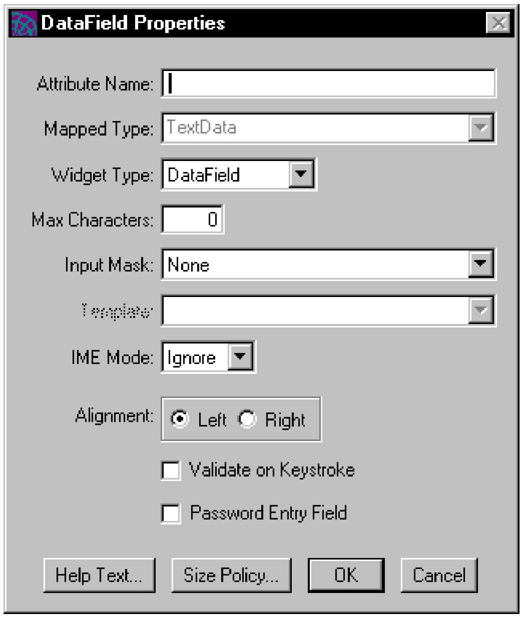
Data formatting templates Depending on the type of data you choose for the field, you can apply a specific formatting template to restrict data entry into the field to a certain format. The Data Field Properties dialog offers a wide variety of sample formats for each data type. You can use these formatting templates for the field, alter them to suit your needs, or design your own templates from scratch. For more information on the codes used in data field formatting templates, see the DataField class in the Display Library online Help.

Setting Data Field Properties

The Data Field Properties dialog, shown in Figure 8-3, allows you to set the following properties for a data field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the data field.
Mapped Type	To set the data type for the text in the data field. Choices are: DateTimeData, TextData, IntegerData, DoubleData, DecimalData, boolean, integer, string, double, IntegerNullable, TextNullable, DoubleNullable, DateTimeNullable, and DecimalNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Max Characters	To set the maximum number of characters the data field allows. Set to zero to allow maximum number for the particular host window system.
Input Mask	To specify the masking type that restricts the data the end user can type into the field. You can use this instead of a complex template to provide for simple data input.
Template	A predefined or custom formatting template for the data field. The list of templates varies according to the mapped type set for the data field. A data field with a mapped type setting of DateTimeData, for example, would show a set of sample date templates. For more information on formatting templates, see the DataField class in Display Library online Help.
IME Mode	To specify whether the double-byte input manager is on or off when entering the field. The default, IGNORE, leaves the current setting, whether on or off, unaltered. The double-byte input manager allows entry of multi-byte characters, such as Kanji.
Alignment	To set the alignment of the data in the data field, Left or Right.
Validate on Keystroke	To specify whether or not the data stored in the variable associated with the widget is updated after every keystroke by the end user. If this toggle is off, the data stored in the variable associated with the widget is updated after the user exits the field.
Password Entry Field	To enable/disable echoing of typed characters in the data field.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-3 Data Field Properties Dialog


DataField Properties

Attribute Name:

Mapped Type: TextData

Widget Type: DataField

Max Characters: 0

Input Mask: None

Template:

IME Mode: Ignore

Alignment: [✓ Left] Right

☐ Validate on Keystroke

☐ Password Entry Field

Creating a Text Field

A text field displays multiple lines of text, which the end user can scroll. Use text fields to display text data to the end user, or to provide a data entry field where the end user can type in or modify text.

A text field always has a vertical scrollbar. Text fields support the text-editing conventions of host window systems, and also offer options for word-wrapping or horizontal scrolling.

A text field defines text in lines and columns. A line of text is a string of characters from left to right. A column is a character in a line. A text field using a monospaced font considers each character a column. A text field using a proportionally spaced font calculates an average character width for the font, and considers that measurement a column.

Normally you set the dimensions of text fields in terms of visible lines and columns, rather than in discrete units such as mils. This way, a text field's display remains consistent across window systems despite font variations, which might otherwise change the number of its visible lines or columns.

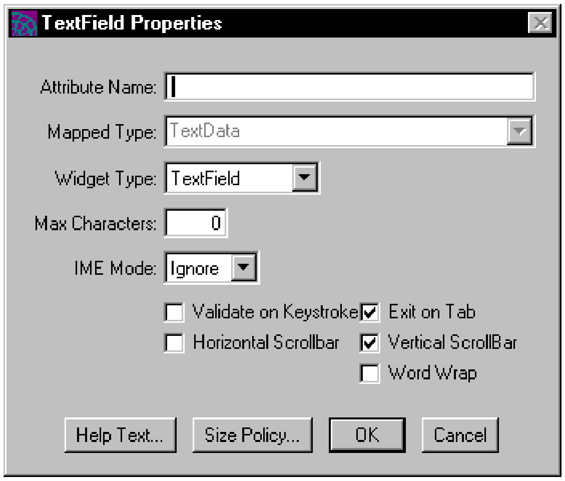
Setting Text Field Properties

The Text Field properties dialog, shown in Figure 8-4, allows you to set the following properties for a text field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the text field.
Mapped Type	To set the data type for the text in the text field. Choices are: TextData, string, and TextNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Max Characters	To set the maximum number of characters the text field allows. Set to zero to allow maximum number for the particular host window system.
IME Mode	To specify whether the double-byte input manager is on or off when entering the field. The default, IGNORE, leaves the current setting, whether on or off, unaltered. The double-byte input manager allows entry of multi-byte characters, such as Kanji.
Validate on Keystroke	To specify whether or not the data stored in the variable associated with the widget is updated after every keystroke by the end user. If this toggle is off, the data stored in the variable associated with the widget is updated after the user exits the field.
Exit on Tab	To specify whether or not the tab character is treated as the field exit character. If this toggle is off, the tab character is treated as data.
Horizontal Scrollbar	To specify whether or not to provide a horizontal scrollbar for the text field. If a text field uses a horizontal scrollbar, it cannot use word-wrapping.
Vertical Scrollbar	To specify whether or not to provide a vertical scrollbar for the text field.
Word Wrap	To specify whether or not to provide word-wrapping for the text field. If a text field uses word-wrapping, it cannot use a horizontal scrollbar.

Use This Property	For This Purpose
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

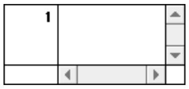
Figure 8-4 Text Field Properties Dialog


TextField Properties

Attribute Name:
Mapped Type: TextData
Widget Type: TextField
Max Characters: 0
IME Mode: Ignore

☐ Validate on Keystroke
☑ Exit on Tab
☐ Horizontal Scrollbar
☑ Vertical ScrollBar
☐ Word Wrap

Creating a Text Edit Field



A text edit field provides a simple text editor. The field displays multiple lines of text, which the end user can scroll horizontally or vertically. It can also provide line numbers and or icons for the lines of text.

The main difference between a text edit field and a text field is that a text edit field:

- has no word wrap
- can display line numbers and icons
- has automatic cut/copy/paste and undo/redo functions

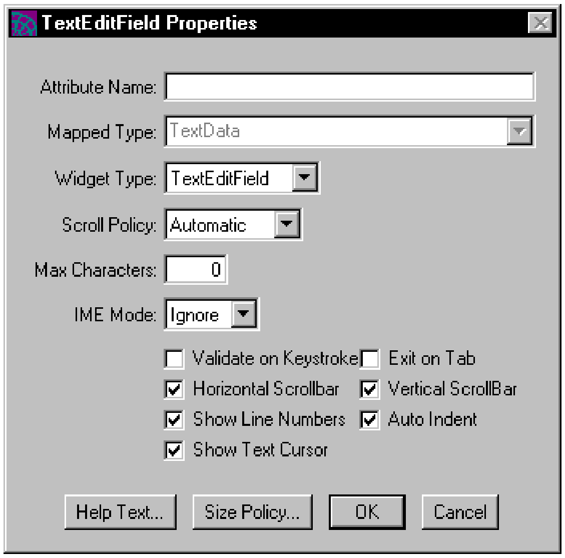
Setting Text Edit Field Properties

The Text Edit Field properties dialog, shown in Figure 8-5, allows you to set the following properties for a text edit field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the text field.
Mapped Type	To set the data type for the text in the text field. Choices are: TextData, string, and TextNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Scroll Policy	To set the scroll behavior for the field when the cursor position is moved. Automatic scrolling (the default) puts the selection in view, Top scrolls the selection to the top, Bottom scrolls the section to the bottom, Middle scrolls the selection to the middle.
Max Characters	To set the maximum number of characters the text field allows. Set to zero to allow maximum number for the particular host window system.
IME Mode	To specify whether the double-byte input manager is on or off when entering the field. The default, IGNORE, leaves the current setting, whether on or off, unaltered. The double-byte input manager allows entry of multi-byte characters, such as Kanji.
Validate on Keystroke	To specify whether or not the data stored in the variable associated with the widget is updated after every keystroke by the end user. If this toggle is off, the data stored in the variable associated with the widget is updated after the user exits the field.
Exit on Tab	To specify whether or not the tab character is treated as the field exit character. If this toggle is off, the tab character is treated as data.
Horizontal Scrollbar	To specify whether or not to provide a horizontal scrollbar for the text field. If a text field uses a horizontal scrollbar, it cannot use word-wrapping.
Vertical Scrollbar	To specify whether or not to provide a vertical scrollbar for the text field.
Show Line Numbers	To specify whether or not the field displays a number for each line of text in the field (like line numbers for code). If this toggle is off, the field does not display line numbers.

Use This Property	For This Purpose
Auto Indent	To specify whether or not each new line the user enters in the text edit field is automatically indented. Every new line starts at the same position of the first character in the line it follows.
Show Text Cursor	To specify whether or not the field displays the input cursor. If this toggle is off, the field does not display the input cursor.
Help Text	To open the Help Text dialog.
Size Policy	To open the Size Policy dialog.

Figure 8-5 Text Edit Field Properties Dialog



TextEditField Properties

Attribute Name:

Mapped Type: TextData

Widget Type: TextEditField

Scroll Policy: Automatic

Max Characters: 0

IME Mode: Ignore

- [ ] Validate on Keystroke
- [ ] Exit on Tab
- [x] Horizontal Scrollbar
- [x] Vertical ScrollBar
- [x] Show Line Numbers
- [x] Auto Indent
- [x] Show Text Cursor

Creating a Picture Field

A picture field provides a picture frame where you can display image data. At runtime, the data attribute associated with the field provides the image that is displayed within the picture field's boundaries.

Image Size policy For picture fields, iPlanet UDS allows you to ensure that the entire image is always displayed by scaling the image to fit the field. Or, you can ensure that the field does not change size by clipping the image to fit into the field. You specify the image size policy on the Size Policy dialog for the picture field. See "Size Policies" on page 439 for information.

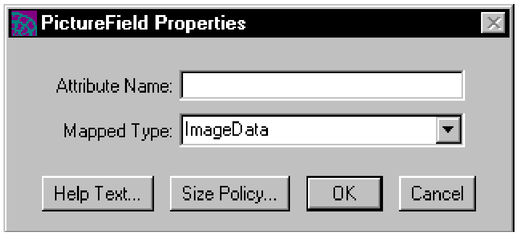
Image Gravity property When a picture field is larger the image it contains, you can set a picture field's Image Gravity property to determine how the picture field aligns the image within the field. A picture graphic can, for example, center an image using an image gravity setting. You specify the image gravity on the Size Policy dialog for the picture field. See "Size Policies" on page 439 for information.

Setting Picture Field Properties

The Picture Field Properties dialog, shown in Figure 8-6, allows you to set the following properties for a picture field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the picture field.
Mapped Type	To set the data type for the picture field elements. Choices are: ImageData and ImageNullable.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the SIze Policy dialog for the field.

Figure 8-6 Picture Field Properties Dialog



Creating a Toggle Field

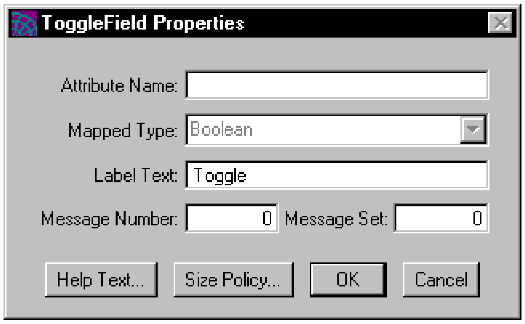
A toggle field displays a labeled check box. The check box has two possible values: on (checked) and off (unchecked). Each time the end user clicks the toggle, the field's value toggles from the selected value to the other. To change a toggle field's label, use the Toggle Field Properties dialog.

Setting Toggle Field Properties

The Toggle field properties dialog, shown in Figure 8-7, allows you to set the following properties for a toggle field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the toggle field.
Mapped Type	To set the data type for the toggle field. Choices are: Boolean and BooleanData.
Label Text	To set the toggle field label text.
Message Number	To specify a message number for the label text. This property is for creating a multilingual window.
Message Set	To specify the message set for the toggle field's message number. If no message set is specified, the default message set for the window is used. This property is for creating a multilingual window.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-7 Toggle Field Properties Dialog



Creating a Push Button

A push button is a labeled button that the end user can click to give a command or instruction. When you place a push button on the form, iPlanet UDS provides a default label. You can change the label in the Push Button Properties dialog.

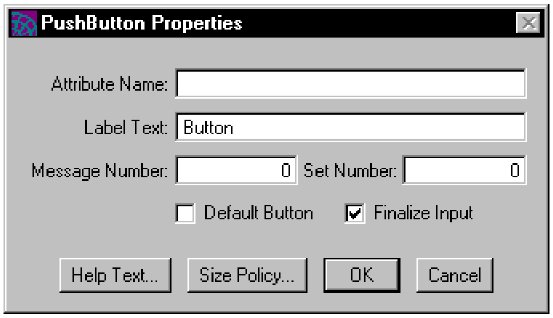
Setting Push Button Properties

The Push Button Properties dialog, shown in Figure 8-8, allows you to set the following properties for a push button:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the push button.
Label Text	To set the text for the button label.
Message Number	To specify a message number for the label text. This property is for creating a multilingual window.
Set Number	To specify the message set for the push button’s message number. If no message set is specified, the default message set for the window is used. This property is for creating a multilingual window.
Default Button	To set the push button to be the default button for its form.
Finalize Input	To specify whether or not a button click initiates input finalization for the character field that has the input focus at the time of the click. For more information about input finalization and form validation, see the Display Library online Help.

Use This Property	For This Purpose
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-8 Push Button Properties Dialog


PushButton Properties

Attribute Name:

Label Text: Button

Message Number: 0 Set Number: 0

☐ Default Button ✓ Finalize Input

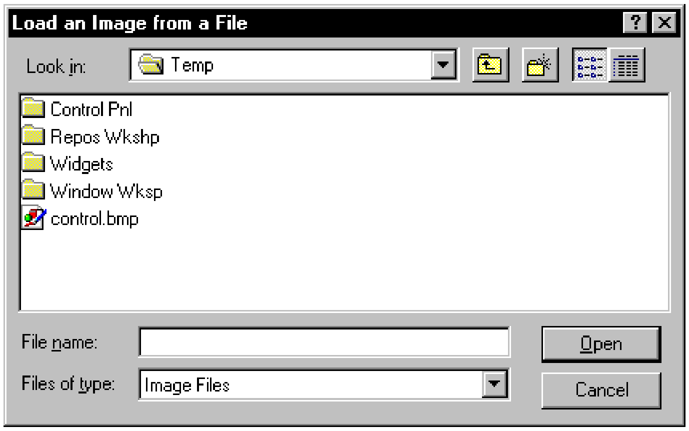
Creating a Picture Button

A picture button is a button that displays an image that the end user can click to give a command.

➤ To create a picture button

1. Click the picture button icon.
2. Click on the form where you wish to place the button.

   The Window Workshop opens the Load an Image from a File dialog, where you can choose an image file to use for the button.



Look in: Temp

Control Pnl
Repos Wkshp
Widgets
Window Wksp
control.bmp

File name:
Files of type: Image Files

3. In the file selection dialog, choose the image file and click the OK button to create the picture button.

Drag and drop for Windows In Microsoft Windows, you can replace the image on a picture button by dragging an image file from your File Manager onto the form.

➤ To replace an existing image

1. Select an image file from the file manager.
2. Drag the file on top of the picture button whose image you want to replace. The picture button now contains the image file you dropped.

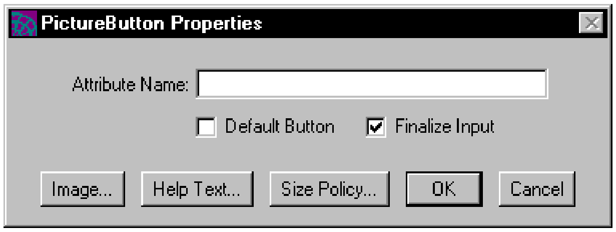
The picture button assumes a size just large enough to accommodate the image; you resize a picture button by choosing a larger or smaller image in the Picture Button Properties dialog.

Setting Picture Button Properties

The Picture Button Properties dialog, shown in Figure 8-9, allows you to set the following properties for a picture button:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the picture button.
Default Button	To set the picture button to be the default button for its form.
Finalize Input	To specify whether or not a button click initiates input finalization for the character field that has the input focus at the time of the click. For more information about input finalization and form validation, see the Display Library online Help.
Image…	To change the picture button image. The Image button opens a file directory dialog from which you can select a new image.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-9 Picture Button Properties Dialog



Default Button [ ] Finalize Input [✓]

Creating a Scrollbar

An independent scrollbar looks just like the scrollbars used in other iPlanet UDS fields, such as text fields and array fields. It generally includes the bar itself, arrows at either end of the bar, and a scroll box within the bar. Clicks in the bar itself cause incremental scrolling, clicks in the arrows cause line-by-line scrolling, and a click/drag sequence of the scroll box causes proportional scrolling. In proportional scrolling, the scroll box assumes a new position, and the scroll itself occurs when the mouse button is released.

Unlike scrollbars integrated into other widgets, an independent scrollbar does not automatically scroll anything else. You must write TOOL code that reacts to user manipulation of the scrollbar by waiting for the AfterValueChange event on the scrollbar and responding appropriately when the event is posted.

You can use scrollbars as widgets for representing and setting ranges, as in volume controls for playing sound, or sliding scales for setting color display intensity ranges.

In a volume control, for example, a scrollbar might represent volume at levels extending from one to eleven, in increments of one. To change the volume, a user would click and drag the scroll box from one value to another. The scroll box takes a new position, and the value changes only when the user releases the mouse button.

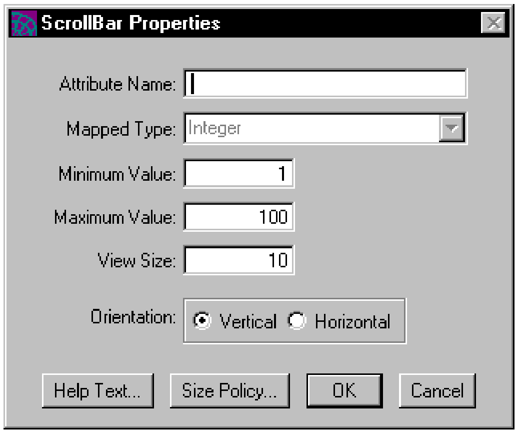
Using a viewport If you simply wish to add scrolling to a single field, we recommend placing the field in a viewport, which provides horizontal and vertical scrolling for the field it contains. For example, to display an image that exceeds the size you wish to use to display it, you can enclose it in a viewport. This way, you maintain complete access to the image for users, but can use a smaller space to display it. For more information about viewports, see “Creating a Viewport” on page 524.

Setting Scrollbar Properties

The Scrollbar Properties dialog, shown in Figure 8-10, allows you to set the following properties for a scrollbar:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the scrollbar.
Mapped Type	To set the data type for the scrollbar. Choices are: integer, IntegerData
Minimum Value	To set the scrollbar's minimum value.
Maximum Value	To set the scrollbar's maximum value.
View Size	To set the increment by which the scroll bar scrolls in response to clicks in the bar.
Orientation	To set the scrollbar's orientation: vertical or horizontal.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-10 Scrollbar Properties Dialog



ScrollBar Properties

Attribute Name:

Mapped Type: Integer

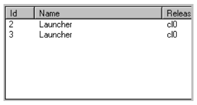
Minimum Value: 1

Maximum Value: 100

View Size: 10

Orientation: [✓ Vertical] Horizontal

Creating a List View Field

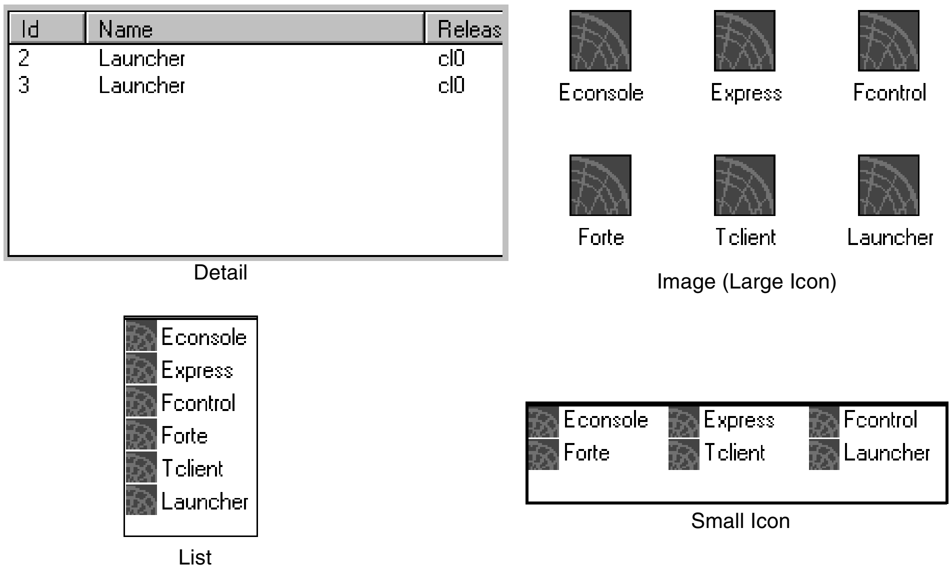


Id	Name	Releas
2	Launcher	cl0
3	Launcher	cl0

A list view field displays a set of items, each consisting of an icon with a label, from which the end user can make selections. There are four styles for list view fields: image (large icon), small icon, list, and detail. On Windows 95/NT, iPlanet UDS uses the native list view control. On all other window systems, iPlanet UDS creates a custom widget.

The following figure illustrates the list view styles:

Figure 8-11 List View Field Styles



Id	Name	Releas
2	Launcher	cl0
3	Launcher	cl0

Note that the small icon, image (large icon), and list styles are available only on Windows 95/NT.

Very often, user interface designers use list view and tree view fields together. A tree view field on the left allows the user to search through a hierarchy for the item he wants, and a list view field on the right provides the detail information for the selected item in the tree view field.

The ListView class, which defines the list view widget, is a subclass of the OutlineField class, and it provides a simplified version of the outline field. The advantages of using a list view field over an outline field are:

- iPlanet UDS provides the look and feel of a list view control
- it is easier to define the data

  The DisplayNode class provides attributes that define the first two columns of the list view field, DVSmallIcon, DVLargeIcon, and DVNodeText. For the small icon, image (large icon), and list styles, and for some detail list view fields, these attributes are all you need.
- it is easier to assign the data

  You do not need to create a node hierarchy for a list view field as you do with an outline field.
- on Windows 95/NT, the native list view control provides automatic column sorting and resizeable columns

List view styles When you create a list view field, you can choose one of four styles. For the small icon, image (large icon), and list styles, there are always two standard columns, one icon and one text column. (Figure 8-11 shows how each of these styles displays the two "columns.") For the detail style, you can create any number of resizeable columns.

Portability The small icon, image (large icon), and list styles are available on Windows 95/NT only. Therefore, if you plan to deploy your application on more than one window system, we recommend that you use only the detail style. Using the detail style ensures visual consistency between the various platforms.

Sorting On Windows 95/NT, the detail list view field provides automatic sorting, in ascending order, when the end user clicks on the list view field's header. If you wish to provide a descending sort, you can handle the list view field's RequestSort event. See Display Library online Help for information.

For information on providing the data to be displayed in the list view field, see *iPlanet UDS Programming Guide*.

Mapped type for list view fields By default, the mapped type for a list view field is DisplayNode. You can use the DisplayNode class as the mapped type for the following list view styles:

- small icon style list view field
- image style (large icon) list view field

- list style list view field
- detail style list view field with only one text column

For a detail list view field with multiple columns, you must use a *subclass* of DisplayNode for the mapped type. The DisplayNode subclass must define the attributes that provide the data for each column you wish to include. The user-defined DisplayNode subclass must be defined before you can completely define the list view field using the Window Workshop. See *iPlanet UDS Programming Guide* for information on defining the DisplayNode subclass.

Column names and other column properties The ListView Properties dialog provides an array field that allows you to define each of the columns in the list view field. The Column Name property in the array field specifies the name of the attribute that defines the particular column. The column name must be an existing attribute in the DisplayNode class (DVNodeText) or subclass (a user-defined attribute). For each Column Name that you enter, iPlanet UDS creates a corresponding column in the list view field. The other properties in the array field allow you to control the appearance and behavior settings for the individual columns.

➤ **To create a list view field**

1. In the Window Workshop, choose the Widget > New > ListView command or click the New List View tool.
2. On the form, draw a rectangle to indicate the size of the list view field you wish to create.
3. Double-click the list view field to open its properties dialog.
4. On the ListView Properties dialog, use the List Style property to set the style of the list view field. (Small Icon, Image (large icon), and List styles take effect only on Windows 95/NT.)
5. For the Mapped Type property, specify either DisplayNode or a user-defined subclass of DisplayNode. Set other list view field properties as desired (see summary below).
6. Define the individual columns. You must enter the Column Name property for each column you wish to display. The other properties are optional.

   If you are using DisplayNode, use DVNodeText for the first column in the list view field. If you are using a user-defined subclass of DisplayNode, you can use DVNodeText for any column in the list view field and use your user-defined attributes for the remaining columns.

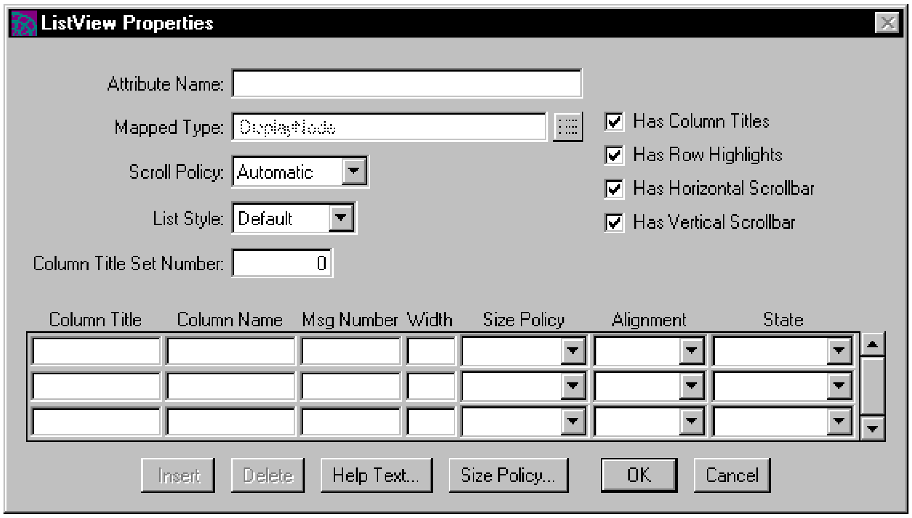
Setting List View Field Properties

The List View Properties dialog, shown in Figure 8-12, allows you to set the following properties for a list view field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the list view field.
Mapped Type	To set the data type for the list view field contents. The type must be the DisplayNode class or a user-defined subclass of DisplayNode.
Scroll Policy	To determine where list view field scrolls a node when it becomes the current node. Values are: Automatic, Top, Bottom, Middle, and No Scroll.
List Style	To set the style of the list view field: Detail, Image (large icon), Small Icon, and List. Image (large icon), Small Icon, and List styles are available only on Windows 95/NT.
Column Title Set Number	The set number for the column titles' message numbers. If the set number is unspecified, the default set number for the window is used. This property is used for creating a multilingual window.
Has Column Titles	To turn the list view field's column titles on or off.
Has Row Highlights	To set row highlighting for the current node in the list view field. Highlighting is reverse video.
Has Horizontal Scrollbar	To turn the horizontal scrollbar on or off.
Has Vertical Scrollbar	To turn the vertical scrollbar on or off.
Column Title	To set the title for an individual column in the list view field.
Column Name	To specify the name of the attribute in the DisplayNode class (or subclass) that defines the data to be displayed in the current column. The attribute is either the DVNodeText attribute or a user-defined attribute in the DisplayNode subclass (the mapped type).
Msg Number	To set the message number for the individual column title. A value of 0 means the current value for the Column Title property. This property is used for creating a multilingual window.

Use This Property	For This Purpose
Width	To set the maximum number of characters that can be displayed in the current column.
Size Policy	To set the size policy for the current column. Values are: Default, Fixed, and Size to Text.
Alignment	To set the alignment of the data displayed in the field. Values are: Default (appropriate for the type), Left, Right, and Center.
State	To set the state of a column in the list view field: draggable, visible, or invisible.
Insert/Delete	To insert or delete a column in the list view field.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-12 List View Properties Dialog

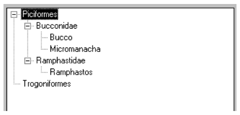

ListView Properties

Attribute Name:
Mapped Type: DisplayNode
Scroll Policy: Automatic
List Style: Default
Column Title Set Number: 0

☑ Has Column Titles
☑ Has Row Highlights
☑ Has Horizontal Scrollbar
☑ Has Vertical Scrollbar

Column Title	Column Name	Msg Number	Width	Size Policy	Alignment	State

Creating a Tree View Field



A tree view field displays hierarchical information in an indented outline, providing controls that let the end user expand and collapse the outline. Unlike an iPlanet UDS outline field, a tree view field has a standardized two-column format, with one small icon column and one text column. On Windows 95/NT, iPlanet UDS uses the native list view control. On all other window systems, iPlanet UDS creates a custom widget.

Very often, user interface designers use list view and tree view fields together. A tree view field on the left allows the user to search through a hierarchy for the item he wants, and a list view field on the right provides the detail information for the selected item in the tree view field.

The TreeView class, which defines the tree view widget, is a subclass of the OutlineField class, and it provides a simplified version of the outline field. The advantages of using a tree view field over an outline field are:

- iPlanet UDS provides the look and feel of a tree view control
- it is easier to define the data

  The DisplayNode class provides attributes that define both columns of the tree view field. These attributes are all you need.

DisplayNode class The data for a tree view field consists of a hierarchy of DisplayNode objects. The DisplayNode class defines a single data node in the tree view field. Each "row" in the tree view field corresponds one to one with a DisplayNode object.

A tree view field displays hierarchical information in a standard format, with two standard columns, a small icon column and a text data column. Therefore, the DisplayNode class provides the attributes you need for these two columns: DVSmallIcon, DVSelectedIcon, and DVNodeText. You do not need to create a subclass of DisplayNode.

For information on providing the data to be displayed in the tree view field, see the *iPlanet UDS Programming Guide*.

➤ **To create a tree view field**

1. In the Window Workshop, choose the Widget > New > TreeView command or click the New Tree View tool.
2. On the form, draw a rectangle to indicate the size of the tree view field you wish to create.
3. Double-click the tree view field to open its properties dialog.

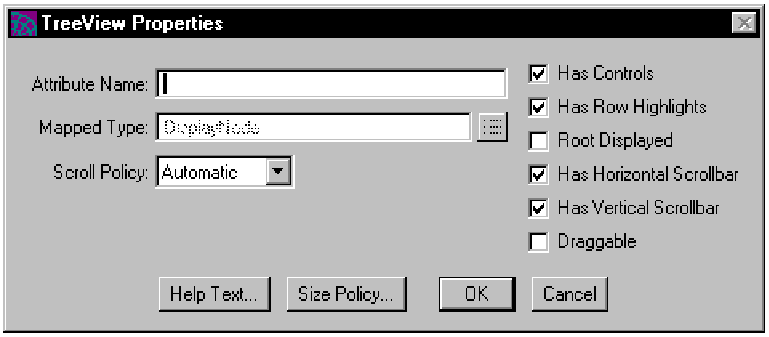
   On the TreeView Properties dialog, set the tree view field properties as desired (see summary below).

Setting Tree View Field Properties

The Tree View Field Properties dialog, shown in Figure 8-13, allows you to set the following properties for a tree view field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the tree view field.
Mapped Type	To set the data type for the tree view field contents. The mapped type must be the DisplayNode class or a subclass of DisplayNode.
Scroll Policy	To determine where tree view field scrolls a node when it becomes the current node. Values are: Automatic, Top, Bottom, Middle, and No Scrolling.
Has Controls	To turn the controls for opening and closing folder nodes on or off.
Has Row Highlights	To turn row highlighting for the current node on or off. Highlighting is reverse video.
Root Displayed	To specify whether or not the root node is displayed.
Has Horizontal Scrollbar	To turn the horizontal scrollbar on or off.
Has Vertical Scrollbar	To turn the vertical scrollbar on or off.
Draggable	To allow the end user to drag the individual nodes in the tree view field.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-13 Tree View Field Properties Dialog



Creating an Outline Field

An outline field provides a browser for multi-column data. It can display a hierarchy of data as an indented outline, or it can display one level of multi-column information. If the data structure is larger than the outline field, the outline field provides a scrollbar that lets the end user scroll through the data.

Unlike list view fields and tree view fields, which use the window systems' native list view and tree view controls, an outline field is a widget created by iPlanet UDS. You can therefore use it to create a customized widget that displays information in any format you choose.

A tree view field displays information in a hierarchical format, however, you are limited to two standard columns, one icon and one label. A list view field displays any number of columns, however, the list view field provides a flat list, without showing hierarchical relationships. An outline field has none of the restrictions of either the tree view or list view fields, and is therefore useful when you wish to display multiple columns of hierarchical information. Generally, if a list view field or tree view field provides the functionality that you need, you should use those fields. When you need more flexibility, use an outline field.

DisplayNode class The data displayed by an outline field is a hierarchy of data nodes, each node corresponding to a row of information. These data nodes are subclasses of the DisplayNode class, described in the Display Library online Help. Each column in the outline field corresponds to an attribute from the DisplayNode subclass to which the outline field maps. To assign the data to the outline field, you must link the DisplayNode subclass objects into a hierarchy and assign the root node of the hierarchy to the mapped attribute of the outline field. The *iPlanet UDS Programming Guide* provides complete information about providing the data for an outline field. See the iPlanet UDS example programs SimpleOutline and FileBrowser for complete code examples using outline field.

➤ **To create an outline field**

1. In the Window Workshop, choose the Widget > New > OutlineField command or click the New Outline Field tool.
2. On the form, draw a rectangle to indicate the size of the field you wish to create.
3. Double-click the field to open its properties dialog.

   On the OutlineField Properties dialog, set the outline field properties as desired (see summary below).

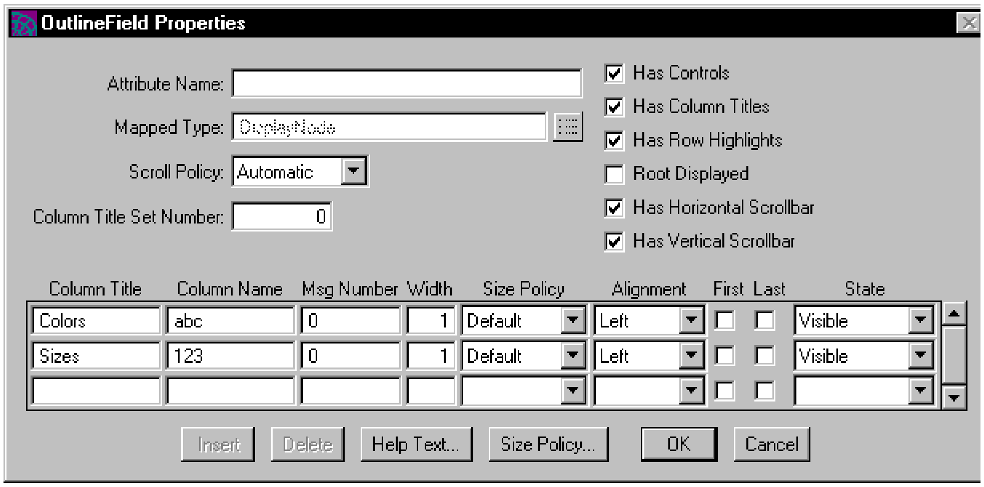
Setting Outline Field Properties

The Outline Field Properties dialog, shown in Figure 8-14, allows you to set the following properties for an outline field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the outline field.
Mapped Type	To set the data type for the outline field contents. The mapped type must be the DisplayNode class or a subclass of DisplayNode.
Scroll Policy	To determine where outline field scrolls a node when it becomes the current node. Values are: Automatic, Top, Bottom, Middle, and No Scrolling.
Has Controls	To turn the controls for opening and closing folder nodes on or off.
Has Column Titles	To turn the outline field's column titles on or off.
Has Row Highlights	To turn row highlighting for the current node on or off. Highlighting is reverse video.
Root Displayed	To specify whether or not the root node is displayed.
Has Horizontal Scrollbar	To turn the horizontal scrollbar on or off.
Has Vertical Scrollbar	To turn the vertical scrollbar on or off.

Use This Property	For This Purpose
Column Title Set Number	The set number for the column titles’ message numbers. This is for use when creating a multilingual window.
Column Title	To set the title for an individual column in the outline field.
Column Name	To specify the name of the attribute in the DisplayNode subclass that defines the data to be displayed in the current column.
Msg Number	To set the message number for the individual column title. A value of 0 means the current value for the Column Title property. This is for use when creating a multilingual window.
Width	To set the maximum number of characters that can be displayed in the current column.
Size Policy	To set the size policy for the current column. Values are Default, Fixed, and Size to Text.
Alignment	To set the alignment of the data displayed in the field. Values are: Left, Right, and Center.
First/Last	To set the column as the first or last indent level in the outline field.
State	To set the state of a column in the outline field: Default, Draggable, Visible, or Invisible.
Insert/Delete	To insert or delete a column in the outline field.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-14 Outline Field Properties Dialog


OutlineField Properties

Attribute Name:
Mapped Type: DisplayNode
Scroll Policy: Automatic
Column Title Set Number: 0

✓ Has Controls
✓ Has Column Titles
✓ Has Row Highlights
☐ Root Displayed
✓ Has Horizontal Scrollbar
✓ Has Vertical Scrollbar

Column Title	Column Name	Msg Number	Width	Size Policy	Alignment	First	Last	State
Colors	abc	0	1	Default	Left	☐	☐	Visible
Sizes	123	0	1	Default	Left	☐	☐	Visible
						☐	☐	

Creating an OLE Field

An OLE field (Windows platform only) displays OLE linked or embedded objects. When your iPlanet UDS application is running, the user can work directly with the linked or embedded object. The OLE field is only available for Windows platforms.

For general information about integrating with OLE and using the OLE field in your iPlanet UDS windows, see *Integrating with External Systems*.

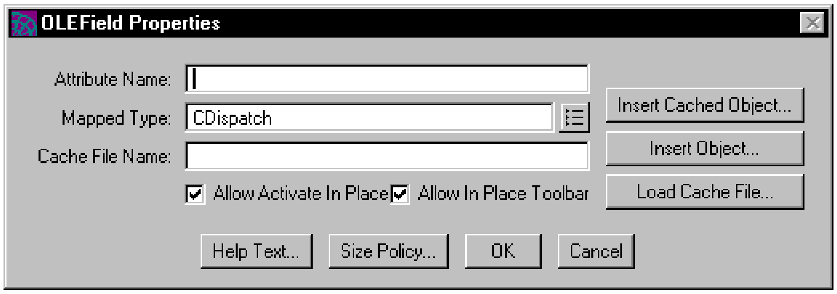
Setting OLE Field Properties

The iPlanet UDS properties dialog for the OLE field allows you to set only a subset of the properties for the OLE field. To set the cache file name, and the default linked or embedded object for the OLE field, you must use the Insert Cached Object button on the OLEField Properties dialog to open the Windows' Cache File and Insert Object dialogs. Because these dialogs are provided by Windows, you need to be using Windows in order to access them. See *Integrating with External Systems* for further information.

The iPlanet UDS OLE Field Properties dialog, shown in Figure 8-15, allows you to set the following properties for an OLE field:

Use This Property	For This Purpose
Attribute Name	Sets an attribute name for the picture field.
Mapped Type	Specifies the mapped data type for the OLE field. This value must be CDispatch or a subclass of CDispatch.
Allow Activate in Place	Sets whether to start the application for the current object as part of the current window.
Allow In Place Toolbar	Sets whether to display the tool bar for the application activated in-place as part of the current window.
Insert Cached Object button	Allows you to add a linked or embedded object that can be saved. The dialog that this button displays is provided by Windows, so you must be running on a Windows machine to access it. See your Windows documentation for information about the Insert Cached Object dialog.
Insert Object button	Lets you add a linked or embedded object. This button displays a dialog, in which you can define the linked or embedded OLE object. See *Integrating with External Systems* for information.
Load Cache File button	Lets you link to an OLE object stored in a cache file. This button displays a file selection dialog to select the cache file.
Help Text	Opens the Help Text dialog for the field.
Size Policy	Opens the SIze Policy dialog for the field.

Figure 8-15 OLE Field Properties Dialog


OLEField Properties

Attribute Name:

Mapped Type: CDispatch

Cache File Name:

☑ Allow Activate In Place ☑ Allow In Place Toolbar

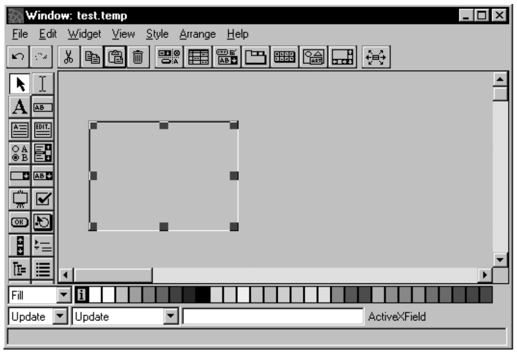
Creating an ActiveX Field

You can use ActiveX controls in the windows of your iPlanet UDS client application. (This only applies to the Windows platform.) Using predefined controls can save you the work of developing complex controls yourself.

When you use an ActiveX control in your iPlanet UDS window, you need to use an ActiveX field to contain the ActiveX control. The ActiveX control is actually a mapped attribute of the ActiveX field. To create the ActiveX field, you use the Widget > New > ActiveXField command.

➤ **To define an ActiveX field**

1. Choose the Widget > New > ActiveXField command.
2. Draw an ActiveX field of the size you want in the window.



➤ **To define the mapped type**

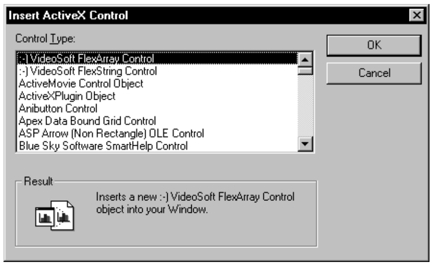
1. Open the ActiveX Field Properties dialog by double-clicking on the ActiveX field.
2. In the Mapped Type field, enter the name of the ActiveX interface class, or use the browser button to display a list of available classes, and select the ActiveX interface class.

   For example, if you want to set the mapped type to the class for the FourDir control, set the Mapped Type field to the FDIRLib.fdir class.

NOTE If the mapped type is not CDispatch, the specified type must match whatever control is inserted; otherwise, you will get a runtime error.

➤ To insert the ActiveX control into the ActiveX field

1. In the ActiveX Field Properties dialog, click the Insert Control button.
2. In the Insert Control dialog, select the ActiveX control that you want to insert into the ActiveX field.


Insert ActiveX Control

Control Type:

:-) VideoSoft FlexArray Control
:-) VideoSoft FlexString Control
ActiveMovie Control Object
ActiveXPlugin Object
Anibutton Control
Apex Data Bound Grid Control
ASP Arrow (Non Rectangle) OLE Control
Blue Sky Software SmartHelp Control

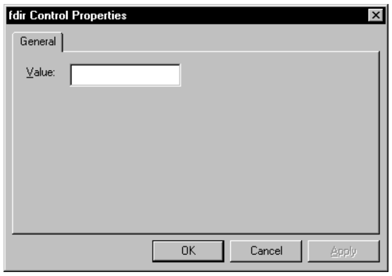
Result

Inserts a new :-) VideoSoft FlexArray Control object into your Window.

NOTE You can only insert ActiveX controls into ActiveX fields on Windows platforms where the ActiveX controls have been installed and registered.

➤ To set the initial property values for the ActiveX control

1. In the ActiveX Field Properties dialog, click the ActiveX Properties button.
2. In the tab folder, set the values that are appropriate for the ActiveX control. For specific information about these properties, see the documentation for the ActiveX control.


fdir Control Properties

General

Value:

NOTE When you change values in this dialog and click either OK or Apply, the values are changed for the ActiveX control, even if you later cancel out of the ActiveX Field Properties dialog. If you click the Cancel button on this dialog, any changed values are not changed for the ActiveX control.

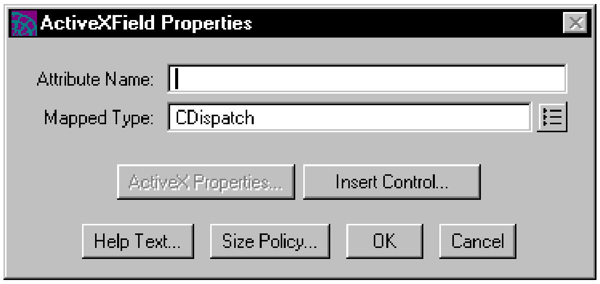
Setting ActiveX Field Properties

The iPlanet UDS properties dialog for the ActiveX field allows you to set only a subset of the properties for the ActiveX field. To select the control you wish to include on your window, you must use the Insert Control button to open the Insert ActiveX Control dialog provided by Windows. To set individual properties on the chosen control, you must use the ActiveX Properties... button, described in the previous section, to open the Control Properties dialog provided by Windows.

The iPlanet UDS ActiveX Field Properties dialog, shown in Figure 8-16, allows you to set the following properties for an ActiveX field:

Use This Property	For This Purpose
Attribute Name	Sets an attribute name for the picture field.
Mapped Type	Specifies the mapped data type for the OLE field. This value must be CDispatch or a subclass of CDispatch.
ActiveX Properties button	Lets you view and set properties of the ActiveX control. This button displays a dialog containing the properties defined by the ActiveX control.
Insert Control button	Lets you add an ActiveX control. This button displays a dialog, in which you can define the type of ActiveX control.
Help Text	Opens the Help Text dialog for the field.
Size Policy	Opens the SIze Policy dialog for the field.

Figure 8-16 ActiveX Field Properties Dialog


ActiveXField Properties

Attribute Name:

Mapped Type: CDispatch

Working with List Widgets

A list widget displays a fixed set of choices to the end user. iPlanet UDS provides five list widgets:

List Widget	Definition
Radio list	Displays a set of radio buttons, from which the end user makes one selection.
Drop list	Displays a list of choices that drop down from the current value. The end user can make one selection.
Scroll list	Displays a scrollable list of choices, from which the end user can make one selection.
Palette list	Displays a set of images, from which the end user can make one selection.
Fillin field	A combination of a data field and a drop list. The end user can either type in a value or select a value from the list.

This section provides general information about how to create list widgets. For details about specific list widgets, see the individual widget descriptions in this chapter.

List elements For every list widget, you must provide the list of values that the list displays to the end user. These lists of values are called *list elements*. For a radio list, drop list, scroll list, and fillin field, you specify list elements as a set of text values. For a palette list, you specify list elements as a set of image values. You specify the list element values by listing them on the widget's properties dialog.

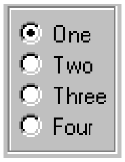
Using integer values to refer to list elements Although you can reference the string or image values for a list widget directly in your TOOL code, you may wish to provide a list of integer values to represent list element text or image values. You can then use the integer value in your code to refer to the corresponding text or image. The integers can be arbitrary numbers, and need not follow a sequence.

Attribute data type The type of the list's data attribute determines how you can refer to the list elements from your code. If you use the string or TextData type, you must refer to the list element by referencing the string or TextData object. If you use the ImageData class for a palette, you must refer to the list elements by referencing the ImageData object. If you use the integer or IntegerData type, you must use the integer value.

See the ListField class in Display Library online Help for information on working with lists in your TOOL code.

Multiple selection for scroll list For a scroll list, you have the choice of specifying that the end user can select any number of items from the list. In this case, the data attribute associated with the scroll list is an integer that represents the number of items currently selected, not the values of the items. For multiple selection scroll lists, you must use the GetElementList, SetElementList, and IsElementSelected methods of the ScrollList class to access and manipulate the values for the list elements. See "Creating a Scroll List" on page 483 for information about creating multiple selection scroll lists and the Display Library online Help for information about working with multiple scroll lists from TOOL.

Creating a Radio List



A radio list displays a set of radio buttons from which the end user can make one selection. Radio lists are useful for presenting the end user a short list of mutually exclusive choices.

In the Radio List Properties dialog, you specify the number of list elements—radio buttons—in the list and the label to use for each button.

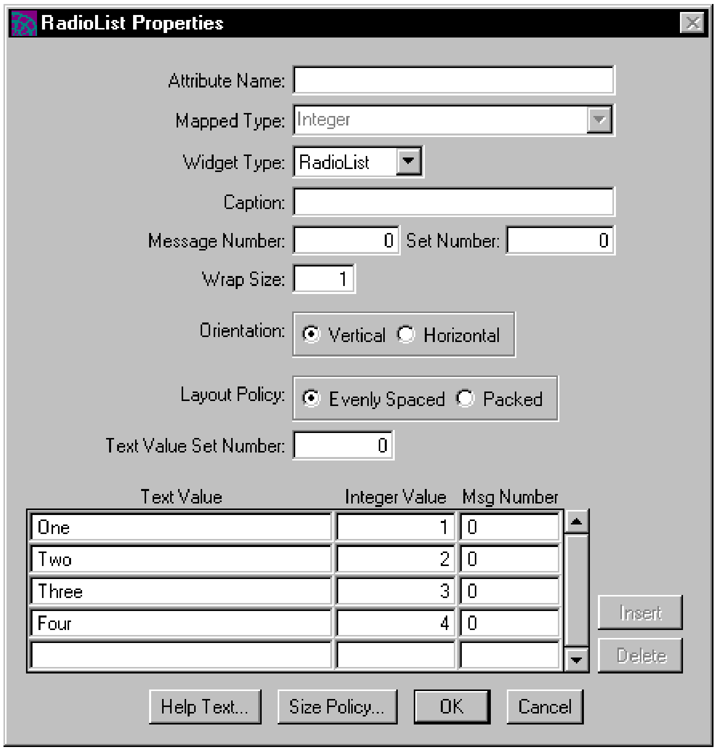
Setting Radio List Properties

The Radio List Properties dialog, shown in Figure 8-17, allows you to set the following properties for a radio list:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the radio list.
Mapped Type	To set the data type for the radio list. Choices are: TextData, string, TextNullable, integer, IntegerData, and IntegerNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.

Use This Property	For This Purpose
Caption	To specify the caption for the radio list. The caption is displayed as part of the radio list's upper border.The Caption property is available only on Windows platforms. On all other platforms, the property is ignored. Note that the caption size can affect the size of the radio list. If the caption is longer than the current width of the radio list, the radio list will be resized to accommodate the caption.
Message Number	To set the message number for the field's caption. This property is for creating a multilingual window.
Set Number	To specify the set number for the caption's message number. If the set number is unspecified, the default set number for the window is used. This property is for creating a multilingual window.
Wrap Size	To set the point at which the radio list wraps into new columns or rows.
Orientation	To set the orientation of the radio list: vertical or horizontal.
Layout Policy	To set the spacing of a horizontal radio list. Values are: Evenly Spaced—all list elements are set to the width of the longest element, or Packed—the width of each list element is determined by the length of its own text.
Text Value Set Number	To specify the set number for the list elements' message numbers. If the set number is unspecified, the default set number for the window is used. This property is for creating a multilingual window.
Text Value	To set the text for the elements in the radio list.
Integer Value	To set the integer value reference for the elements in the radio list.
Msg Number	To set the message number for the list element's text value. This property is used for creating a multilingual window.
Insert/Delete	To insert or delete a list element from the radio list.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

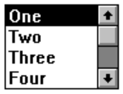
Figure 8-17 Radio List Properties Dialog



Attribute Name:
Mapped Type: Integer
Widget Type: RadioList
Caption:
Message Number: 0 Set Number: 0
Wrap Size: 1
Orientation: [✓ Vertical] Horizontal
Layout Policy: [✓ Evenly Spaced] Packed
Text Value Set Number: 0

Text Value	Integer Value	Msg Number
One	1	0
Two	2	0
Three	3	0
Four	4	0

Creating a Scroll List



A scroll list displays a list of choices, from which the end user can make one selection, or, if you turn on the Multiple Selection toggle, any number of selections. You specify the number of choices that are currently visible. The user can scroll to view the rest. Use a scroll list to present the end user a list of many choices, while displaying only a selected number of choices at any time.

Single selection or multiple selection For a scroll list, you have the choice of specifying whether the user can select only one item from the list or whether she can select any number of items. By default, a scroll list allows single selection only. In this case, like the other iPlanet UDS list widgets, the data attribute associated with the scroll list stores the value of the list item that is currently selected.

However, when you create a multiple selection scroll list, the data attribute associated with the scroll list is an integer that represents the number of items currently selected, not the values of the items. For multiple selection scroll lists, you must use the GetElementList, SetElementList, and IsElementSelected methods of the ScrollList class to access and manipulate the values for the list elements. See the Display Library online Help for information about working with multiple selection scroll lists.

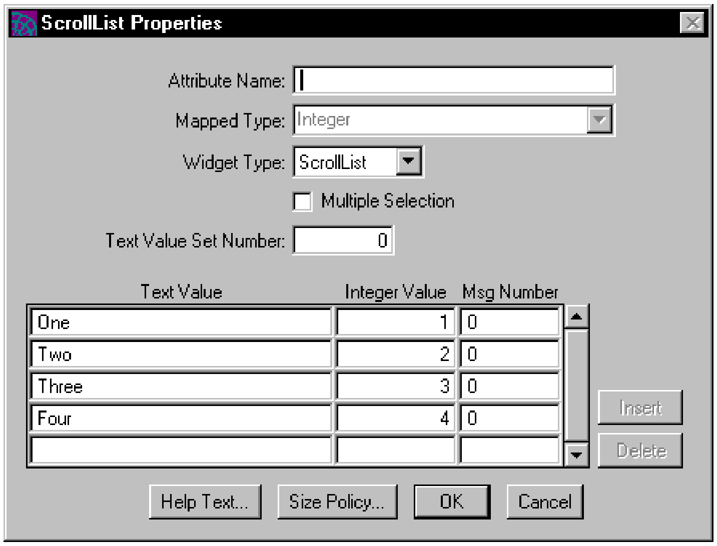
Setting Scroll List Properties

The Scroll List Properties dialog, shown in Figure 8-18, allows you to set the following properties for a scroll list:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the scroll list.
Mapped Type	To set the data type for the scroll list. Choices are: TextData, TextNullable, string, integer, IntegerData, and IntegerNullable. If the Multiple Selection toggle is set to on, the Mapped Type will automatically be set to integer.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Multiple Selection	To specify whether or not to allow the end user to select more than one item from the list.
Text Value Set Number	To specify the set number for the list elements' message numbers. If the set number is unspecified, the default set number for the window is used. This property is for creating a multilingual window.
Text Value	To set the text for the elements in the scroll list.
Integer Value	To set the integer value reference for the elements in the scroll list.
Msg Number	To set the message number for the list element's text value. This property is used for creating a multilingual window.

Use This Property	For This Purpose
Insert/Delete	To insert or delete a list element from the scroll list.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-18 Scroll List Properties Dialog


ScrollList Properties

Attribute Name:

Mapped Type: Integer

Widget Type: ScrollList

☐ Multiple Selection

Text Value Set Number: 0

Text Value	Integer Value	Msg Number
One	1	0
Two	2	0
Three	3	0
Four	4	0

Creating a Drop List

A drop list displays a drop-down list of choices, from which the end user can make one selection. When unselected, a drop list displays only its current value. When selected by the end user, the drop list reveals either its complete list of elements, or, if you so specify, a scrolling list.

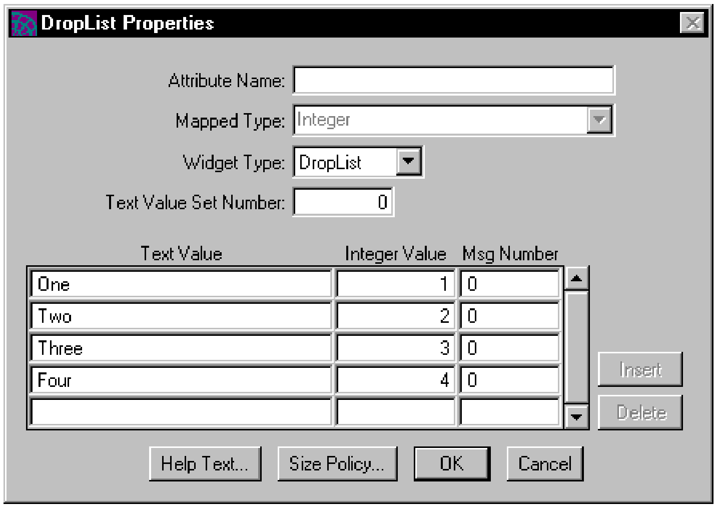
Use a drop list when you want to present a list of choices that are displayed only at user request.

Setting Drop List Properties

The Drop List Properties dialog, shown in Figure 8-19, allows you to set the following properties for a drop list:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the drop list.
Mapped Type	To set the data type for the drop list. Choices are: TextData, string, integer, and IntegerData.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Text Value Set Number	To specify the set number for the list elements' message numbers. If the set number is unspecified, the default set number for the window is used. This property is for creating a multilingual window.
Text Value	To set the text for the elements in the drop list.
Integer Value	To set the integer value reference for the elements in the drop list.
Msg Number	To set the message number for the list element's text value. This property is used for creating a multilingual window.
Insert/Delete	To insert or delete a list element from the drop list.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

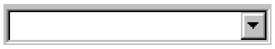
Figure 8-19 Drop List Properties Dialog



Attribute Name:
Mapped Type: Integer
Widget Type: DropList
Text Value Set Number: 0

Text Value	Integer Value	Msg Number
One	1	0
Two	2	0
Three	3	0
Four	4	0

Creating a Fillin Field



A fillin field combines a single-line text field with a list of values from which the user can make a selection. This provides the capabilities of a drop list with some of the capabilities of a text field. In a fillin field, the end user can choose a value or type her own data.

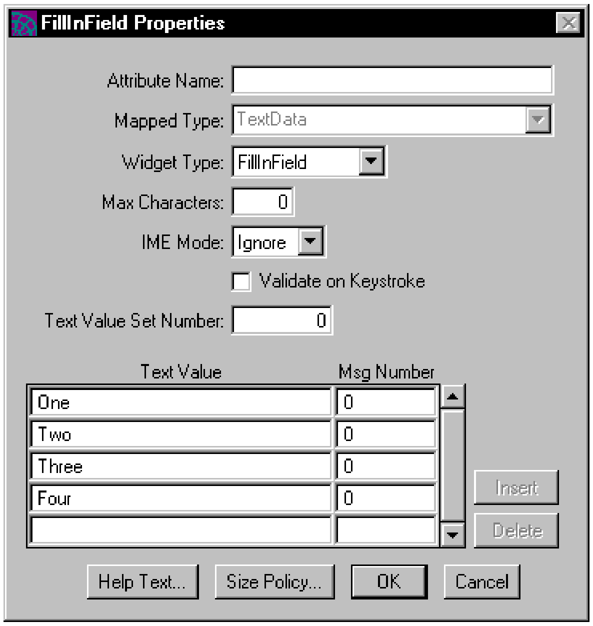
Because there is no guarantee the user will select an element from a fillin field's defined list, fillin fields must use text data types.

Setting Fillin Field Properties

The Fillin Field Properties dialog, shown in Figure 8-20, allows you to set the following properties for a fillin field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the fillin field.
Mapped Type	To set the data type for the fillin field. Choices are: TextData, string, and TextNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Max Characters	To set the maximum number of characters a user can type into the fillin field. Set this to zero for an unlimited number of characters.
IME Mode	To specify whether the double-byte input manager is on or off when entering the field. The default, IGNORE, leaves the current setting, whether on or off, unaltered. The double-byte input manager allows entry of multi-byte characters, such as Kanji.
Validate on Keystroke	To specify whether or not the data stored in the variable associated with the widget is updated after every keystroke by the end user. If this toggle is off, the data stored in the variable associated with the widget is updated after the user exits the field.
Text Value Set Number	To specify the set number for the list elements' message numbers. If the set number is unspecified, the default set number for the window is used. This property is for creating a multilingual window.
Text Value	To set the text for the elements in the fillin field.
Msg Number	To set the message number for the list element's text value. This property is used for creating a multilingual window.
Insert/Delete	To insert or delete a list element for the fillin field.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-20 Fillin Field Properties Dialog


FillInField Properties

Attribute Name:

Mapped Type: TextData

Widget Type: FillInField

Max Characters: 0

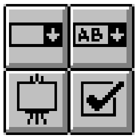
IME Mode: Ignore

Validate on Keystroke

Text Value Set Number: 0

Text Value	Msg Number
One	0
Two	0
Three	0
Four	0

Creating a Palette List



A palette displays a set of images as buttons, from which the end user can make one selection. The widget palette of the Window Workshop, a portion of which is shown at left, is an example of a palette list.

To create a palette list, you must first create the individual picture graphics or picture buttons that you want included on the palette. Although the palette list is not technically a compound field, you use the same technique for creating a palette list as you do for creating a compound field.

➤ **To create a palette**

1. Create the individual picture graphics and picture buttons you want to include on the palette. Arrange them in rows and columns, in the same order as you want them to appear on the palette.
2. Select all the palette pictures.
3. Click the palette icon or choose the Widget > Group Into > PaletteList command.

Because you can use only picture graphics and picture buttons in a palette list, if you try to include other widgets in the palette, iPlanet UDS ignores them.

Order of palette items The order of the images on the palette is determined by the order in which you lay them out on the form. If necessary, you can change the order of the widgets in the palette after creating it by using the Insert and Delete buttons on the palette's properties dialog. This allows you to move the "rows" in the palette into the appropriate order, by deleting them first and reinserting them in the correct order. However, it is easiest for you to place the picture widgets on the form in the correct order before you give the Group Into > PaletteList command.

Palette orientation and wrap size By default, iPlanet UDS creates a vertical palette, with a wrap size equal to the number of widgets in the longest horizontal line of your palette layout. For example, if you lay out two vertical columns of picture buttons, the palette has a wrap size of two. However, if you lay out two horizontal rows with eight picture buttons in each row, iPlanet UDS creates a vertical palette with a wrap size of eight. This is because the default orientation for the palette is vertical. If you need a horizontal palette, you can change its orientation to horizontal by using the Orientation property on the Palette List Properties dialog.

Help for Palette Regions

For a palette list, you can specify float-over and status-line help for individual list items (also referred to as "regions") in the palette.

The ShowRegionFloatOver attribute of the PaletteList class determines whether the float-over help for the individual regions (or list items) in the palette list are actually displayed. By default, this attribute is set to TRUE. When you set the ShowRegionFloatOver attribute to FALSE, the float-over messages for individual regions in the palette are not displayed.

See *iPlanet UDS Programming Guide* for further information about providing float-over help for palette lists.

Multilingual help When you are providing multilingual help for the window, you can use message numbers for the individual float-over and status-line help strings for the individual regions in the palette. The message number for an individual region should point to message that contains both the float-over and status-line messages in the following format:

float_over_help`\n` *status_line_help*

The Text Set Number property for the palette list provides the set number that is used for all regions' message numbers. If no value is specified for the Text Set Number property, iPlanet UDS uses the default message set for the window.

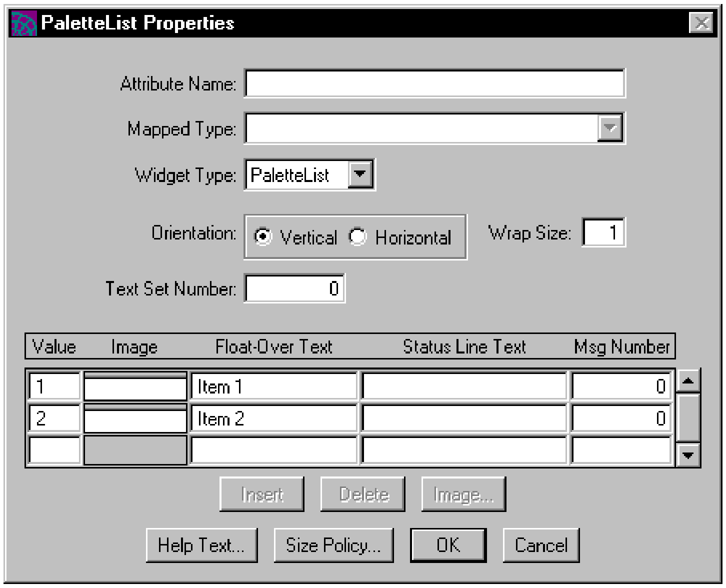
Setting Palette List Properties

The Palette List Properties dialog, shown in Figure 8-21, allows you to set the following properties for a palette list.

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the palette list.
Mapped Type	To set the data type for the palette list. Choices are: TextData, string, TextNullable, integer, IntegerData, and IntegerNullable.
Widget Type	If the current window is a superclass window, you can set the widget type to an abstract class. If the current window is a subclass window and the widget type is abstract, you can use the Widget > Convert To command to convert this widget to a different type.
Wrap Size	To set the number of widgets after which the palette starts a new column or row.
Orientation	To set the orientation of the palette list: vertical or horizontal.
Value	To set the integer value for the elements in the palette list.
Image	To view the images associated with each element in the palette.
Text Set Number	To set the message set number to be used for the region float-over and status-line message numbers. This is for creating multilingual help for the palette list regions.
Float-Over Text	To specify the float-over help associated with the particular palette region.
Status Line Text	To specify the status-line help associated with the particular palette region.
Msg Number	To specify the message number for the message that contains the float-over and status-line help text for the individual region. This is for creating multilingual help for the palette list regions.

Use This Property	For This Purpose
Insert/Delete	To insert or delete a list element from the palette list.
Image button	To open a file selection dialog for changing the image of the selected row.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-21 Palette List Properties Dialog


PaletteList Properties

Attribute Name:

Mapped Type:

Widget Type: PaletteList

Orientation: [✓ Vertical] Horizontal

Wrap Size: 1

Text Set Number: 0

Value	Image	Float-Over Text	Status Line Text	Msg Number
1		Item 1		0
2		Item 2		0

Working with Graphic Widgets

Graphic widgets are decorative widgets, such as rectangles and lines, that you can use as visual organizers for forms or simply as decorations.

Graphic widgets are a variety of simple widget, with two special characteristics that distinguish them from other simple widgets.

First, graphic widgets do not display data.

Second, graphic widgets occupy their own layer in an iPlanet UDS display called the graphic layer. On the graphic layer, widgets can overlap one another. You can use overlapping to create special display effects for graphic widgets that are not possible on the widget field layer, where widgets cannot overlap one another.

One of these special display effects is widget transparency. You can set a picture graphic or other graphic widget to be transparent, so that any graphics below it will show through.

NOTE Points are always transparent.

For more information about widget layering, see Display Library online Help.

Creating a Text Graphic

Text Graphic

A text graphic is a static series of characters that you enter directly onto the form. Use a text graphic to provide the user with information or instructions. Text graphics are also useful for providing titles or labels for other widgets or groups of widgets.

Because it is a graphic widget, you can manipulate a text graphic as you would any graphic widget—stacking it behind or above other graphic widgets or coloring it for decorative effect.

You create a text graphic by clicking in the user window when the Text Graphic tool or menu item is selected. When you click in the window, the Window Workshop places a text graphic in the window with the default label of “Text Graphic,” and switches into text editing mode so that you can immediately type a new label for the text graphic.

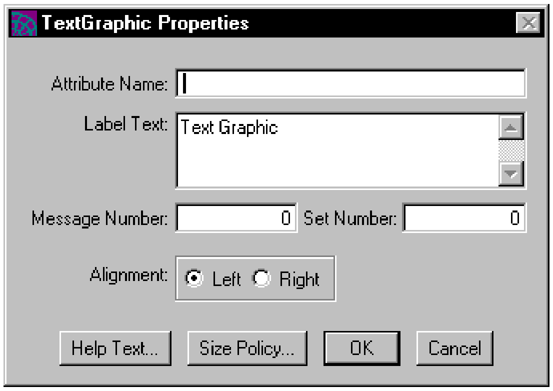
Because it consists of text, you can also edit a text graphic by changing its properties dialog.

Setting Text Graphic Properties

The Text Graphic Properties dialog, shown in Figure 8-22, allows you to set the following properties for a text graphic:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the text graphic.
Label Text	To set the text for the text graphic.
Alignment	To set the alignment of the text in the text graphic: left or right.
Message Number	To specify a message number for the text. This property is for creating a multilingual window.
Set Number	To specify the message set for the text's message number. If no message set is specified, the default message set for the window is used. This property is for creating a multilingual window.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-22 Text Graphic Properties Dialog



Attribute Name:

Label Text: Text Graphic

Message Number: 0 Set Number: 0

Alignment: [✓ Left] Right

Creating a Picture Graphic

FORTÉ

A picture graphic is a static image, which is stored as a permanent part of the form. Use a picture graphic to provide visual information, for example, by displaying a map, or for enhancing the look of a form, for example, by adding a company logo or decorative background.

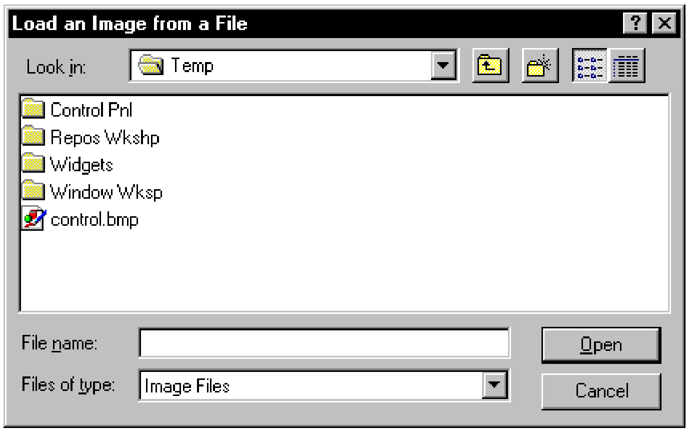
Image Size policy For picture graphics, iPlanet UDS allows you to ensure that the entire image is always displayed by scaling the image to fit the field. Or, you can ensure that the field does not change size by clipping the image to fit into the field. You specify the image size policy on the Size Policy dialog for the picture field. See "Size Policies" on page 439 for information.

Image Gravity property When a picture graphic field is larger then the image it contains, you can set a picture graphic's Image Gravity property to determine how the picture graphic aligns the image within the field. A picture graphic can, for example, center an image using an image gravity setting. You specify the image gravity on the Size Policy dialog for the picture field. See "Size Policies" on page 439 for information.

➤ To create a picture graphic

1. Click the picture graphic icon.
2. Click on the form where you wish to place the picture.

   The Window Workshop opens the Load an Image from a File dialog, where you can choose an image file to use for the picture.



3. In the file selection dialog, choose the image file and click the OK button.

Drag and drop for Windows In Microsoft Windows, you can create a picture graphic by selecting an image file from your file manager and dragging it onto your form.

➤ To drag and drop an image file

1. Select an image file from the file manager.
2. Drag the file onto the form and drop in the appropriate position.

   The Window Workshop creates the picture graphic, using the image file you dropped onto the form.

➤ To replace an existing image

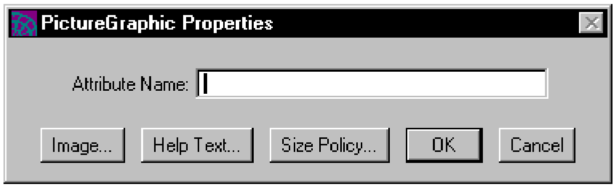
1. Select an image file from the file manager.
2. Drag the file on top of the picture graphic whose image you want to replace. The picture graphic now contains the image file you dropped.

Setting Picture Graphic Properties

The Picture Graphic Properties dialog, shown in Figure 8-23, allows you to set the following properties for a picture graphic:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the picture graphic.
Image…	To change the picture graphic image. The Change Image button opens a file directory dialog where you can select a new image.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-23 Picture Graphic Properties Dialog



Creating a Line

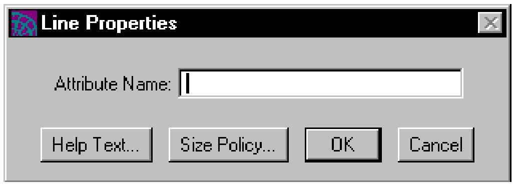
A line is a straight line between two points. You draw a line by clicking at a starting point, dragging the mouse to an ending point, and releasing the mouse button.

Setting Line Properties

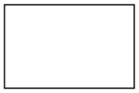
The Line Properties dialog, shown in Figure 8-24, allows you to set the following properties for a picture graphic:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the line.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-24 Line Properties Dialog



Creating a Rectangle



A rectangle is a four-sided shape with right angles. You use rectangles as decorative graphics in your forms.

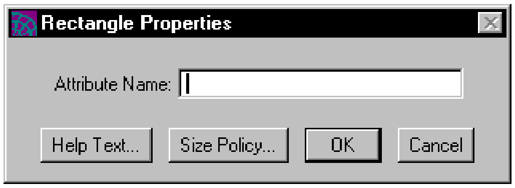
You draw a rectangle by clicking to mark the rectangle's starting point, and then dragging the mouse to the rectangle's ending point and releasing the mouse button.

Setting Rectangle Properties

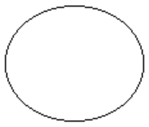
The Rectangle Properties dialog, shown in Figure 8-25, allows you to set the following properties for a rectangle:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the rectangle.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-25 Rectangle Properties Dialog



Creating an Ellipse



An ellipse is an oval shape. You use ellipses as decorative graphics in your forms.

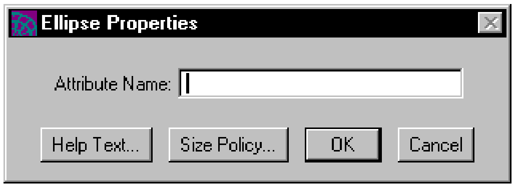
You draw an ellipse by clicking to mark the ellipse’s starting point, and then dragging the mouse to the ellipse’s ending point and releasing the mouse button.

Setting Ellipse Properties

The Ellipse Properties dialog, shown in Figure 8-26, allows you to set the following properties for an ellipse:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the ellipse.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-26 Ellipse Properties Dialog



Creating a Polyline

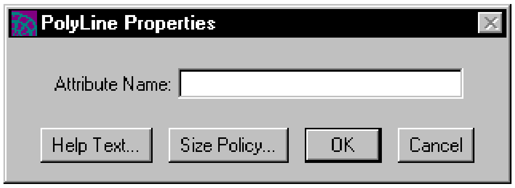
A polyline is a multi-segment line. You draw a polyline from a starting point, through any number of interim points to define segments, to a final end point. You click to mark the beginning polyline point, click to mark interim points for segments, and double-click to mark the final point of the polyline.

Setting Polyline Properties

The Polyline Properties dialog, shown in Figure 8-27, allows you to set the following properties for a polyline:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the polyline.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-27 Polyline Properties Dialog



Creating a Point

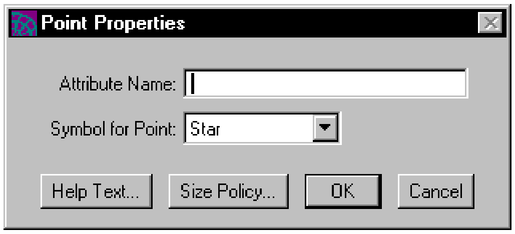
A point is a graphic symbol. You draw a point by clicking to mark the point's location.

Setting Point Properties

The Point Properties dialog, shown in Figure 8-28, allows you to set the following properties for a point:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the point.
Symbol for Point	To set a graphic symbol to represent the point. Choices include triangles at 90 degree angles, circle, square, cross, pixel, diamond, and star (shown above).
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-28 Point Properties Dialog


Point Properties

Attribute Name:

Symbol for Point: Star

Working with Compound Widgets

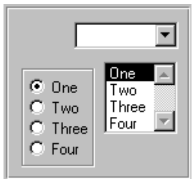
A compound widget is a group of widgets that you can manipulate as a unit. You can move, resize, copy, and delete compound widgets. You set properties for compound widgets the same way you set them for simple widgets. You can set colors, line weights, and other style attributes for compound widgets, while maintaining the style attributes of their child widgets.

You create compound widgets by grouping existing widgets, both simple and compound, into a single compound field.

iPlanet UDS provides the following compound widgets:

Icon	Compound Widget	Definition
	Panel	A subform that organizes fields for collective program action, such as data validation.
	Tab Folder	A set of tabbed pages, which the end user can examine one at a time.
	Grid Field	A table to organize fields into rows and columns for uniform display and portability.
	Compound Graphic	A group of graphic widgets only; contains no fields.
	Viewport	A scrollable field displaying a larger child field.
	Array Field	An array of widgets in a tabular format, like a spread sheet, where the fields are the same from row to row, but their data varies.

Creating a Panel



A panel is a compound widget that allows you to display or manipulate a group of widgets as a unit. Depending on its size policies, a panel has either a fixed boundary or a flexible boundary that changes to accommodate all its child widgets.

You create a panel by selecting one or more widgets, and combining the widgets into a panel, through the Panel tool on the toolbar or the Widget > Group Into command. You can combine any combination of widgets into a panel, including panels and other compound widgets.

You can also create a panel by using the Panel icon on the widget palette. The Panel icon on the widget palette creates an empty panel into which you can drag any combination of widgets, including panels and other compound widgets.

You use panels to organize widgets into subforms, on which you can perform functions apart from the rest of the form. In a data entry application, for example, you might divide a form into several panels so that you could perform data validation for the fields in each panel separately.

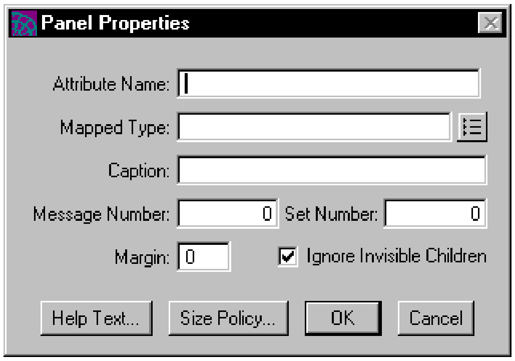
Setting Panel Properties

The Panel Properties dialog, shown in Figure 8-29, allows you to set the following properties for a panel:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the panel.
Mapped Type	To set the panel to map to a class. This is an optional setting; you need not map a panel to a class. However, if you do map the panel to a class, the named attributes of the widgets contained by the panel must match the attribute names and types of the panel's class.
Caption	To specify the caption for the panel. The caption is displayed as part of the panel's upper border. When the panel is a tab page in a tab folder, the Caption property for the panel sets the label used for the tab. See "Creating a Tab Folder" on page 504 for information about tab folders. Note that the caption size can affect the size of the panel. If the caption is longer than the current width of the panel and the panel's size policy is Natural, the panel will be resized to accommodate the caption.
Message Number	To set the message number for the panel's caption. This property is used for creating a multilingual window.
Set Number	To specify the set number for the caption's message number. If the set number is unspecified, the default set number for the window is used. This property is used for creating a multilingual window.
Margin	If the height and width policy are Natural, use this to set the margin between the panel boundaries and the panel's child widgets.
Ignore Invisible Children	If the height and width policy are Natural, this specifies whether or not invisible fields in the panel are used to determine the panel size. By default, invisible children are ignored.

Use This Property	For This Purpose
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-29 Panel Properties Dialog


Panel Properties

Attribute Name:

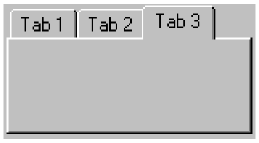
Mapped Type:

Caption:

Message Number: 0 Set Number: 0

Margin: 0 ☑ Ignore Invisible Children

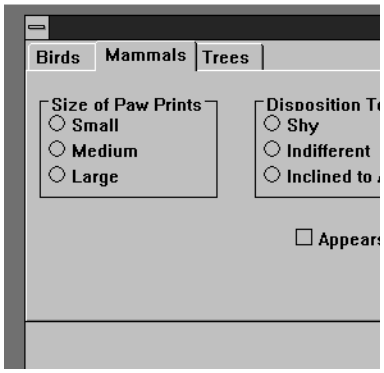
Creating a Tab Folder



A tab folder is a widget that displays one or more pages with labeled tabs. The end user views one tab page at a time by clicking on the tab for the page he wishes to display. Typically, a tab folder provides a set of dialogs, each of which displays a separate group of properties or settings.

Figure 8-30 illustrates a tab folder:

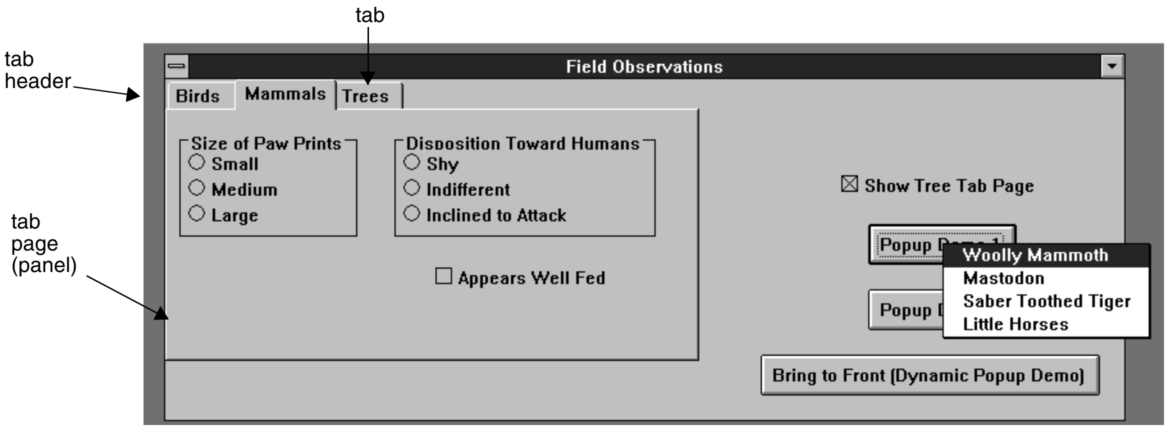
Figure 8-30 Tab Folder



Tab folder widget In iPlanet UDS a tab folder is a compound widget that consists of an array of panels. Each panel has a caption associated with it, and the panel's caption is used as the tab label. The contents of the panel provide the contents of the tab page. When a panel is included in a tab folder widget, it is referred to as a "tab page." However, in the Window Workshop you can open the panel's properties dialog while it is in the tab folder and set any of the panel properties as usual.

Figure 8-31 illustrates the tab folder components:

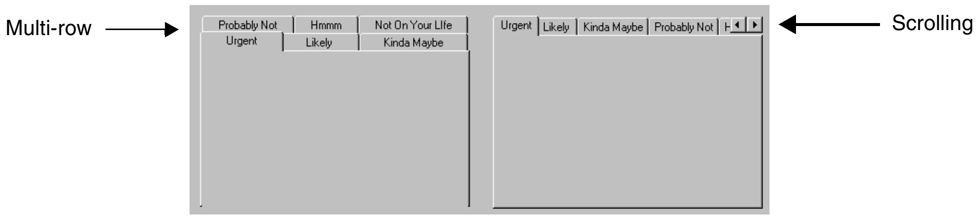
Figure 8-31 Tab Folder Components



Tab folders have two special properties that let you specify how the tabs are sized and displayed: Header Style and Layout Policy.

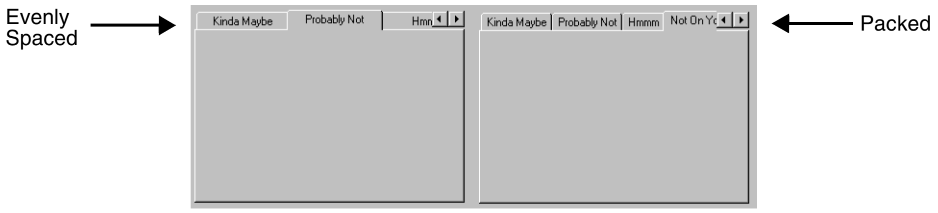
Header Style property By default, when there are more tabs than can fit on the tab header at one time, the tab folder provides a horizontal scrollbar to let the end user view all the tabs. On Windows 95 and Windows NT 4.0/3.5.1, you have the option of displaying multi-row tabs. The Multirow setting for the Header Style property displays the tabs in multiple rows, which lets the end user view all the tabs at once and provides the appearance of a file drawer. The Scrolling setting (the default) for the Header Style property provides a horizontal scrollbar which the end user can use to scroll through the tabs.

Figure 8-32 Header Style Property



Layout Policy property The Layout Policy property specifies whether the tabs are packed or evenly spaced. The Packed setting (the default) creates tabs that are sized individually to accommodate their labels. The Evenly Spaced setting creates tabs with a uniform size—the tab size is determined by the longest label.

Figure 8-33 Layout Policy



Selecting the tab folder To set the properties on the tab folder, you must select the tab folder widget. You can do this by clicking on the background of the tab folder (click in the area to the right of the last tab) or by using Ctrl-Click to select the parent of one of the tab pages (a panel).

There are two ways to create a tab folder in the Window Workshop:

- use the Widget > New > TabFolder command or New Tab Folder tool

  When you use the Widget > New > TabFolder command or the Tab Folder tool on the palette, iPlanet UDS creates a default tab folder, with three empty tab pages. You can then design each of the tab pages, and add or remove tab pages if desired.

- use the Widget > Group Into > Tab Folder command or Group into Tab Folder tool

  Before using the Widget > Group Into > Tab Folder command or Group into Tab Folder tool, you must create the panels or other widgets you wish to include in the tab folder. After grouping the widgets into a tab folder, you can edit them as you wish.

Using the New > TabFolder Command

The New > TabFolder command creates a tab folder widget that contains three empty tab pages. The tabs have the following default labels: Tab 1, Tab 2, and Tab 3.

After creating the tab folder, you can design each of the tab pages by adding widgets to them as you would to any panel. The Window Workshop allows you to select the tab page you wish to edit by clicking on its tab—clicking on its tab brings the tab page to the top.

You can also add or delete tab pages, and reorder the pages as you wish. For information about adding, deleting, and reordering tab pages, see "Editing the Tab Folder" on page 510.

➤ **To create a new tab folder**

1. Choose the Widget > New > TabFolder command or click the New Tab Folder tool.

2. On the form, draw a rectangle to indicate the tab folder's size.

3. Design each of the tab pages by adding widgets to them. Use a grid field within the panel to maintain portability.

Tab labels The label on an individual tab is determined by the Caption property for the panel on the tab page. When you create a new tab folder, iPlanet UDS provides default labels for each tab. To reset the default labels, you must use the Caption property for the individual panels. See "Editing the Tab Folder" on page 510 for information.

Tab properties By default the tabs for the tab folder are set with the following properties:

- scrolling

  If there are more tabs than can fit across the tab folder, iPlanet UDS provides a horizontal scroll mechanism. On Windows 95 and Windows NT 4.0/3.51, you can reset this property to provide multi-row tabs.

- packed

  The Packed setting creates tabs that are sized individually to accommodate their labels. You can reset this property to provide evenly spaced tabs. The Evenly Spaced setting creates tabs with a uniform size—the tab size is determined by the longest label.

Using the Group Into > TabFolder Command

The Group Into > TabFolder command lets you group existing panels, compound widgets, and simple widgets into a tab folder. The Group Into > TabFolder command creates tab pages from the selected widgets as follows:

- every panel becomes an individual tab page
- every compound widget is grouped into a new panel, which in turn, becomes an individual tab page
- all simple widgets are grouped into single panel, which in turn, becomes a single tab page

Page order The order of the tab pages in the tab folder is determined by the order in which the widgets were originally added to the form. The first compound widget is the first tab page, the second compound widget is the second tab page, and so on. After creating the tab folder, you can reorder the tab pages as you wish (see "Editing the Tab Folder" on page 510 for information).

Generally, you design the tab pages in the form of panels before giving the Group Into > TabFolder command. Be sure to use grid fields within the panels to ensure portability. If you wish to specify the tab label for each panel before giving the Group Into command, you can do so by setting each panel's Caption property on the panel's properties dialog.

➤ To create a tab folder

1. On your form, create the widgets you wish to use as or include on tab folder pages.
2. Select all the widgets you want to include in the tab folder.
3. Choose the Widget > Group Into > TabFolder command.

After creating the tab folder, you can edit each of tab page by adding widgets, deleting widgets, or changing widgets on it as you would on any panel. The Window Workshop allows you to select the tab page you wish to edit by clicking on its tab—clicking on its tab brings the tab page to the top.

You can also add or delete tab pages, and reorder the pages as you wish. For information about adding, deleting, and reordering tab pages, see "Editing the Tab Folder" on page 510. For information on setting the properties on the tab folder itself, see "Setting Tab Folder Properties" on page 513.

Tab folder size The Group Into > TabFolder command uses a default size for the tab folder. After creating the tab folder, you can resize it by using the widget's resize handles or by using the Widget > Size Policy... command.

Tab labels The label on an individual tab is determined by the Caption property for the panel on the tab page. When you group widgets into a tab folder, iPlanet UDS provides a default caption for each panel that does not already have one. To reset the default captions, you must use the Caption property for the individual panels. See "Editing the Tab Folder" below for information.

Tab properties By default, the tabs for the tab folder are set with the following properties:

- scrolling

  If there are more tabs than can fit across the tab folder, iPlanet UDS provides a horizontal scroll mechanism. On Windows 95 and Windows NT 4.0/3.51, you can reset this property to provide multi-row tabs.

- packed

  The Packed setting creates tabs that are sized individually to accommodate their labels. You can reset this property to provide evenly spaced tabs. The Evenly Spaced setting creates tabs with a uniform size—the tab size is determined by the longest label.

Editing the Tab Folder

You can use the Edit menu to copy, cut, delete, and paste the individual tab pages within the tab folder the same way you use the Edit menu commands to manipulate other widgets.

After creating a tab folder, you can make any of the following edits:

- add new tab pages
- delete tab pages
- reorder the pages in the tab folder
- change the labels on the tabs

Adding a tab page There are two different ways you can add a new tab page to a tab folder: copy an existing tab folder page or create a new panel.

Copying an existing tab page is useful when the basic design of your tab pages is similar. When you copy an existing tab page, everything on the tab page is duplicated, including the tab label and all widgets on the panel. To copy an existing tab page, you can use the Edit > Copy command or the Edit > Duplicate command. Using the Edit > Copy command enables you to paste the new tab page exactly where you want it. The Edit > Duplicate command always inserts the new tab page at the beginning of the set.

➤ To copy an existing tab page with the Edit > Copy command

1. Select the tab page you wish to copy.
2. Choose the Edit > Copy command.
3. Choose the Edit > Paste command.
4. Indicate the position for the new tab page.

   You can either select a tab page for the new page to precede or you can select a tab page for the new tab page to follow. To indicate which tab page you want your new tab page to precede, click on the left side of the tab. To indicate which tab page you want your new tab page to follow, click on the right side of the tab.
5. Edit the new tab page as desired.

Creating a new panel is useful when you want to design the tab page format from scratch. To provide the label for the new tab page, you can set the panel's Caption property.

➤ To create a new tab page

1. Place the simple and compound widgets for the tab page directly on the form.
2. Choose the Widget > Group Into > Panel command to group the widgets into a panel.
3. Choose the Edit > Cut command.
4. Choose the Edit > Paste command.
5. Indicate the position for the new tab page.

   You can either select a tab page for the new page to precede or you can select a tab page for the new tab page to follow. To indicate which tab page you want your new tab page to precede, click on the left side of the tab. To indicate which tab page you want your new tab page to follow, click on the right side of the tab.

The Header Style property (described under "Setting Tab Folder Properties" on page 513) lets you control how the user views the tabs, with scrolling or in multiple rows. However, we strongly recommend that you limit the number of tabs so that all the tabs are visible without using scrolling or multiple rows.

Deleting a tab page To delete a tab page, you simply remove the panel from the tab folder using the Edit > Delete command.

➤ To delete a tab page

1. Click the tab for the page you wish to delete.

   Clicking the tab brings the selected page to the top of the tab folder.

2. Select the tab page you wish to delete by selecting its panel.

3. Choose the Edit > Delete command.

If you wish to remove the page from the tab folder but save it for use elsewhere, you can use the Edit > Cut command, and then paste the panel wherever you want it.

Reordering tab pages To reorder the tab pages in a tab folder, you must use the Edit > Cut and Paste commands. Use the Cut command to remove the tab page you wish to move, and use the Paste command to paste the tab page into the correct position.

➤ To move a tab page

1. Select the tab page you wish to move.

2. Choose the Edit > Cut command.

3. Choose the Edit > Paste command.

4. Indicate the new position for the tab page.

   You can either select a tab page for the moved page to precede or you can select a tab page for the moved tab page to follow. To indicate which tab page you want your moved tab page to precede, click on the right side of the tab. To indicate which tab page you want your moved tab page to follow, click on the left side of the tab.

Editing the tab label To edit a tab label, you must reset the Caption property for the top-most panel.

➤ To edit a tab label

1. Double-click the panel, or select the panel and choose the Widget > Properties... command.

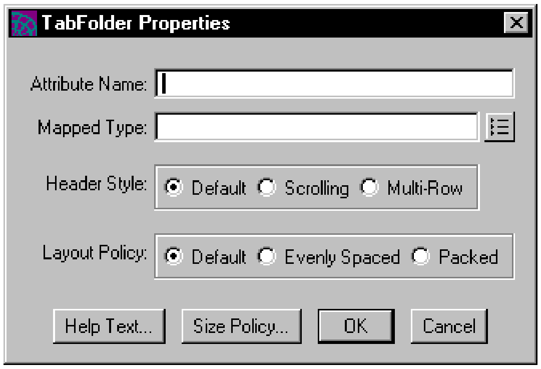
2. On the Panel Properties dialog, enter the tab label in the Caption field.

Setting Tab Folder Properties

The Tab Folder properties dialog, shown in Figure 8-34, allows you to set the following properties for a tab folder:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the tab folder.
Mapped Type	To set the tab folder to map to a class. This is an optional setting; you need not map a tab folder to a class. However, if you do map the tab folder to a class, the named attributes of the widgets contained by the tab folder must match the attribute names and types of the tab folder's class.
Header Style	To specify how the tabs are displayed when there are more tabs than can fit on one line across the top of the widget, Scrolling or Multi-row.
Layout Policy	To specify how the size of the tabs is determined, Evenly Spaced or Packed.
Help Text	To open the Help Text dialog.
Size Policy	To open the Size Policy dialog.

Figure 8-34 Tab Folder Properties Dialog


TabFolder Properties

Attribute Name:

Mapped Type:

Header Style: [✓ Default] Scrolling Multi-Row

Layout Policy: [✓ Default] Evenly Spaced Packed

Creating an Array Field

An array field is compound field that displays an array of widgets in a tabular format, like a spread sheet. In an array field, widgets are the same from row to row, but their data varies. The array field includes tabular data, optional column titles, and an optional scrollbar.

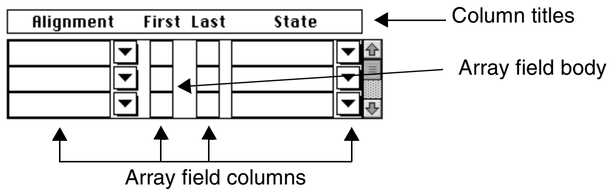
You create an array field by combining a group of widgets into an array field. Each widget in the group becomes a template field for a column in the array field. The array field initially consists of three identical rows. A row consists of one field from each column in the array field.

Within an array field column, information can vary from row to row, but formatting, based on a column's template field, remains uniform.

Let's say that you create an array field from a text field, a data field, and a picture field. You'll get an array field of three columns and three rows. Each row consists of a text field, a data field, and a picture graphic, based on its template fields. When you create new rows, they too will consist of these widgets. From row to row, the widgets that map to data—text fields and data fields—can have different values, while the picture graphic, a static widget which does not map to data, remains constant.

Array field components An array field consists of five components. Each component has its own properties dialog, which you can open by selecting that component. Figure 8-35 illustrates the components of an array field.

Figure 8-35 Array Field Components



The following table describes how to select each of the array field components.

Array Field Component	How to Select It
Single column	Click on any widget in the column.
Single column title	Click on the individual column title.
Column titles (a grid field)	Lasso the titles, or click on an individual column title and then select its parent.
Array field body (a grid field)	Lasso the body of the array field, or select an individual column and then select its parent.
Array field (complete)	Lasso the entire array field, click on the background of the array field, or select the parent of the body or title grid fields.

Modifying an Array Field

After creating an array field, you may wish to add or delete columns, or move existing columns.

Adding a column To add a new column, you can either create a new widget and drop it on top of the array field, or you can cut an existing widget from the form and then paste it on top of the array field.

➤ **To add a new column to the array field**

1. On the widget palette, click the icon for the widget type you wish to add to the array field.
2. Position the cursor over the array field and click to drop the new widget on top of the array field. This will automatically add a column to the array field.

Deleting a column To delete a column from the array field, select the column you wish to delete and then choose the Edit > Delete command. iPlanet UDS automatically adjusts the other columns in the array field.

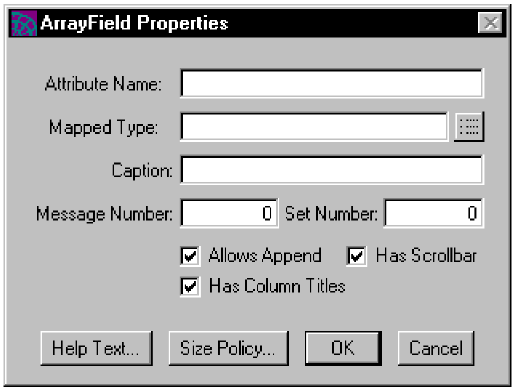
Moving a column To move a column in an array field, select the column you wish to move and drag it into its new position. iPlanet UDS automatically adjusts the other columns in the array field.

Setting Array Field Properties

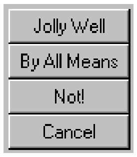
The Array Field Properties dialog, shown in Figure 8-36, allows you to set the following properties for an array field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the array field.
Mapped Type	The type of array to which the array field maps. If you map the array field to a class, the named attributes of the widgets contained by the array field must match the attribute names and types of the array's class.
Caption	To specify the caption for the array field. The caption is displayed as part of the array field's upper border.
Message Number	To set the message number for the array field's caption. This property is used for creating a multilingual window.
Set Number	To specify the set number for the caption's message number. If the set number is unspecified, the default set number for the window is used. This property is used for creating a multilingual window.
Allows Append	To set the array field to allow the end user to automatically add rows.
Has Scrollbar	To give the array field a vertical scrollbar.
Has Column Titles	To give titles to array field columns.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-36 Array Field Properties Dialog



Creating a Grid Field



A grid field is a compound field that arranges its component fields in rows and columns. This is useful for aligning widgets, either for form design or for uniform display across different window systems (geometry management). See the *iPlanet UDS Programming Guide* for a discussion of using grid fields to create a portable user interface.

You create a grid field by creating the widgets to be contained by the grid field and then "grouping" the widgets into a single grid field. The grid field organizes the child widgets into rows and columns of *cells*.

Field size policies The size policies of the child fields within the grid field affect their relationship to the grid field size. If a field in a grid field has a size policy of Parent, iPlanet UDS uses the grid field cell size to determine the field's height or width. When the grid field changes size, the height or width of the child field will automatically be resized to fit the cell. See "Size Policies" on page 439 for information.

Grid field partnerships You can link grid fields together in row and column partnerships, so that each grid field row or column sizes itself with the rows or columns of the other grid fields in the partnership. To link grid fields in a row or column partnership, you use the Arrange > GridFields Into Column Partnership and GridFields Into Row Partnership commands.

Modifying a Grid Field

After creating a grid field, you may wish to add or delete widgets, or move existing widgets.

Adding a widget To add a new widget, you can either create a new widget and drop it on top of the grid field, or you can cut an existing widget from the form and then paste it on top of the grid field.

➤ **To add a new widget to the grid field**

1. On the widget palette, click the icon for the widget type you wish to add to the grid field.
2. Position the cursor over the grid field and click to drop the new widget on top of the grid field.

   iPlanet UDS automatically adjusts the other widgets in the grid field. If necessary, iPlanet UDS automatically adds a column to the grid field.

Deleting a widget To delete a widget from the grid field, select the widget you wish to delete and then choose the Edit > Delete command.

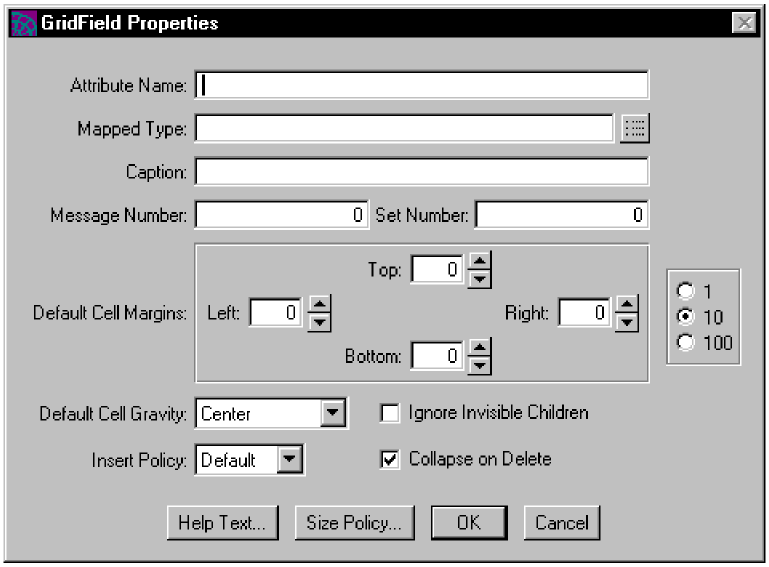
Moving a widget To move a widget in a grid field, select the widget you wish to move and drag it into its new position. iPlanet UDS automatically adjusts the other widgets in the grid field.

Setting Grid Field Properties

The Grid Field Properties dialog, shown in Figure 8-37, allows you to set the following properties for a grid field:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the grid field.
Mapped Type	The class to which the grid field maps. This is an optional setting; you need not map a grid field to a class. However, if you do map the grid field to a class, the named attributes of the widgets contained by the grid field must match the attribute names and types of the grid field's class.
Caption	To specify the caption for the grid field. The caption is displayed as part of the grid field's upper border. Note that the caption size can affect the size of the grid field. If the caption is longer than the current width of the grid field, the grid field will be resized to accommodate the caption.
Message Number	To set the message number for the grid field's caption. This property is used for creating a multilingual window.
Set Number	To specify the set number for the caption's message number. If the set number is unspecified, the default set number for the window is used. This property is used for creating a multilingual window.
Default Cell Margins	To set the margins between grid field cell boundaries and cell contents for the grid field as a whole. (You can set the increments used for the margin steppers to 1, 10, or 100 mils.)
Default Cell Gravity	To set the cell gravity for the grid field as a whole. Cell gravity determines how a cell's contents are aligned within the cell.
Ignore Invisible Children	To specify whether or not the grid field leaves row and column space for cells containing only invisible widgets.
Insert Policy	To set the grid field's policy for accepting or rejecting end user attempts to insert widgets into occupied cells. Policies are to expand the grid field to accommodate new widgets, replace affected widgets, or reject the insertion.
Collapse on Delete	To specify whether the grid field deletes rows and columns when their contents are deleted, or to leave empty rows and columns.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field. See below for information.

Figure 8-37 Grid Field Properties Dialog

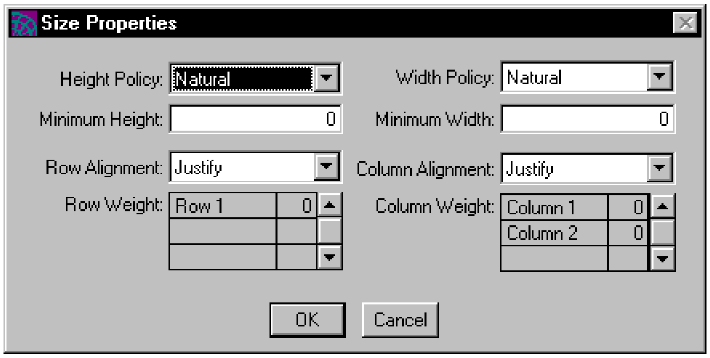

GridField Properties

Attribute Name:	
Mapped Type:	
Caption:	
Message Number:	0
Set Number:	0
Default Cell Margins: Top:	0
Default Cell Margins: Left:	0
Default Cell Margins: Right:	0
Default Cell Margins: Bottom:	0
Increment:	1 [✓ 10] 100
Default Cell Gravity:	Center
Insert Policy:	Default
Ignore Invisible Children	☐
Collapse on Delete	☑

About Grid Field Sizing and Alignment

Size Policy dialog The grid field's Size Policy dialog allows you to set the row and column alignment, and the row and column justify weight for the grid field.

Figure 8-38 Grid Field Size Properties Dialog


Size Properties

Height Policy:	Natural	Width Policy:	Natural
Minimum Height:	0	Minimum Width:	0
Row Alignment:	Justify	Column Alignment:	Justify
Row Weight:	Row 1 0	Column Weight:	Column 1 0
			Column 2 0

Row alignment By default, when the grid field is enlarged vertically, iPlanet UDS justifies the rows in the grid field, adding extra space evenly above, below, and between the rows. By selecting another row alignment option, you can specify that all space is added only above or below the rows, or to both above and below (though not between).

Row justify weight When the row alignment for the grid field is Justify, the amount of space added to each row is based on each row's justify weight. The justify weight for a grid field row determines what percentage of extra space is allocated to the particular row when the grid field that contains the row is enlarged vertically. The justify weight takes effect only when the Row Alignment option for the grid field is set to Justify and the Height Policy properties for the fields in the row are not set to Natural.

By default, the justify weight for all rows is zero. When all rows have a zero justify weight, any extra vertical space is distributed evenly between the rows. However, if you want certain rows in the grid field to get a larger percentage of the space, you can specify the distribution explicitly. For example, in a grid field with three rows, you could assign one row to get 80 percent of the space, while the other two get only 10 percent.

You set the row justify weight for the grid field by using the Size Policy command. On the Size Policy dialog for the grid field, select the row whose row justify weight you wish to set and then enter a number to represent the percentage of space that row should receive.

You can use any numbers you like to represent percentages. The weights are converted by adding them up and calculating a percentage based on the total (normalization). For example, specifying row justify weights for three rows of 1, 1, and 2 is the same as 25 percent, 25 percent, and 50 percent.

Note that if you do not set the justify weight for all of the rows, the rows whose justify weight you do not set keep the default justify weight of zero and so get zero percent of the extra space.

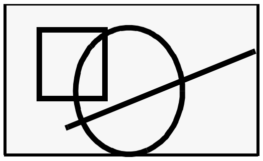
Column alignment The column alignment for the grid field specifies how extra space is added to the grid field when the grid field is enlarged horizontally. By default, iPlanet UDS justifies the columns in the grid field, adding extra space evenly to the left, right, and between the columns. By selecting another column alignment option, you can specify that all space is added only to the left or right of the columns, or to both to the right and left (though not between).

Column justify weight When the column alignment for a column is Justify, the amount of space added to each column is based on each column's justify weight. The justify weight for a grid field column determines what percentage of extra space is allocated to the particular column when the grid field that contains the column is enlarged horizontally. The justify weight takes effect only when the Column Alignment property for the grid field is set to Justify and the Width Policy properties for the fields are not set to Natural.

By default, the justify weight for all columns is zero. When all columns have a zero justify weight, any extra horizontal space is distributed evenly between the columns. However, if you want certain columns in the grid field to get a larger percentage of the space, you can specify the distribution explicitly. This works exactly the same way as the row justify weight described above.

You set the column justify weight for the grid field by using the Size Policy command. On the Size Policy dialog for the grid field, select the column whose column justify weight you wish to set and then enter a number to represent the percentage of space that column should receive.

Creating a Compound Graphic



A compound graphic is a compound field that contains any number of graphic fields or compound graphics. The purpose of a compound graphic is to allow you to combine a set of graphics into a single "picture" that you can display or manipulate as a unit. The main difference between a compound graphic and a panel is that a compound graphic can contain only graphic widgets while a panel can contain any kind of widgets. Also, you can use the Style > Transparent command to make the entire compound graphic transparent.

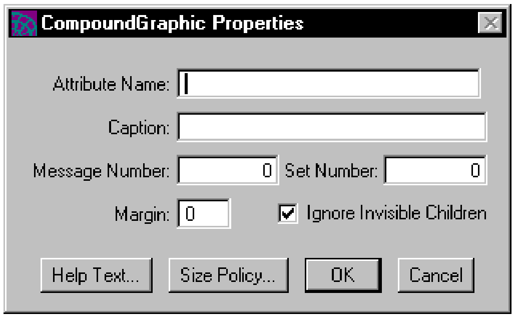
The compound graphic field provides a flexible boundary that expands or shrinks when you move its components.

Setting Compound Graphic Properties

The Compound Graphic Properties dialog, shown in Figure 8-39, allows you to set the following properties for a compound graphic:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the compound graphic.
Caption	To specify the caption for the compound graphic. The caption is displayed as part of the compound graphic's upper border.
Message Number	To set the message number for the compound field's caption. This property is used for creating a multilingual window.
Set Number	To specify the set number for the caption's message number. If the set number is unspecified, the default set number for the window is used. This property is used for creating a multilingual window.
Margin	If the height and width policies for the compound graphic are Natural, use this to set the margin between the compound field boundaries and its child graphic widgets.
Ignore Invisible Children	If the size policies for the field are Natural, use this property to specify whether or not invisible fields in the compound graphic are used to determine the compound graphic size. By default, invisible children are ignored.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-39 Compound Graphic Properties Dialog


CompoundGraphic Properties

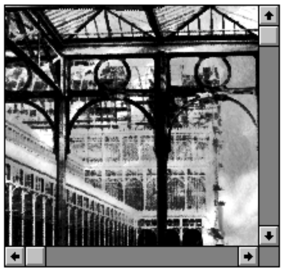
Attribute Name:

Caption:

Message Number: 0 Set Number: 0

Margin: 0 ☑ Ignore Invisible Children

Creating a Viewport



A viewport is a compound field that provides a window onto another, larger widget. The end user can scroll a viewport to bring different parts of the viewport's child widget into view. This allows you to add a widget to a form that would not otherwise fit in a specified area.

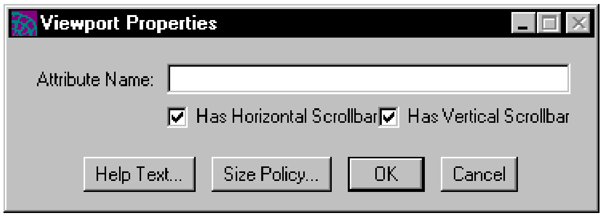
Normally, you use a viewport to display large picture fields or compound fields. If you use a viewport to display a compound field, remember that the viewport can contain only one widget. Therefore, you should create the compound field first, for example, by grouping a set of simple fields into a panel. After you create the compound field, you can then group the compound field into a viewport.

Setting Viewport Properties

The Viewport Properties dialog, shown in Figure 8-40, allows you to set the following properties for a viewport:

Use This Property	For This Purpose
Attribute Name	To set an attribute name for the viewport.
Has Horizontal Scrollbar	To specify whether or not the viewport has a horizontal scrollbar. The property is turned on by default.
Has Vertical Scrollbar	To specify whether or not the viewport has a vertical scrollbar. The property is turned on by default.
Help Text	To open the Help Text dialog for the field.
Size Policy	To open the Size Policy dialog for the field.

Figure 8-40 Viewport Properties Dialog



Chapter 9

Using the Menu Workshop

This chapter provides background information about menu bars and popup menus, and describes how to use the Menu Workshop.

In this chapter, you will learn how to:

- examine a menu bar definition
- create a menu bar
- edit a menu bar
- import and export a menu bar
- test a menu bar
- create popup menus
- implement Help commands on your menu bar
- set Menu Workshop preferences

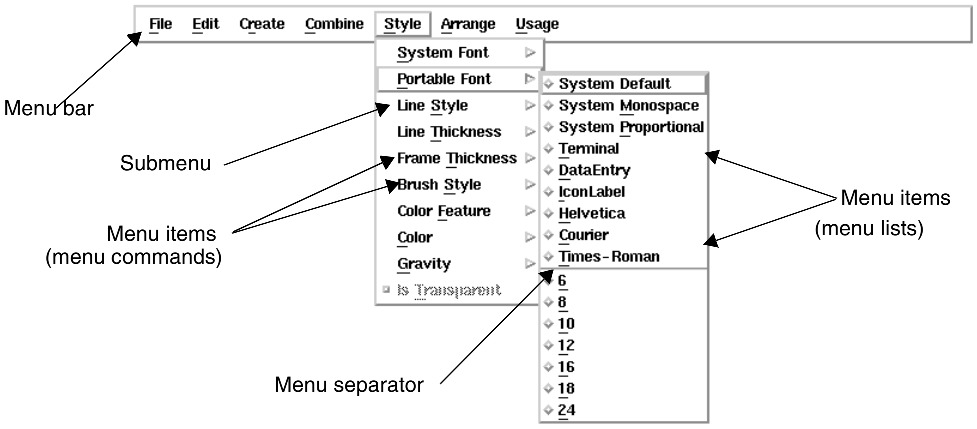
About Menus

A menu bar is a set of pull-down menus available at the top of a window. To create the menu bar for your window, you use menu widgets. The simple menu widgets are controls that let the end user give commands or make selections. The compound menu widget, submenu, lets you create pull-down and slide-off menus.

The following table describes the menu widgets. Figure 9-1 shows some menu widgets as they are used in the Window Workshop.

Tool	Menu Widget	Description
Item	menu command	A simple control that displays a command.
◉Item ○Item	menu list	A simple control that displays a set of options from which the end user can make one selection.
√Item	menu toggle	A simple control that displays a toggle which the end user can switch on or off.
Item Item	menu separator	A horizontal line used to separate items on a submenu.
	submenu	A compound widget that provides a pull-down or slide off menu.

Figure 9-1 Menu Widgets



The following sections provide information about menu widgets in general. This is followed by details about the individual menu widgets.

Popup menus The Menu Workshop also allows you to create popup menus. A popup menu is an independent menu associated with a specific widget on the window. The popup menu is displayed next to the widget when the end user uses the appropriate key combination. See "About Popup Menus" on page 540 for background information about popup menus.

OLE menu groups When your window includes one or more OLE fields (described under "Creating an OLE Field" on page 475, the Menu Workshop allows you to create an OLE menu group. The OLE menu group enables you to merge the submenus belonging to the OLE server application with the TOOL submenus. See "About OLE Menu Groups" on page 543 for information.

Attribute Names

Like a form widget, every menu widget has an optional name. If you plan to manipulate the menu widget from your TOOL code or reference the data that it displays, you should name the widget.

Widget attribute and data attribute The widget name serves two functions. First, the name identifies the attribute that points to the widget. To manipulate the widget from your TOOL code, you must use the widget name enclosed in brackets. Second, for the two menu widgets that display data, menu toggle and menu list, the name identifies the attribute that contains the data. To manipulate the data from your TOOL code, you must use the widget name without brackets.

Using an existing name For menu toggles and menu lists, if the name you give is the same as an existing attribute in the window class and the data type of the attribute is appropriate for the widget, iPlanet UDS uses the attribute for the widget's data.

Using a new name If the name you give is a new name, iPlanet UDS adds a new attribute to the class with the mapping data type you specify on the properties panel (described under "Setting Widget Properties" on page 553).

Mapped type For the two widgets that display data, you can choose the data type of the attribute that contains the data. The following are the data types available for each of the widgets:

Menu Widget	Attribute Data Types
menu toggle	boolean, BooleanData
menu list	string, TextData, TextNullable, integer, IntegerData, IntegerNullable

See "Using iPlanet UDS Data Types" on page 132 for information on these data types.

Menu Widget Events

Except for menu separators, every simple menu widget has one or more events that the end user posts by clicking on the widget. For example, a menu command has an Activate event, which indicates that the end user selected the command.

You use the menu events in your TOOL code to provide processing in response to an end user's actions. For example, your program can respond to the Activate event on a Quit command by prompting the end user before closing the window. The following example illustrates:

```
when <quit_menu>.Activate do
  if self.Window.QuestionDialog
    (messageText = 'Really?',
     buttonSet = BS_OKCANCEL,
     defaultButton = BV_OK) = BV_OK then
        exit;
  end if;
```

The class for a widget determines the types of events that the widget can produce. For example, the MenuList class provides an AfterValueChange event, which is posted when the end user selects a new item from the list. On the other hand, the MenuCommand class provides an Activate event, which is posted when the end user activates the command. The state of the widget (described below) also determines whether or not the event will be posted.

See the Display Library online Help for complete information about the events defined for each of the widgets. See the *TOOL Reference Guide* for information about handing events in your TOOL code.

Status Text for Menu Widgets

You can provide status-line help for individual menu items that is displayed on the Windows and Motif window systems. Status-line help for a menu item is displayed when the mouse pauses over the menu item and status-line help for the window is turned on.

Turning on status-line help To turn on status-line help for the window, you must set up a status line widget on your window and map it to the StatusText attribute of the Window object.

➤ **To provide status-line help**

1. Create a status line field on the window where you wish to display status line help.

   The status line field can be any field that display TextData and can be at any location you wish.

2. Map the status line field to a TextData attribute.

3. In your code, set the value of Window.StatusText to the TextData attribute that is mapped to your status line field.

4. Each time the mouse moves onto a menu item, iPlanet UDS automatically sets the value of the WindowSystem.StatusText attribute to the status text value you specified for the menu item with the Status Text command in the Menu Workshop. The window is automatically refreshed, and the new value of the StatusText attribute is displayed in your window's status line field.

If Window.StatusText is NIL, there is no status line help displayed for the window.

See "Setting Status Line Help Text" on page 554 for information about using the Menu Workshop to specify status-line help for individual menu widgets. See *iPlanet UDS Programming Guide* for general information about implementing status-line help for a window.

NOTE This feature is available on Windows and Motif only. On all other platforms, the status text associated with a menu item is ignored.

Using States for Menu Widgets

Like a form widget, a menu widget is always in a particular state. The state of the menu widget determines how the user can interact with it. For example, when a menu list is set to the Enabled state, the end user can select an item from the list. However, when the menu list is set to the Disabled state, the field is visible but the end user cannot interact with it at all.

States and events The state of the menu widget also determines which events will be posted. For example, a menu command that is set to a Disabled state will not post an Activate event when the end user clicks on the command.

The widget states allow you to control which menu items are available to the end user at a given time. The menu widget states are:

State	Description
Enabled	Visible and selectable.
Disabled	Visible but grayed. Not selectable.
Invisible	Not displayed. Not selectable.

States and usage Like form widgets, the usage of the window determines which state is in effect for a widget. When you create a widget in the Menu Workshop, iPlanet UDS uses a default state of Enabled for the widget for each window usage. However, if you want one of the widgets to be Disabled or Invisible, you can change the widget state for a particular usage. Changing the widget state for a usage determines what the state will be for the widget when that window usage is in effect.

You set the state for an individual menu widget by using the Item > Starting State command in conjunction with the Item > For Usage command.

Changing widget states from TOOL You can also change the widget state from your TOOL code. The State attribute of the MenuWidget class lets you specify the current state of the widget, regardless of what usage is in effect for the window. This is often the simplest and most effective way of changing the states of your widgets. See the Display Library online Help for information on the State attribute.

About Internationalizing Menus

The section "About Internationalizing Windows" on page 382 described how you can use the Window Workshop to create a multilingual window. All menu items except menu separators and the prefabricated commands have Msg Number and Set properties in the Menu Workshop that you can use to specify a message number and optional set number for the individual widget.

Message set used for menu items By default, all messages for a window are loaded from the same message set, which you specify using the Default Set Number property on the Window Properties dialog (see "Setting Window Properties" on page 396). If you do not specify a value for an individual menu item's set number, iPlanet UDS will use the default set number for the window. If you do specify a set number for the individual menu item, that set will be used.

Multilingual Help You can also create multilingual help for individual menu items by using the Message Number and Set Number properties on the widget's Help Properties dialog.

About Submenus

A submenu is a group of menu items displayed either as a pull-down or a slide-off menu. Your menu bar consists of a set of submenus. When you create a submenu in the Menu Workshop, you give it a title and then provide the list of menu items to appear on the submenu.

Note that you can construct the submenu or add items to it dynamically from your TOOL code. For information on this, see the Submenu class in the Display Library online Help.

Pull-down menus A pull-down menu is a set of menu items, which the end user can display by pulling down the menu from its title on the menu bar. In iPlanet UDS, if you add a submenu to your menu hierarchy at the top level, it becomes a pull-down menu on the menu bar.

Slide-off menus A slide-off menu is a submenu that is displayed either on a pull-down menu or another slide-off menu. The end user can display the items on the slide-off menu by sliding to the right of its title. If you add a submenu to your menu hierarchy at any level other than the first, it becomes a slide-off menu. Slide-off menus can be nested to any level.

Submenu type When you create a new submenu, you have the option of specifying the submenu type. The submenu type is either Custom, which means that you provide the submenu title, or one of three prefabricated submenus.

Custom submenu The default type for a submenu is Custom. When you create a custom submenu, you must specify the label that is displayed on the menu.

Prefabricated submenus The prefabricated iPlanet UDS submenus are provided to let you ensure portability across window systems. When you include a prefabricated submenu on your menu bar, iPlanet UDS ensures that the appearance of the submenu (that is, its label and position on the menu bar) is consistent with the standard for each window system on which your application runs. For example, if you include the prefabricated Help submenu, it will appear in the correct position on the menu bar.

The three prefabricated submenus are:

Submenu	Description
File	Uses the window system's standard File menu title and position on the menu bar.
Edit	Uses the window system's standard Edit menu title and position on the menu bar.
Help	Uses the window system's standard Help menu title and position on the menu bar. See "Using the Prefabricated Help Commands" on page 561 for further information.

Note that these prefabricated submenus do not contain commands; you must provide the commands yourself. See "About Menu Commands" on page 535 for information on prefabricated commands you can use on the File, Edit, and Help menus.

Events for submenus A submenu has two events: ChildActivate and ChildAfterValueChange. You use these child events primarily to add items to the submenu dynamically from your TOOL code. In order to handle events on a menu item that was added to the menu dynamically, you must wait for the ChildActivate and ChildAfterValueChange events posted on its parent submenu. See the Display Library online Help for information about these events.

The following table briefly describes the submenu properties:

Submenu Property	Description
Attribute Name	The attribute name for the submenu must be a new name. A submenu does not map to a data attribute.
Submenu Type	Either a custom submenu or one of a set of prefabricated submenus.
Label Text	The menu title that is displayed either on the menu bar (for pull-down menus) or on the menu (for slide-off menus).
Msg Number	Specifies the message number for the widget, which is used for providing a multilingual menu.
Set	Specifies the set number for the widget's message number.

About Menu Commands

A menu command is a single command that the end user can choose to give an instruction to the application. Normally, you provide the code that is executed when the user activates the command. However, for your convenience, iPlanet UDS provides a set of prefabricated commands that you can include on your menu bar if it is appropriate.

Command type When you create a new menu command, you have the option of specifying the command type. The command type is either Custom, which means that you provide the code that is executed when the end user gives the command, or one of the seven prefabricated commands.

Custom commands The default command type for a menu command is Custom. When you create a custom menu command, you must specify the label that is displayed on the menu. Then, in your program, you must provide the code that is executed when the end user chooses the command. For example, for a custom "Properties" command, you specify a label of "Properties" and then provide the code to display the properties dialog and change property settings.

Prefabricated commands iPlanet UDS provides prefabricated commands to let you ensure portability across window systems. When you include a prefabricated command on a menu, iPlanet UDS ensures that the appearance of the command (that is, the text and shortcut key) and, in some cases, its behavior, is consistent with the standard for each window system on which your application runs. For international applications, prefabricated commands automatically use the translated command provided by the current operating system.

The prefabricated commands fall into four categories:

Categories	Description
Edit menu commands	Cut, Copy, Paste, Delete, Undo, Redo, Find, Find Next, Replace, Replace Next, and Select All commands that look and behave according to your window system standards. The automatic behavior for the Cut, Copy, Paste, Delete, and Select All commands for subclasses of CharacterField (text field, text edit field, data field, and fillin field). The automatic behavior for the Undo and Redo commands take effect only for the text edit field.
Print commands	A Print command that prints the current window and a Print Setup or Page Setup command that opens the window system's print or page setup dialog. These commands are described in further detail below.
Help commands	Help Contents and Help Search commands, which automatically access the default Help file. A Help on Help command, which automatically accesses the appropriate Windows Help screen. An About command, which has no default behavior. See "Using the Prefabricated Help Commands" on page 561 for more information.
File menu commands	Close and Exit commands for inclusion on a File menu. These provide the window system standard name and shortcut key. However, you must implement the behavior of the command.

Note that if you use an event statement to provide event handling code for any of the prefabricated commands, your code will override the default behavior that iPlanet UDS provides for these commands.

Printing commands The prefabricated Print command prints the current window. The iPlanet UDS Print command uses the RenderAsImage method in the FieldWidget class to convert the window into image data, and then sends the image to the printer specified by the Print Setup or Page Setup command. The image that is printed includes only those parts of the window that are visible on the screen.

The prefabricated Print Setup command displays the window system's Print Setup or Page Setup dialog, where the end user can specify basic printing options. When the end user clicks the OK button on the print dialog, the updated print option information is reflected in the DefaultPrintOptions attribute for the window system. You can then use this DefaultPrintOptions object whenever you need to specify or set print options. See DefaultPrintOptions attribute of the WindowSystem class in the Display Library online Help for information about setting print options. See the PrintDocument class in the Display Library online Help for general information about printing.

Text label for custom command For custom commands, you specify the text that is displayed on the menu. If you wish, you can also specify a single mnemonic character that can be used to activate the command. To do this, insert an ampersand character before the appropriate character in the label. For example, if you specify "&Save" as the label, the "S" will be the mnemonic character. On window systems that support mnemonic characters, the mnemonic character will be underlined. On window systems that do not support mnemonic characters, the mnemonic character will simply be ignored.

Shortcut key for custom command For a custom menu command, you can specify a shortcut key that the end user can enter to activate the command. The drop list on the property panel lets you choose either a standard window system shortcut key or key combination.

Is Input Finalized Normally, when the end user chooses a menu command, the data validation events for the field that the end user has just left will be automatically posted. For example, if the end user chooses a Close command after entering data into a text field, the AfterValueChange events for the text field are posted. This ensures that the data validation code for the text field is executed before the window is closed.

However, under certain circumstances, you may wish to prevent automatic data validation. For example, if you include a Cancel command that lets the user close the window without making any changes, you do not want the data validation code for any fields on the window to be executed. In this case, iPlanet UDS lets you turn off the Is Input Finalized option for the menu command. When this option is turned off, any AfterValueChange events for the previous field will be ignored.

The following table briefly describes the menu command properties:

Menu Command Property	Description
Attribute Name	The attribute name for the menu command must be a unique name. A menu command does not map to a data attribute.
Command Type	Either a custom command or one of a set of standard commands.
Is Input Finalized	Turning this off suppresses AfterValueChange events for the field the user interacted with before activating the menu command.

Menu Command Property	Description
Label Text	The text that appears on the menu for this command button.
Short Cut	For a custom command, a key or key combination the end user can type to activate the menu command.
Msg Number	Specifies the message number for the widget, which is used for providing a multilingual menu.
Set	Specifies the set number for the widget's message number.

About Menu Toggles

√Item

A menu toggle displays a label along with a toggle or a check box that the end user can switch on or off. The Test Window menu item in the Window Workshop is an example of a menu toggle.

Data for a menu toggle The mapped type for a toggle's data attribute can be either a simple boolean type or the BooleanData class. These data types represent the current setting of the menu toggle as either TRUE (for on) or FALSE (for off).

Events for menu toggles A menu toggle has two events: Activate and AfterValueChange. Essentially, they both mean the same thing. The Activate event is posted whenever the user clicks the toggle. Since this always changes the toggle's value, the AfterValueChange is also posted. You can use either event in your event handling code.

The following table briefly describes the menu toggle properties:

Menu List Property	Description
Attribute Name	The name of the toggle and the name of the attribute that contains the current value of the toggle.
Label Text	The text displayed on the menu next to the toggle.
Mapped Type	The data type of the attribute that contains the current value of the toggle: boolean or BooleanData.
Msg Number	Specifies the message number for the widget, which is used for providing a multilingual menu.
Set	Specifies the set number for the widget's message number.

About Menu Lists

A menu list displays a fixed set of choices from which the end user can make one selection. When you create the menu list, you provide the list of string values that are displayed to the end user.

If you wish, you can reference these string values directly in your TOOL code. However, you also have the option of providing a list of integer values to represent the strings. You can then use the integer value in your code to refer to the corresponding string. The integers can be arbitrary numbers, and do not need to be in sequential order.

Note that you can construct the menu list or add items to it dynamically from your TOOL code. For information on this, see the Display Library online Help.

Mapped type for menu lists The mapped type of the menu list's data attribute determines how you can refer to the list elements from your code. If you use the string, TextData, or TextNullable type, you must refer to the list element by referencing the string or TextData object for the label text. If you use the integer, IntegerData, or IntegerNullable type, you must use the integer value in the List Item Value property.

Events for menu lists Menu lists have two events that the end user can cause to be posted: Activate and AfterValueChange. The Activate event is posted every time the user clicks a menu item, even if he or she clicks the list item that is currently selected. The AfterValueChange event is posted only when the end user selects a different item on the list. Depending on your particular needs, you should use whichever event is most appropriate.

The following table briefly describes the menu list properties:

Menu List Property	Description
Attribute name	The name of the menu list and the name of the attribute that contains the current value of the menu list. This must be the same for all the individual elements on the list.
Label Text	The text displayed by the individual menu item on the list.
Mapped type	The data type of the attribute that contains the current value of the menu list: string, TextData, TextNullable, integer, IntegerData, or IntegerNullable.

Menu List Property	Description
List Item Value	For lists with a numeric mapped type, the list item value is the integer value for the individual list item.
Msg Number	Specifies the message number for the widget, which is used for providing a multilingual menu.
Set	Specifies the set number for the widget's message number.

About Menu Separators

A menu separator is a horizontal line that you can use to divide menu items into separate sections.

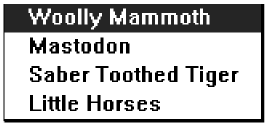
The following table briefly describes the menu separator properties:

Menu Separator Property	Description
Attribute Name	The attribute name for the separator must be a new name. A separator does not map to a data attribute.

About Popup Menus

A popup menu is an independent menu associated with a specific widget on the window. The popup menu is displayed next to the widget when the end user uses the appropriate key combination. Figure 9-2 illustrates a popup menu:

Figure 9-2 Popup Menu



Activating a popup menu The key combination needed to activate a popup menu depends on the particular window system. To activate a popup menu:

Window System	Key Combination to Activate Popup Menu
Windows	Use mouse outer button, or select the field and use Shift F10. The outer mouse button is the right-most button on a right-handed mouse and the left-most button on a left-handed mouse.
Motif	Use mouse outer button, or select the field and use Shift F10. The outer mouse button is the right-most button on a right-handed mouse and the left-most button on a left-handed mouse.

iPlanet UDS uses the position of the cursor at the time the popup key combination is entered to determine which popup menu to display. If the widget over which the cursor is positioned has a popup menu assigned to it, iPlanet UDS displays the current widget's popup menu. If the current widget does not have a popup menu assigned to, iPlanet UDS displays its parent's popup menu. If the parent widget does not have a popup menu assigned to it, iPlanet UDS displays the popup menu for the parent's parent, and so on. If the top-level container for the widget, the Window object, does not have a popup menu, nothing happens.

Choosing a menu item from a popup Once the popup menu is displayed, the end user can choose a menu item from it the same way he chooses a menu item from a menu on the menu bar.

Closing a popup menu The end user can close the popup menu any time with the appropriate action. To dismiss the popup menu:

Window System	Key Combination to Dismiss Popup Menu
Windows	Click outside the menu or press Esc key.
Motif	Click outside the menu or press Esc key.

In iPlanet UDS, a popup menu is a submenu, just like a menu on the menu bar. Like any pull-down or slide-off menu, a popup menu can contain any number of menu toggles, menu lists, slide-off menus, and menu separators. The only difference between a popup menu and a menu-bar menu is that the menu-bar menu automatically appears on the menu bar while the popup menu appears on the window when the end user activates it.

Creating a popup You create the popup menu using the Menu Workshop the same way you create any submenu. Each time you add a new submenu to the menu bar, you specify whether it is a menu-bar menu, whether it is a popup menu, or whether it is both a menu-bar menu and a popup menu. A submenu that is both a menu-bar menu and a popup menu appears as a standard menu on the menu bar and, when it is activated, as a popup menu next to the widget to which it is assigned. See "Creating Popup Menus" on page 558 for details about using the Menu Workshop to create a popup menu.

Of course, you can also create popup menus in your TOOL code, as you would any submenus, and add or delete menu items dynamically. See the PopupUsage attribute in the Display Library online Help for information about creating popup menus dynamically.

PopupMenu attribute Once the popup menu is created, you must use TOOL code to display it. You can either assign the popup menu to a field widget so that iPlanet UDS automatically displays it when the end user activates it or you can explicitly display the popup yourself.

To assign the popup to a field widget, you use the field widget's PopupMenu attribute. When the submenu is assigned to the PopupMenu attribute, iPlanet UDS automatically displays the popup menu. See the Display Library online Help for information about the PopupMenu attribute.

Typically, you assign popups to field widgets in the Init method for the UserWindow class. The following example illustrates:

```
-- Assign a Popup SubMenu to a PushButton field.
<PopupDemoButton>.PopupMenu = <MoveTabSubMenu>;
```

PopupRequest event To explicitly display the popup, you can handle the PopupRequest event on the window, and use the Window.ShowPopupMenu method to display the popup menu when and where you wish. See the Display Library online Help for information.

Activate and AfterValueChange events The events posted when the end user chooses a menu item from a popup menu are the same events, Activate and AfterValueChange, that are posted when the end user chooses a menu item from a pull-down or slide-off menu. Both these events have a menuType parameter, which allows you to detect whether the event was posted from a popup menu or a menu bar menu. You handle the Activate and AfterValueChange events in your code the same way you handle them normally. The following code illustrates:

```
-- When the user selects an item on the popup menu, respond.
when <TabOneItem>.Activate do
  <BasicTabFolder>.TopPageIndex = 1;
```

When the end user activates the popup menu, iPlanet UDS posts the PopupRequest event on the window, indicating which field widget received the request. By default, iPlanet UDS automatically handles the PopupRequest event by displaying the popup menu. However, you may wish to provide your own code to display the popup menu. You can explicitly handle the PopupRequest event in your code, and use the ShowPopupMenu method to display the menu at the location you specify. See the Display Library online Help for information.

The section "Creating Popup Menus" on page 558 provides information about using the Menu Workshop to create a popup menu. The PopupUsage attribute section in the Display Library online Help provides a brief description of how to create a popup menu dynamically.

About OLE Menu Groups

When your window includes one or more OLE fields (described under "Creating an OLE Field" on page 475, you may wish to create an OLE menu group. This feature is available for Windows only.

Microsoft defines three groups for submenus: File, View, and Window. You can define the menus of a TOOL window as belonging to one of these groups. These groups define how the TOOL window merges the submenus belonging to the OLE server application with the TOOL submenus when you edit an embedded OLE object in place in a TOOL window. The OLE menus are displayed only when the user is editing an embedded object in place by clicking on the object.

For general information about integrating with OLE, see *Integrating with External Systems*

The Item > OLE Menu Group command lets you define TOOL submenus that merge with the OLE application submenus.

The OLE Menu Group command offers the following settings:

Setting	**Description**
Invisible	Menu is not displayed when the embedded OLE object is edited in place. This is the default value.
File Group	Menu is displayed before the File menu of the application that edits the embedded OLE object. If there is no File menu in the editing application, this menu is displayed before any iPlanet UDS View Group menus.
View Group	Menu is displayed after the File menu and before the View menu of the application that edits the embedded OLE object. If there is no View menu in the editing application, the menu is displayed after the application's File menu and before any iPlanet UDS Window Group menus.
Window Group	Menu is displayed after the View menu and before the Window menu of the application that edits the embedded OLE object. If there is no Window menu in the editing application, the menu is displayed after the application's View menu.

For example, a TOOL window has the following submenus with the following OLE Menu Group settings in this order:

- AppFile—File Group
- AppEdit—Invisible
- Select—View Group
- Data—View Group
- Special—Window Group

The OLE server application that edits an embedded OLE object has the following submenus in the following order:

- File—File Group
- Edit—File Group
- View—View Group
- Window—Window Group

When the end user double-clicks the embedded OLE object, the following submenus appear in the menu bar of the TOOL window in the following order:

- AppFile (TOOL)
- File (OLE server application)
- Edit (OLE server application)
- Select (TOOL)
- Data (TOOL)
- View (OLE server application)
- Special (TOOL)
- Window (OLE server application)

Using the Menu Workshop

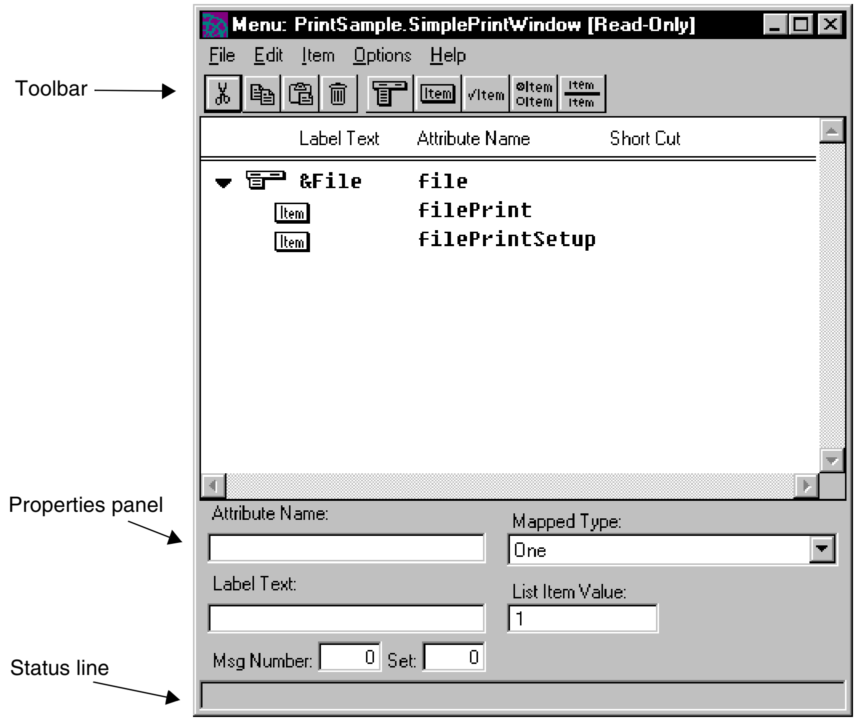
You can open the Menu Workshop either from the Class Workshop or from the Window Workshop using the File > Menu… command. From the Class Workshop, you can also open the Menu Workshop by single-clicking the Menu Workshop tool.

If a menu bar already exists for the window, the Menu Workshop displays the existing menu bar. If a menu bar does not exist for the window, the Menu Workshop displays an empty menu.

The Menu Workshop Window

The Menu Workshop provides an outline field, where the menu bar is displayed in a hierarchical form, and a panel, where you can view or set the properties for the currently selected menu item. The toolbar provides a palette of menu widgets, which you use to add new widgets to the menu bar, and a set of tools for basic operations. Figure 9-3 shows the Menu Workshop, and Figure 9-4 shows the Menu Workshop Toolbar.

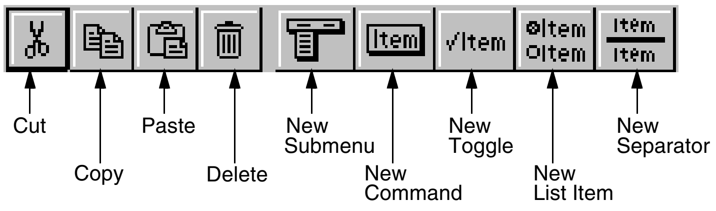
Figure 9-3 Menu Workshop



Label Text	Attribute Name	Short Cut
&File	file	
	filePrint	
	filePrintSetup	

Attribute Name:
Mapped Type: One
Label Text:
List Item Value: 1
Msg Number: 0 Set: 0

Figure 9-4 Menu Workshop Toolbar



Status line The status line in the Menu Workshop provides hints about the steps you should follow next. It is a good idea to check the status line as you proceed with your work.

Access to Other Workshops

From the Menu Workshop, you can access one other workshop:

Workshop	How to access it
Class Workshop	The File > Open Class… command opens the Class Workshop to display the definition of the class to which the current menu bar belongs. If the class is already being displayed, the Open Class… command moves the input focus to the appropriate Class Workshop window.

Leaving the Menu Workshop

To leave the Menu Workshop, use the File > Close command. When you leave the Menu Workshop, iPlanet UDS creates the widget attributes for all the menu widgets you have added to the menu bar.

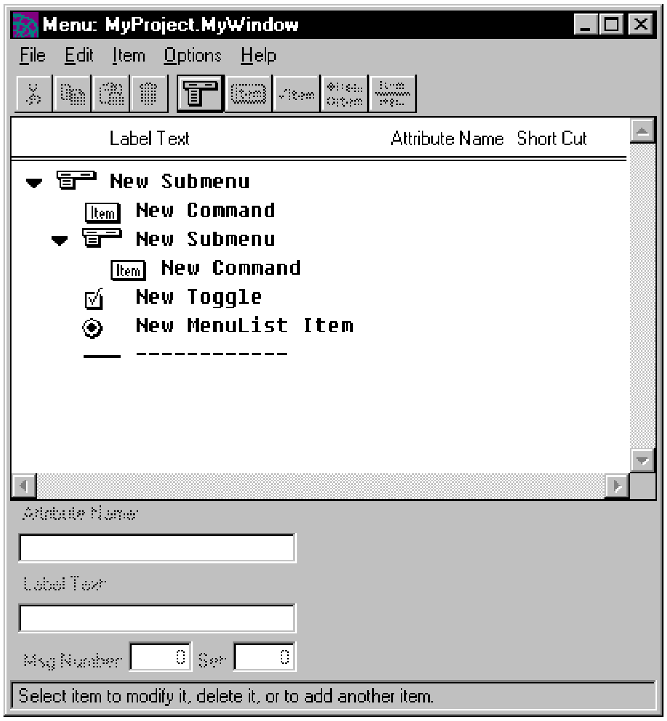
If you wish to remove all changes you have made to the menu bar since you opened the Menu Workshop, you can use the File > Cancel command. The Cancel command closes the Workshop and ignores all work done since the workshop was opened or since your last Save All command.

Examining a Menu Bar

You can examine the menu bar for a window either from the Class Workshop or from the Window Workshop, by choosing the File > Menu… command.

Figure 9-5 illustrates a menu bar as displayed in the Menu Workshop:

Figure 9-5 Menu Bar in the Menu Workshop



```
New Submenu
    New Command
    New Submenu
        New Command
    New Toggle
    New MenuList Item
    ------------
```

Select item to modify it, delete it, or to add another item.

In the Menu Workshop, the menu bar definition is displayed in an outline field in the form of a hierarchy. The entries at the first level of the hierarchy are the pull-down menus on the menu bar. The entries at the second level represent the menu items on the pull-down menus. These items include menu buttons, menu toggles, menu lists, menu separators, and slide-off menus. The third level represents the items on the slide-off menus, which, again, can include any of the simple menu items or other slide-off menus. The slide-off menus may be nested up to any level.

You can browse through this menu hierarchy the same way you browse through any other outline field.

Icons in the outline field The outline field displays kind icons indicating the type of each menu widget in the hierarchy. These icons are miniature versions of the picture buttons you use to create the menu widgets, as shown below.

Icon	Menu Widget
▭	Submenu
▢	Menu Command
☑	Menu Toggle
◉	Menu List Item
—	Menu Separator

Attribute name and short cut The outline field also displays the attribute name for each menu item and, for menu commands that have them, the short cut key assignments.

Opening and closing submenus For a submenu, iPlanet UDS uses the expansion arrow to indicate whether the submenu has menu items on it. A right-pointing expansion arrow indicates that the submenu has menu items on it that are not being displayed. To open the submenu, click once on the expansion arrow. A downward-pointing expansion arrow indicates that the menu items are being displayed. To close the submenu, click once on the expansion arrow.

Viewing the Widget Properties

The properties panel at the bottom of the workshop displays the appropriate properties for the currently selected widget. If the widget is not already selected, simply click on it to view its properties.

Popup menus If the window's definition includes popup menus, these will be displayed as part of the menu bar (even though they will not be displayed on the menu bar when the window is running). To see whether or not a menu item is on the menu bar, is a popup menu, or is both on the menu bar and a popup menu, select the menu item and choose the Item > Popup Usage command. The submenu for the Popup Usage command indicates the current setting for the selected menu item.

Status text To examine the status-line help text associated with a menu item, select the menu item and choose the Item > Status Text command. This command opens the Status Text dialog, which displays the menu item's current status-line help text.

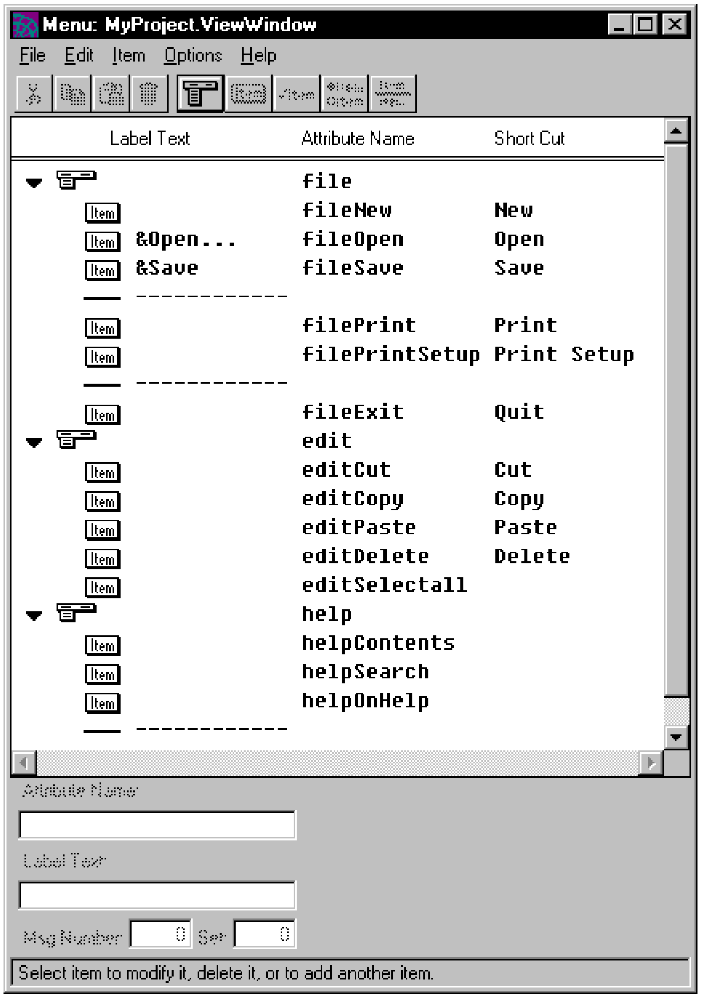
Creating a Menu Bar

You create a new menu for a window by simply opening the Menu Workshop. From the Class or Window Workshop, simply choose the File > Menu… command. From the Class Workshop, you can open the Menu Workshop by single-clicking the Menu Workshop tool.

Once in the Menu Workshop, you have the choice of using the Create Default command to start with a default menu or simply creating your own menu bar from scratch.

When you choose the File > Create Default command to create a default menu bar, iPlanet UDS creates a menu bar with two pull-down menus.

Figure 9-6 Menus Created by Create Default Command



Menu: MyProject.ViewWindow

File Edit Item Options Help

```
          Label Text          Attribute Name   Short Cut
                              file
                              fileNew          New
          &Open...            fileOpen         Open
          &Save               fileSave         Save
          ------------
                              filePrint        Print
                              filePrintSetup   Print Setup
          ------------
                              fileExit         Quit
                              edit
                              editCut          Cut
                              editCopy         Copy
                              editPaste        Paste
                              editDelete       Delete
                              editSelectall
                              help
                              helpContents
                              helpSearch
                              helpOnHelp
          ------------
```

Attribute Name

Label Text

Msg Number 0 Set 0

Select item to modify it, delete it, or to add another item.

The menu commands on the File and Edit pull-down menus are iPlanet UDS prefabricated commands (as described under "About Menu Commands" on page 535), with the exception of New..., Open..., and Save, which are simply labeled menu commands.

Pull-down menus at the top level To create a menu bar, you build a hierarchy in the outline field that represents the pull-down menus on the menu bar. The entries at the first level of the hierarchy are always submenus, which become the pull-down menus on the menu bar. For example, in the default menu, "File" is a submenu that contains the four menu commands. Only submenus are allowed at the first level of the hierarchy.

The entries at the second level become the menu items on the pull-down menus. These include menu buttons, menu toggles, menu lists, menu separators, and submenus. Adding a submenu at this level creates a slide-off menu.

The third level represents the items on the slide-off menus, which can include any of the simple menu items or other slide-off menus. You can nest slide-off menus up to any level (although we do recommend that you keep your menus as simple as possible).

Adding Menu Widgets

To build the menu bar, you add menu widgets one by one to the hierarchy in the outline field. After you add a new item, it is currently selected. You can then move to the properties panel to set its properties.

By default, the new item you add to the hierarchy is inserted *after* the item you select. However, if you wish to add new items *before* the item you select, you can choose the Options > Insert Before command. This command switches you from Insert After mode to Insert Before mode. To change back to Insert After mode, you can choose the Options > Insert After command.

➤ **To add a new menu widget to the menu bar**

1. In the outline field, click in menu item that you want the new menu item to follow or precede (depending on whether you are using Insert After or Insert Before mode).
2. Click the appropriate menu widget button on the toolbar.
3. Set the properties for the menu item on the properties panel.

iPlanet UDS inserts the new menu item at the same level as the selected menu item. If you insert a submenu, iPlanet UDS displays an "(empty menu)" entry one level below the submenu. This allows you to insert menu items at the appropriate level for the submenu.

Adding list items The procedure you follow to create a menu list in the Menu Workshop is different than the procedure you follow in the Window Workshop. Rather than dropping the entire list widget onto the menu bar, you must add each list element individually. Start by inserting each list element to the hierarchy one at a time. Then, to unite the individual elements into a single menu list, give them the same attribute name. (Note that if you change the attribute name of any of the elements in the list, that element will no longer be part of the list widget.)

After adding the new menu items, you may wish to change them by moving them, deleting them, or editing their properties. This is described under "Editing a Menu Bar" on page 555.

Although iPlanet UDS automatically creates the widget attributes for the menu items you have added to the menu bar when you close the Menu Workshop, you may wish to create these attributes while you are still working on the menus. See "Editing a Menu Bar" on page 555 for information on this.

Setting Widget Properties

You can set the widget properties for the currently selected widget on the properties panel. If the widget is not already selected, select it by clicking on it.

Each type of widget has a different set of properties. The properties that are unavailable for the particular widget will either be invisible or inactive. For information about the properties you can set for an individual widget, see "About Menus" on page 527.

Note that the changes you make to the widget properties will not be reflected in the Menu browser until you select a different property or another menu item.

Setting Menu Widget States

To set the starting state for a menu item, you must use a combination of the Item > For Usage and Starting State commands.

➤ **To set a menu item's state**

1. Select the menu item whose starting state you wish to change.
2. Choose the Item > For Usage command, and select the window usage for which you wish to specify the menu item's starting state.
3. Choose the Item > Starting State command, and select the starting state for the menu item.

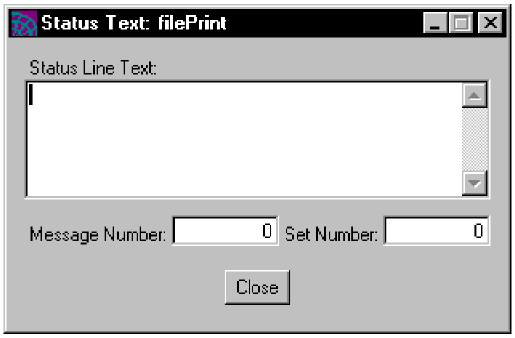
Setting Status Line Help Text

To set status line help text for a menu item, you must use the Item > Status Text command.

➤ **To enter status line text**

1. Select the menu item.
2. Choose the Item > Status Text command.

   The Status Text dialog opens.



3. Enter the text you wish to appear in the status line and click OK.

Note that if you plan to use this menu in a multilingual application, you need to enter a message number and set number. See the *iPlanet UDS Programming Guide* for information on internationalizing applications.

Compiling the Menu Bar

Normally iPlanet UDS does not create the attributes for the menu items you add to the menu bar until you close the Menu Workshop. However, if you wish to create them while you are still working on the menu bar, you can use the File > Compile command. (The File > Save All command also creates the attributes.)

Note that you are not required to use the Compile command. The Compile command is provided for your convenience, so you can keep the list of attributes in the Class Workshop synchronized with the attributes you are adding to the window class when you add menu items to the menu bar.

Editing a Menu Bar

To edit the menu bar, you can open the Menu Workshop either from the Class Workshop or the Window Workshop, using the File > Menu… command. In the Class Workshop, you also can open the Menu Workshop by single-clicking the Menu Workshop tool.

Besides adding new menu items to the menu bar, you can modify the menu bar by:

- moving menu items
- changing submenu levels
- using edit commands to cut, copy, paste, and delete menu items
- modifying an individual menu item's properties

Moving Menu Items

You can move any menu item (including a submenu) to another position by dragging the item to a new position. If you move a submenu, iPlanet UDS moves the menu widgets on the submenu.

➤ **To move a menu item**

1. Select the menu item's icon.
2. Drag the row into position.

iPlanet UDS inserts the row you are moving either before or after the row on which you drop it (depending on whether you are using Insert Before or Insert After modes). The moved menu item will be at the same level as the menu item on which it was dropped.

Changing Submenu Levels

If you wish to change the level of a submenu in the hierarchy, you can use the Item > Promote Submenu and Demote Submenu commands on the Item menu to move the entire submenu up or down one level.

➤ **To change the level of a submenu**

1. Select the submenu title.
2. Choose the Item > Promote Submenu or Demote Submenu command as appropriate.

Cut, Copy, Paste, and Delete

The Cut, Copy, Paste, and Delete commands on the Edit menu allow you to manipulate the items the same way you can cut, copy, paste, and delete text in a text editor.

➤ **To use the Cut, Copy, and Delete commands**

1. Select the menu item.
2. Choose the Edit > Cut, Copy or Delete command.

➤ **To use the Paste command**

1. Select the menu item that the copied or cut menu item should precede or follow (depending on whether you are using Insert Before or Insert After modes).
2. Choose the Edit > Paste command.

Modifying Properties

You can change any of the menu item's properties by editing the settings on the properties panel.

➤ **To modify a menu item's properties**

1. Select the item.
2. Make any appropriate changes to the menu item's properties.

The changes will not take effect until you select another property or another menu item.

Importing and Exporting a Menu Bar

To allow you to copy a menu bar from one window class to another, the Menu Workshop provides Import… and Export… commands. These commands let you export a menu bar from one class into a file and then import the menu bar stored in that file into another class.

The Export… command in the Menu Workshop allows you to export a menu bar to a portable file, which you can then import into another iPlanet UDS window class. The Import… command in the Menu Workshop replaces the menu bar in the window class with the menu bar in the specified text file.

Importing a Menu Bar

The Import… command replaces the current menu bar with the menu bar in the file you specify.

The menu bar that you import must have been created by the Export… command in the Menu Workshop.

➤ **To import a menu bar**

1. Choose the File > Import… command.
2. Choose a menu bar file (a file with an .fsm suffix) to load.

After the menu bar has been successfully imported, iPlanet UDS displays the menu bar in the Window Workshop. If there is an error, such as a bad file, iPlanet UDS displays an error message.

Exporting a Menu Bar

The Export… command writes the definition of the current menu bar into a portable file. iPlanet UDS stores the menu bar in a file with the name you specify and an .fsm suffix.

➤ **To export a menu bar**

1. Choose the File > Export… command.
2. In the file selection dialog, specify the name of the file to contain the menu bar definition. If you give the name of an existing file, the Export… command overwrites the file.

While the menu bar is being exported, iPlanet UDS displays a message indicating that the menu bar is being written to the specified file and prevents all input until the export is complete.

Testing a Menu

To test the menu, you must test the window associated with the menu. To test a window, open the window using the Window Workshop and switch into testing mode. See "Testing a Window" on page 427 for information.

Creating Popup Menus

To create a popup menu in the Menu Workshop, you must create an ordinary submenu. In fact, you must place the popup menu on the menu bar, even though you do not want it to be displayed there.

Unlike menu bar menus, popup menus do not have titles. When you create a new popup menu in the Menu Workshop, you still create a menu title. However, the menu title is not displayed when the popup menu is displayed.

After creating the submenu itself, you should create all the menu commands, menu lists, menu separators, or slide-off menus that you wish to include on the popup menu.

Finally, you must use the Item > Popup Usage command. By default, any submenu you add to a menu bar has the same popup usage setting as its parent menu. And when top-level menu widgets are set to their default, they display only on the menu bar. Therefore, by default, *all* submenus on the menu bar are displayed only on the menu bar, and are not displayed as popup menus.

To change the setting for an individual submenu, select the submenu and choose either the Item > Popup Usage > Popup Only or Popup Usage > Both commands. You can simply keep the default popup usage settings for the individual items on the submenu, because they will inherit the setting you specify for their parent.

The following table describes the Popup Usage commands:

Popup Usage Value	Description
Default	Same as Inherit.
Menu Only	The submenu or menu item is displayed on the menu bar, and is *not* available for use as a popup menu.
Popup Only	The submenu or menu item is *not* displayed on the menu bar, and is available for use as a popup menu.
Both	The submenu or menu item is displayed on the menu bar, and is also available for use as a popup menu.
Inherit	The submenu or menu item inherits the setting of its parent submenu.When top-level menu widgets are set to Inherit, they use the Menu Bar Only setting.

You do not always need to create a new submenu. If you wish to use an existing submenu as a popup menu, you can simply use the Item > Popup Usage > Popup Only or Popup Usage > Both command to make the submenu into a popup menu.

Set PopupMenu attribute or use PopupRequest event Once the popup menu is created, you must use TOOL code to display it. You can either assign it to a field widget so that iPlanet UDS automatically displays it when the end user activates it or you can explicitly display the popup yourself.

To assign the popup to a field widget, you use the field widget's PopupMenu attribute. When the submenu is assigned to the PopupMenu attribute, iPlanet UDS automatically displays the popup menu. See the Display Library online Help for information about the PopupMenu attribute.

To explicitly display the popup, you can handle the PopupRequest event on the window, and use the Window.ShowPopupMenu method to display the popup menu when and where you wish. See the Display Library online Help for information on the PopupRequest event and ShowPopupMenu method.

The Popup Usage command is available for all the menu items that you create in the Menu Workshop, not just for submenus. Using the Popup Usage command for individual menu items enables you to specify which individual menu items within a given submenu will appear on the popup menu and which will not. For example, you could create a submenu that is both a menu-bar menu and a popup menu. The individual items on the menu could include some menu items that show only on the menu bar, some that show only on the popup, and some that show both on the menu bar and the popup.

Remember, however, that the Popup Usage setting for the submenu itself determines whether or not it is displayed as a popup menu. A submenu whose Popup Usage setting is Menu Bar Only will never be displayed as a popup menu, even if some menu items on it are set to Both or Popup Only. Likewise, a submenu whose Popup Usage setting is Popup Only will never be display on the menu bar, even if some menu items on it are set to Both or Popup Only.

➤ **To create a popup menu in the Menu Workshop**

1. Create a submenu, and add the appropriate menu items to it.

   If an existing submenu on the menu bar already exists, you can use that.

2. Select the submenu, and choose the Item > Popup Usage >Popup Only or Popup Usage > Both command.

3. In your TOOL code, assign the popup menu to a field widget by setting the field widget's PopupMenu attribute, and iPlanet UDS will automatically display the popup. Or, use the ShowPopupMenu method to explicitly display the popup in response to a PopupRequest event.

Designing a popup iPlanet UDS displays a popup menu relative to the cursor or pointer hot spot. The position of the popup menu relative to the hot spot depends on the size of the menu and the amount of space available on the screen. If the popup menu can fit to the right of the hot spot without running over the right edge of screen, iPlanet UDS displays it there. If not, iPlanet UDS displays the menu to the left of hot spot. If the entire menu can fit below the hot spot, iPlanet UDS displays it there. If it cannot, iPlanet UDS moves the menu up to display the entire menu. The submenu label is never displayed. Keep these limitations in mind when designing your popup menus.

Further guidelines are:

- keep the menu as short as possible
- use separators to group commands
- use standard ordering for commands the end user is accustomed to seeing on the menu bar (for example, File or Edit menu commands)

For information about creating popup menus dynamically, see the PopupUsage attribute section in the Display Library online Help.

Using the Prefabricated Help Commands

The Menu Workshop provides a prefabricated Help menu and four prefabricated Help commands. Using the prefabricated Help menu ensures that your Help menu is portable—iPlanet UDS uses the title and position on the menu bar that is customary on each particular window system. Some platforms, such as Windows 95, do not use these help menu commands in their standard online help. To accommodate heterogeneous environments, iPlanet UDS will display the menu items on the appropriate platforms, and hide them otherwise.

The iPlanet UDS Help commands that you can include on the Help menu are:

Help Command	Description
Help Contents	Displays the Help Contents window, using the default help file.
Help Search	Displays the Help Search window, using the default help file.
Help on Help	Displays the standard Help on Help window, provided by Windows.
About	Provides an About command in the appropriate location. You must register for the AboutMenuActivate event and provide processing to display the appropriate window.

Of course, if you wish to provide your own help facilities, you can override the default behavior of any of these commands by registering for their Activate events. When you explicitly handle the Activate event on the Help Contents, Help Search, and Help on Help commands, iPlanet UDS automatic help facilities are ignored. If you wish to interact directly with Window Help, you can use the iPlanet UDS WinHelp method defined on the WindowSystem class. See the *iPlanet UDS Programming Guide* for complete information on implementing help in your application.

To use iPlanet UDS automatic help facilities on your window, there are four basic steps you must follow.

➤ **To include a Help menu on a window**

1. Create one Windows Help document for the application, or one for each window in the application.

   The Help document should be in the appropriate format, and should include the information needed by the Help Contents and Help Search commands. See *iPlanet UDS Programming Guide* for information.

2. Add the Help submenu to the menu bar for each window where you wish to provide help.

   iPlanet UDS positions the Help menu in the appropriate place for the window system, and provides the correct title.

3. Add the Help Contents, Help Search, and Help on Help prefabricated commands to the Help menu.

   You can also include the About command, but must take extra steps after doing so (described under "Implementing the About Command" on page 562).

4. Set the DefaultHelpFile attribute for the partition or for each of the Window objects to the help file you wish to use.

Implementing the About Command

The About command does not provide automatic behavior, but including this command on your Help menu ensures that it will be portable. When you include the prefabricated About command anywhere on your menu bar, iPlanet UDS uses whatever position on the menu bar is customary on the particular window system.

AboutMenuActivate event iPlanet UDS also provides a special AboutMenuActivate event, which allows you to detect when the end user chooses the About command. To display information to the end user in response to his choosing the About command, you must register for the AboutMenuActivate event and provide processing to open the appropriate window.

AppTitle attribute The label for the About command consists of two parts, the word "About" and the application title. You must specify the application title by using the AppTitle attribute on the Partition object.

➤ **To use the About command**

1. Include the About command on the Help menu.
2. Set the AppTitle attribute on the Partition object to the application title you wish to have displayed in the About command menu item.
3. Create an About window for the application.
4. Register for the AboutMenuActivate event on the About menu item and provide processing to display the About window you created in the previous step.

Creating OLE Menu Groups

By default, OLE menus are not merged with TOOL menus when the end user is editing the embedded OLE object in an OLE field.

To request that an iPlanet UDS submenu be merged with the OLE menus, you can use the Item > OLE Menu Group command to select one of the following options:

Option	Description
Invisible	Menu is not displayed when the embedded OLE object is edited in place. This is the default value.
File Group	Menu is displayed before the File menu of the application that edits the embedded OLE object. If there is no File menu in the editing application, this menu is displayed before any iPlanet UDS View Group menus.
View Group	Menu is displayed after the File menu and before the View menu of the application that edits the embedded OLE object. If there is no View menu in the editing application, the menu is displayed after the application's File menu and before any iPlanet UDS Window Group menus.
Window Group	Menu is displayed after the View menu and before the Window menu of the application that edits the embedded OLE object. If there is no Window menu in the editing application, the menu is displayed after the application's View menu.

➤ **To use the OLE Menu Group command**

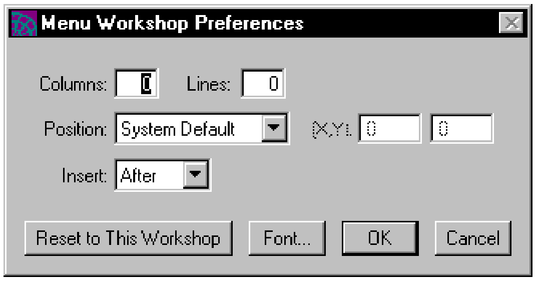
1. In the Menu Workshop, select the submenu you wish to merge with the OLE menus.
2. Choose the Item > OLE Menu Group command.
3. On the OLE Menu Group submenu, choose the appropriate option: File Group, View Group, or Window Group.

Setting Workshop Preferences

The Menu Workshop allows you to set preferences that are saved as part of your current workspace. The preferences that you set take effect for the current Menu Workshop and all Menu Workshops that are opened in the future from the same workspace. However, if any other Menu Workshops are already open, these will not be changed.

To set the workshop preferences, give the Workshop Preferences… command on the File menu. This command opens the Menu Workshop Preferences dialog, where you can set any number of preferences.

Figure 9-7 Menu Workshop Preferences Dialog


Menu Workshop Preferences

Columns: 0 Lines: 0

Position: System Default (X,Y): 0 0

Insert: After

The preferences you can set for the Menu Workshop fall into the following general categories:

- workshop window size and position
- insert preference
- font preference

The workshop window size and position and font preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Menu Workshop.

Insert Preference

The Insert preference specifies whether new items are added to the menu hierarchy before or after the selected item.

By default, any new items you add to the menu hierarchy in the Menu Workshop are inserted *after* the item you select. However, if you wish to add new items *before* the item you select, you can choose the Insert Before preference on the Menu Workshop Preferences dialog. This option switches the Menu Workshop from Insert After mode to Insert Before mode.

To change back to Insert After mode, you can either choose the Insert After preference on the Menu Workshop Preferences dialog, or you can use the Options > Insert After command.

Chapter 10

Using the Method Workshop

This chapter provides background information about methods and describes how to use the Method Workshop.

In this chapter, you will learn how to:

- examine a method
- create a method
- write the method source
- set method breakpoints
- edit a method
- import and export a method
- compile a method
- set code preferences

About Methods

A method is a procedure that is written specially for a particular class. A method for a given class can be invoked only on objects of that class or of its subclasses.

All methods consist of a name and the TOOL source code that is executed when the method is invoked. Most methods also include parameters, which allow the caller to pass information to and from the object that the method is manipulating.

Method Components

A method consists of the following components:

Component	Description
name	The name the caller will use to invoke the method.
parameters	Used by the caller to provide input values to the method, to hold output values from the method, or both.
return type	The data type of one value that will be passed back to the caller by a `return` statement in the method.
return event	An event that is posted when the method is invoked with the `start task` statement and the task completes successfully.
exception event	An event that is posted when the method is invoked with the `start task` statement and the task is terminated due to an unhandled exception.
method source	A list of TOOL statements, with an optional exception clause, that provides the processing for the method.

A method's *signature* consists of the method name, parameters, return type, return event, and exception event. In other words, the method signature defines the "interface" to the method.

The following sections provide detailed information about these components.

Method Name

The method name is the name the caller must use to invoke the method. The name you select for the method is very important because if you use the name of an inherited method, you will be overriding the inherited method. If you do not wish to override the inherited method, you should not use the existing name. See "Overriding Methods" on page 571 for information about overriding.

Parameters

A method may have any number of parameters. Parameters provide input values for the method, hold output values, or both.

Parameter name and type Every parameter has a name and a type. The name identifies the parameter and is for use when the method is invoked, so that callers can specify parameters by name. The type specifies the type of value allowed for the parameter. The parameter type can be any simple data type or any class.

Parameter mechanism By default, a parameter is for input only. An input only parameter means that the caller specifies a value for the parameter when invoking the method and the method uses that value for processing.

However, you can specify that a parameter is for both input and output, or for output only. An output parameter specifies a variable or attribute to hold a result value from the method. After the method completes, the attribute or variable stores the output value, which the caller can then use for processing. When a parameter is declared for input and output, the attribute or variable specifies an input value when the method is invoked, and holds the output value when the method completes. When a parameter is declared for output only, the value of the variable or attribute has no effect when the method is invoked.

Class type parameters When the type of a parameter is a class, iPlanet UDS passes a reference to the object, not the object itself. A class parameter means that even if the parameter is for input only, if the method makes changes to the object, these changes are reflected when you return from the method. The object is changed because both the invoking method and the invoked method are referencing the same object.

Copy option The copy option allows you to prevent the calling and called method from referencing the same object. When you specify copy for a class parameter, iPlanet UDS makes a copy of the object and passes a reference to that copy as the parameter. For an output parameter, iPlanet UDS returns a reference to the copy. The copy option is also useful for improving the efficiency of distributed applications. If the invoking method and the invoked method are going to be on separate partitions, using a copy of the object rather than sharing a single object may reduce communication costs.

Default value Normally, the parameters are required when the caller invokes the method. However, if you wish to make a parameter optional, you can specify a default value for the parameter. iPlanet UDS uses this value for the parameter when the caller leaves this parameter off the parameter list. The default value for the parameter must be compatible with the parameter's data type. For parameters with a class type, the default value must be NIL, which means "no object."

Return Type

The method's return type specifies the data type of the value that you will pass back from the method with the `return` statement. The return value can be any simple data type or any class. If you do not specify a return type, you cannot use the `return` statement in the method to pass a value back to the caller. The return type for the method has a copy option, which works exactly like the copy option for a parameter, as described above.

Return Event

If you plan to use the `start task` statement to invoke the method as a separate task and you want to receive notification when the task completes, you must define a return event for the method. When you invoke the method with the `start task` statement and you request the return event with the `completion=event` clause, the method automatically produces a return event when it completes. The return event uses the output and input-output parameters defined for the method. And if the method is defined as having a return type, the last parameter for the event is a return value called "return." See the *TOOL Reference Guide* for information on the `start task` statement and the return event.

Exception Event

If you plan to use the `start task` statement to invoke the method as a separate task and you want to receive notification when the task is terminated due to an unhandled exception, you must define an exception event for the method. When you invoke the method with the `start task` statement and you request the exception event with the `completion=event` clause, the method automatically produces an exception event when it is terminated. The exception event has one parameter called "exception," which contains the exception that terminated the task. See the *TOOL Reference Guide* for information on the `start task` statement and the exception event.

Method Source

The source of the method is where you provide code that operates on the object, invokes methods on other objects, and so on. To write the method, you use TOOL, the iPlanet UDS object-oriented programming language. See the *TOOL Reference Guide* for complete information on the language.

The method source consists of two parts, the statement block and the optional exception handler.

Method statement block The method statement block consists of TOOL statements and comments. Within an individual method, you can provide many different kinds of processing, such as declaring variables, handling events, starting transactions, initiating new tasks, accessing the database, and so on.

return statement If the method has a return type, you must use the `return` statement in the method to exit from the method and return a value to the invoking method. See the *TOOL Reference Guide* for information on the `return` statement.

Exception handler After the statement block, the optional `exception` clause provides exception handling for the method as a whole. If you use it, the exception handler for the method must be at the end of the method. See the *TOOL Reference Guide* for information on using exception handlers.

Statement breakpoints When you are entering your method source code in the Method Workshop, you can set statement breakpoints in the code for use with the Debugger. The breakpoints that you set in the Method Workshop take effect in the Debugger every time the method is invoked by the application.

Statement breakpoints added in the Method Workshop are persistent. Persistent breakpoints stay in effect until you remove them from the method. You can remove the persistent breakpoints either from the Method Workshop or from the Debugger.

For information about setting breakpoints in your method source code, see "Setting Breakpoints" on page 582.

Overriding Methods

iPlanet UDS allows you to override an inherited method. Overriding a method means creating a method for the current class that has the same name and parameters as a method it inherited from one of its superclasses. By providing different source code for an inherited method, you are creating a different "version" of the method for objects of the current class. When you invoke the method on an object of the current class, iPlanet UDS uses the method you defined specifically for the class rather than the inherited method.

For example, the Stream class in the Framework library defines a Close method that closes a stream. The File class, a subclass of the Stream class, overrides the Close method defined for the Stream class to close a *file* rather than a *stream*. When you invoke the Close method on a Stream object, iPlanet UDS uses the Close method implemented for the Stream class. However, because the File class overrides the Close method, when you invoke the Close method on a File object, iPlanet UDS uses the Close method implemented for the File class.

Polymorphism Using overriding for inherited methods provides the benefit of polymorphism. *Polymorphism* means the ability of a data item to refer at runtime to any number of different objects.

Declared type and runtime type In TOOL, we make the distinction between the declared type and the runtime type. When you declare an item whose type is a given class, the class that defines the data item is its *declared type*. However, when you assign an object value to the data item, the *runtime type* of the object assigned to the data item can be any subclass of the class that defines the data item. In other words, the data item can take any number of forms (the term "polymorphism" is literally defined as the ability to assume many different forms).

The benefit here is that you can perform an operation on a data item without knowing anything about how the object's class has implemented the operation. Because the subclasses override the method to customize it for the particular class, you are assured that the operation will work correctly for the object. For example, invoking the Close method works on a data item whose declared type is Stream will always work, even if the runtime type is a subclass of the Stream class, because each of the subclasses of Stream override the Close method.

Overriding overloaded methods Note that if you override an inherited method that is overloaded (described next), your class will not be able to access any of the inherited methods with that name. Therefore, if your class needs the entire set of overloaded methods, you must create the entire set again for your class.

Using drag and drop to copy method code When you want to create a method in a subclass that overrides a method defined in a superclass, you can drag the method from the Class Workshop for the superclass to the Class Workshop for the subclass. The advantage of this technique is that the correct method signature and method source code are automatically copied to the subclass.

➤ **To override a method**

1. In the Class Workshop for the subclass, choose the File > Open Superclass command to locate the superclass where the method is defined.
2. Drag-and-drop the method signature from the superclass to your subclass.

   This will create a method in the subclass with the correct name, parameters, and return type.
3. Open the method in the subclass and, using the Method Workshop, edit the source code appropriately.

Overloading Methods

iPlanet UDS also allows you to overload methods. Overloading a method means that more than one method in a class can share the same name, although their parameter lists must differ. Using overloading lets you have different versions of the same method, with different parameter types. In this case, when the method is invoked for the object, iPlanet UDS uses the parameter data types as well as the method name to determine which code to execute. For example, the iPlanet UDS Framework library provides two Add methods for the DateTime class. The first version of the Add method provides two parameters so you can add an IntervalData value to a DateTimeData value and replace the original DateTimeData value with the result. The second version of the Add method has one parameter so you can add an IntervalData value to the original DateTimeData value.

Method signatures When a method is overloaded, each method that shares the same name has its own method signature. A *method signature* is the method name and the parameter list (also the return type, return event, and exception event if there is one). For example, here are two method signatures for the RequestFocus method:

```
RequestFocus( )
RequestFocus(row:TemplateField, scrollPolicy:integer)
```

When you are working with an overloaded method in the Method Workshop, you work with the individual method signatures associated with the overloaded method as *separate* methods. For example, to update an overloaded method, in the Class Workshop, you must open each method signature individually. Each time you open a particular method signature, a new instance of the Method Workshop opens, just for that signature. Once in the Method Workshop, you can edit the method the same way you would edit a method that is not overloaded.

Converter Methods

A *converter* method is a special kind of method that you create to adjust for the differences in a class element between two (or more) versions of a class. When you are performing class version upgrade of an iPlanet UDS application (described under "Class Versions" on page 289), you must write converter methods. The purpose of converters is to enable code that calls or expects one version of a class (by invoking a method or posting an event, for example) to actually use a different version of the class. A converter allows older and newer code to communicate by "bridging" the differences between two versions of a method or an event. The *iPlanet UDS Programming Guide* provides complete information on converter methods.

In the Method Workshop, you work with converter methods as individual methods. For example, to update a converter method, you double-click the individual converter name in the Class Workshop, which opens a new instance of the Method Workshop. Once in the Method Workshop, you can then edit the converter method the same way you would edit any method. See "Examining Methods" on page 298 for information on examining converter methods and "Creating a Converter Method" on page 318 for information on creating converter methods.

CAUTION You should *not* use converters without a thorough understanding of the iPlanet UDS upgrading features. Upgrading an iPlanet UDS production environment requires very careful planning. See the *iPlanet UDS Programming Guide* for complete information.

Method Visibility

By default, the methods in a class are public, which means that any other classes in the project, or classes that use this project as a supplier, can access them. iPlanet UDS provides the option of defining any method as "private." A private method can be accessed only by the class that defines it, not by any other classes (even its own subclasses). Only methods in the current class can invoke a private method.

Using the Method Workshop

You enter the Method Workshop from the Class Workshop either by opening an existing method or by creating a new method.

Opening an existing method If you wish to examine or edit an existing method, double-click the method name, or click the method name and choose the Element > Open command. If the method is overloaded, you must open the particular method signature you wish to examine. See "Examining Methods" on page 298 for information about using the Class Workshop to open the Method Workshop for an existing method.

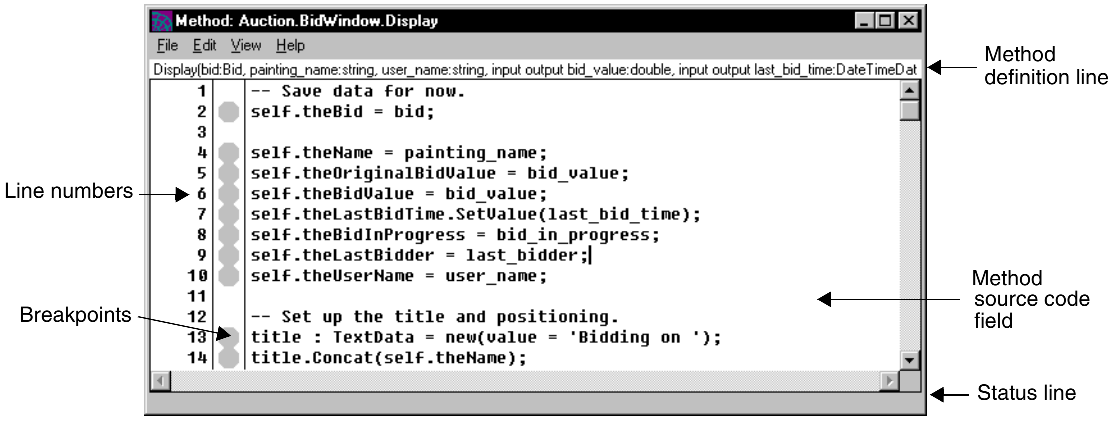
Creating a new method If you wish to create a new method, choose the Element > New Method command or single-click the New Method icon.

The Method Workshop Window

The Method Workshop window consists of three parts: the method definition line, the method source code field, and the status line.

Figure 10-1 illustrates the Method Workshop.

Figure 10-1 Method Workshop



```
Display(bid:Bid, painting_name:string, user_name:string, input output bid_value:double, input output last_bid_time:DateTimeDat
```

```
 1  -- Save data for now.
 2  self.theBid = bid;
 3
 4  self.theName = painting_name;
 5  self.theOriginalBidValue = bid_value;
 6  self.theBidValue = bid_value;
 7  self.theLastBidTime.SetValue(last_bid_time);
 8  self.theBidInProgress = bid_in_progress;
 9  self.theLastBidder = last_bidder;
10  self.theUserName = user_name;
11
12  -- Set up the title and positioning.
13  title : TextData = new(value = 'Bidding on ');
14  title.Concat(self.theName);
```

View menu Note that the View menu in the Method Workshop provides commands that let you specify whether to display the method definition line and the status line. The method source field is always displayed, but you can turn the definition and status lines on or off as you choose.

Access to Other Workshops

From the Method Workshop, you can access one other workshop:

Workshop	How to access it
Class Workshop	The File > Open Class… command opens the Class Workshop to display the definition of the class to which the current method belongs. If the class is already being displayed, the Open Class… command moves the input focus to the appropriate Class Workshop window.

Leaving the Method Workshop

To leave the Method Workshop, choose the File > Close command. This command closes the current workshop only.

To leave the Method Workshop and erase all changes you have made since your last Save All command, choose the File > Cancel command. This closes only the current workshop.

Examining a Method

The Method Workshop displays the definition for the current method. If the method you wish to examine is not already displayed, you must return to the Class Workshop to open it. See "Examining Methods" on page 298 for information about using the Class Workshop to open the Method Workshop for a given method.

Examining Parameters and TOOL Source

The Method Workshop displays the method's definition and TOOL source directly on the main Workshop window.

Method definition The top line of the window displays the definition of the method using the syntax for the TOOL `method` statement (see *TOOL Reference Guide* for information on the `method` statement). The method definition shows the method name, the method parameters, including the name, type, and mechanism (if other than the default), and the method return type. The method definition does not show the return and exception events for the method; you must use the Properties… command (described under "Examining Method Properties" on page 577) to view them.

Parameters Note that if the method definition is not currently displayed in the workshop window, you can turn it on with the View > Parameters command. This command lets you control whether or not the method definition is displayed in the current workshop. Your code preferences (see "Setting Code Preferences" on page 590) determine whether the method definition line is automatically displayed when you enter the workshop.

When the method definition is displayed, the parameter list may be too long to be completely displayed. In this case, you can scroll horizontally to view the complete definition.

Note that you can use the text editing commands to copy parameter text from the method definition and paste it into the source code of a different, calling method.

TOOL source code The lower section of the window displays the source code for the method. The TOOL source code field is a multiline text field, which you can scroll to view the entire method. Your code preferences (see "Setting Code Preferences" on page 590) determine whether line numbers and breakpoints are automatically displayed when you enter the workshop.

Viewing line numbers If line numbers are not currently displayed for the source code, you can turn them on with the View > Line Numbers command.

Viewing breakpoints If the breakpoints are not currently displayed for the source code, you can turn them on with the View > Breakpoints command.

Searching for a line If you wish to search for specific lines in the method, you can use the Edit > Go to Line or Find commands (see "Editing a Method" on page 582 for information).

Examining Method Properties

To view the method's properties, choose the File > Properties… command. This command opens the Method Properties dialog, which displays the method's name, parameters, return type, return event, exception event, and visibility.

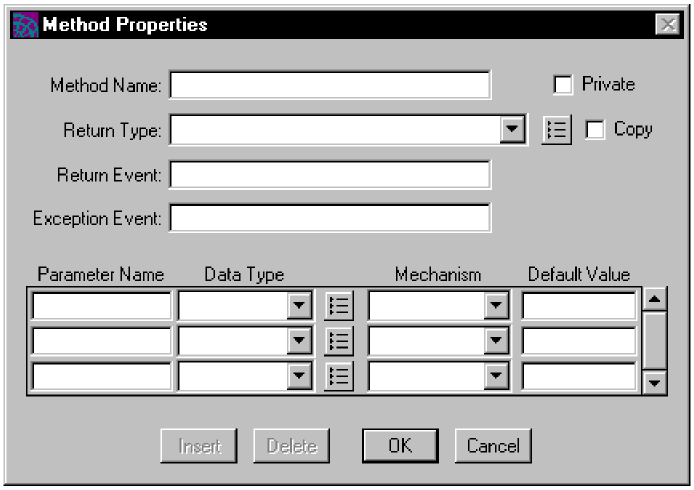
Creating a Method

To create a new method, you must start from the Class Workshop. Unless you already have write access to the class, you must use the Project Workshop to check out or branch the class before you can create a new method for it. See "Write Access to Project Components" on page 257 for information on this.

➤ **To create a new method**

1. In the Class Workshop, choose the Element > New Method command or single-click the New Method tool.

2. The Method Properties dialog appears.



3. In the Method Properties dialog, enter the method's name. The return type, return event, exception event, and parameters are optional.

4. Click the OK button to add the method to the class and open the Method Workshop.

The next section provides information about how to fill in the Method Properties dialog, followed by information about how to enter the method source code.

Converter methods The procedure for creating converter methods is different than the procedure described above (see "Creating a Converter Method" on page 318 for information). However, the section "Writing the Method Source Code" on page 580 applies both to standard methods and converter methods.

Specifying Method Properties

Method Name property The method name is any legal iPlanet UDS name. Unless you wish to override the method, the name must be unique for the public names of its superclasses.

Overriding an inherited method If you specify the name of an inherited method, the new method you are creating will override the inherited method.

Private property By default, a method is public, which means all the other classes in the project have access to the method. The Private toggle lets you set the method's visibility to private. A private method can be accessed only by the class that defines it, not by any of the other classes in the project (not even by any of its subclasses). Creating a private method essentially prevents the method from being inherited.

Return Type property The Return type field lets you specify the type for the return value of the method. This fillin field allows you to choose one of the standard data types (see "Using iPlanet UDS Data Types" on page 132) or type in a class name. The Copy toggle allows you to specify whether or not an object return value should be a copy. If you are overriding a method, you must specify the same return type as the inherited method or leave the field blank.

Return Event property The Return Event field lets you specify a name to use for the method's return event. The return event is the event that is posted when you invoke the method using the `start task` statement with the `completion= event` clause and the method completes successfully. The return event name can be any legal iPlanet UDS name. If you are overriding a method, you must leave the field blank, because only the original method can define the return event.

Exception Event property The Exception Event field lets you specify a name to use for the method's exception event. The exception event is the event that is posted when you invoke the method using the `start task` statement with the `completion= event` clause and the method is terminated due to any exception. The exception event can be any legal iPlanet UDS name. If you are overriding a method, you must leave the field blank, because only the original method can define the exception event.

Method parameters To specify the method's parameters, you fill in an array field that specifies the name, type, mechanism, and optional default value for each parameter. The following table describes how to fill in the array field:

Property	How to fill it in
Parameter Name	The Parameter Name field is a data field. Simply type in any legal iPlanet UDS name.
Type	The Type field is a fillin field that allows you to choose one of the standard data types or type in a class name.
Mechanism	By default, the mechanism for a parameter is input only. If you wish to choose a different mechanism or if you wish to use the copy option for a parameter of a class type, you can choose the appropriate mechanism from the drop list.
Default Value	If you wish to make a parameter optional, you can specify a default value for it. The default value for the parameter must be compatible with the parameter's data type. For parameters with a class data type, the default value must be "NIL," which means "no object." If you leave the default value blank, the parameter is required.

Writing the Method Source Code

To write the method source code, you can either type the TOOL code directly into the source field, or you can write the method in a text file and then use the Import Text command to copy it into the source field. See "Importing and Exporting a Method" on page 588 for information on the Import Text command.

Exception handler for method The method code in Method Workshop is enclosed by an implicit `begin/end` statement (even though you do not see the begin and end key words in the method code). Therefore, if you want to add an exception handler for the method, simply put the exception section at the end of the method. The following example illustrates:

Code Example 10-1 Exception Handler Example

```
ImageStatusMessage.SetValue('Requesting Image.');
PaintingImage = self.ImageManager.GetImage
    (name = self.thePaintingForBid.Name);
exception
  when e : GenericException do
    ImageStatusMessage.SetValue('Image not available.');
    task.ErrorMgr.Clear();
```

Typing TOOL Code

Text editing The source code field is an iPlanet UDS text edit field. To type your method directly into the field, simply position the cursor in the source code field and start typing. To end a line, use the Return character. To indent a new line, use the tab character. In the Method Workshop, the tab character is always set to four character spaces.

Editing shortcut keys The Method Workshop provides text edit shortcut keys, which you can use to select text and move the cursor. In addition, the Edit menu provides editing commands for modifying the method source. See "Editing the TOOL Source Code" on page 583 for information on the shortcut keys and the Edit menu.

Drag and drop names A convenient way to enter text into the source code field is to drag the names of project components or class elements displayed in other workshops and drop them into the source code field. Dragging and dropping project components allows you to quickly copy the names rather than keying them in.

Automatic indenting If you have your code preferences set with automatic indenting turned on, each line that you type in the source code field will automatically be indented with the same number of tabs as the line it follows. If you have your preferences set with automatic indenting turned off, each line that you type will be aligned with the left margin. In this case, you must enter tabs or use spaces to indent the line or, after entering the code, you can use the Indent and Unindent commands (described under "Editing a Method" on page 582) to control the indenting of selected blocks of code. See "Setting Code Preferences" on page 590 for information on automatic indenting.

Setting Breakpoints

The Method Workshop allows you to set statement breakpoints in your method source code. You can set your breakpoints before running the Debugger or while the Debugger is running.

Before you can set any breakpoints, you must be sure that:

- The breakpoints column in the source code field is displayed. If it is not displayed, you can turn it on with the View > Breakpoints command.
- The method is compiled. If the method has not been compiled, you can compile the method with the File > Compile command.

You will know that you can set a breakpoint on a statement if a grey stop sign icon is displayed in the column next to the statement. If the grey breakpoint icons are not displayed, the method needs to be compiled before you can set breakpoints. See "Compiling the Method" on page 589 later in this chapter for information about using the Compile command.

➤ **To set a breakpoint for a statement**

1. Scroll to the statement where you want the breakpoint.
2. Click the grey icon to the left of the statement. A red stop sign icon indicates that the statement now has a breakpoint.

Removing a breakpoint To remove a breakpoint, simply click the stop sign icon. A grey toggle indicates that the statement no longer has a breakpoint. You can remove breakpoints in the Method Workshop at any time, even while the Debugger is running.

Note that you can also remove breakpoints from a method while you are in the Debugger by using the Global Breakpoint Manager (see "Setting Breakpoints" on page 640 for information on the Global Breakpoint Manager).

Editing a Method

To edit a method, you must start from the Class Workshop. Unless you already have write access to the class, you must use the Project Workshop to check out or branch the class before you can edit the method. See "Write Access to Project Components" on page 257 for information on this.

➤ **To edit a method**

1. In the Class Workshop, double-click on the method name, or select the method name and choose the Element > Open command.
2. When the Method Workshop opens, you can edit the method by changing its source code or by changing the method's properties.

Editing the TOOL Source Code

The Method Workshop provides the following text editing shortcuts:

Function	Shortcut on Windows/Motif
Select one character to the left	Shift-Left Arrow
Select one character to the right	Shift-Right Arrow
Select one line down	Shift-Down Arrow
Select one line up	Shift-Up Arrow
Select from the cursor position to the beginning of the current word	Shift-Ctrl-Left Arrow
Select from the cursor position to the end of the current word	Shift-Ctrl-Right Arrow
Select from the cursor position to the beginning of the current line	Shift-Home
Select from the cursor position to the end of the current line	Shift-End
Select from the cursor position to the beginning of the text	Shift-Ctrl-Home
Select from the cursor position to the end of the text	Shift-Ctrl-End
Select all text	Ctrl-A
Move cursor to the beginning of the line	Home
Move cursor to the end of the line	End
Move cursor one word to the left	Ctrl-Left Arrow
Move cursor one word to the right	Ctrl-Right Arrow
Move cursor to the beginning of the text	Ctrl-Home
Move cursor to the end of the text	Ctrl-End

Edit menu The Edit menu in the Method Workshop provides a set of basic commands for editing the method. These commands allow you to use the window system clipboard to cut, copy, and paste text, to search for and replace text, and to change indenting of selected lines.

Text Editing

The Edit menu provides the following commands for editing the text:

Command	Description
Undo	Removes your last edit.
Redo	Makes the edit you removed with the **Undo** command.
Cut	Removes the selected text and copies it into the clipboard.
Copy	Copies the selected text into the clipboard.
Paste	Inserts the current contents of the clipboard into the method.
Delete	Removes the selected text.

The Cut, Copy, and Paste commands use the clipboard provided by your window system.

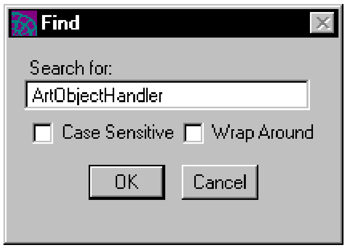
Searching and Replacing

There are two ways to move the cursor to a particular line in your code. You can use the Go to Line… command to go directly to the specified line number or you can use the Find… command to search for a particular string. The Replace… command lets you replace a search string with a new string.

The Go to Line… command prompts you to enter the number of the line to which you wish to move. After you specify the line number, iPlanet UDS moves the cursor to the first character of the line. If your line numbers are not currently displayed, you can use the View > Line Numbers command.

The Edit > Find… command lets you search for a particular string. When you give the Find… command, iPlanet UDS opens the Find dialog, where you specify the search string. The Find… command then searches for the string starting from the current insertion point. When it finds the string, iPlanet UDS highlights the string. Figure 10-2 illustrates the Find dialog:

Figure 10-2 Find Dialog



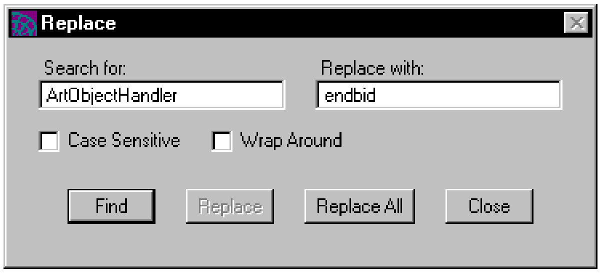
Case Sensitive option By default, the Find… command is case insensitive. If you want to match the case used in the search string, click on the Case Sensitive toggle in the Find dialog.

Wrap Around option By default, the Find… searches from the current insertion point to the end of the method source code. If you wish to search past the end and start from the beginning of the method source code, click on the Wrap Around toggle.

Find Again command If you wish to move to the next occurrence of the string you searched for in the last Find… command without opening the Find dialog, you can simply give the Find Again command. This command uses the search string you specified in your last Find… command. The Find Again command is especially useful with its speed key.

Replace… command The Edit > Replace… command lets you search for a particular string and replace that string with a new string. When you give the Replace… command, iPlanet UDS opens the Replace dialog, where you specify the search and replace strings. You then have the choice of using the Replace button to replace the first occurrence of the search string or the Replace All button to replace all occurrences of the search string within the method. Figure 10-3 shows the Replace dialog:

Figure 10-3 Replace Dialog



Depending on your code preferences, the Replace command works in one of two modes: Find After mode or Find Before mode. (See "Setting Code Preferences" on page 590 for information on setting the Replace modes.)

Find After mode By default, the Method Workshop uses Find After mode. In this mode, the Replace command replaces the currently selected text and then automatically finds the next occurrence of the search string. Find After mode allows you to check the search string in context before you actually make the replacement.

Find Before mode If your code preferences are set to use the Find Before mode, the Replace command searches for the next occurrence of the search string and then automatically performs the replacement. Find Before mode allows you to check the replacement in context before moving on to the next occurrence of the search string.

Replace Again command If you wish to replace the next occurrence of a string without opening the Replace dialog, you can simply give the Replace Again command. This command uses the search and replace strings you specified in your last Replace… command, and performs the replacement according to the current Replace mode. In fact, giving the Replace Again command is the same as clicking the Replace button on the Replace dialog. The Replace Again command is especially useful with its speed key.

Indenting

The indenting commands let you move the currently selected lines of code down or up one level.

The Indent command moves the currently selected lines down one level by inserting a tab character at the start of each selected line.

The Unindent command moves the currently selected lines up one level by deleting a tab character from the start of each selected line.

Cancelling Your Changes

If you ever make a set of edits that you want to erase completely, you can use the File > Cancel command to revert to a previous version of the method. The Cancel command erases all changes you made to the method since you last saved your workspace or, if you have not given a Save All command, since you last opened the method.

Editing the Method’s Properties

To edit the method’s properties, choose the File > Properties… command. This command opens the Method Properties dialog, where you can make any appropriate changes.

Editing the Parameters

The method’s parameters are displayed in the array field in the Method Properties dialog. You can change the existing parameters or add new ones to the end of the list.

Insert button If you wish to add a new parameter to the middle of the list, you can use the Insert button.

➤ **To add a new parameter**

1. Select the parameter above which you wish to add the new parameter.
2. Click the Insert button.
3. Fill in the new row in the array field.

Delete button If you wish to delete a parameter from the list, you can use the Delete button.

➤ **To delete a parameter**

1. Select the parameter you wish to delete.
2. Click the Delete button.

Importing and Exporting a Method

iPlanet UDS lets you write a method or part of a method in an external text file and then import it into the Method Workshop. You can also copy part or all of a method to an external text file for use outside of iPlanet UDS.

Importing TOOL Code

Use the Import Text… command to insert the contents of a text file into the method source. You can then edit the text as desired.

➤ **To import method text**

1. Position the cursor at the point where you wish to insert the imported text.
2. Choose the Edit > Import Text… command.
3. In the File Selection dialog, select the file that you wish to import.

Exporting TOOL Code

Use the Export Text… command to write a method or a section of a method to a text file. This command writes the currently selected text into the file you specify. If you do not select any text, the Export Text… command exports the entire method.

➤ **To export method text**

1. Select the text you wish to export.
2. Choose the Edit > Export Text… command.
3. In the File Selection dialog, specify the file name to which you wish to export the text.

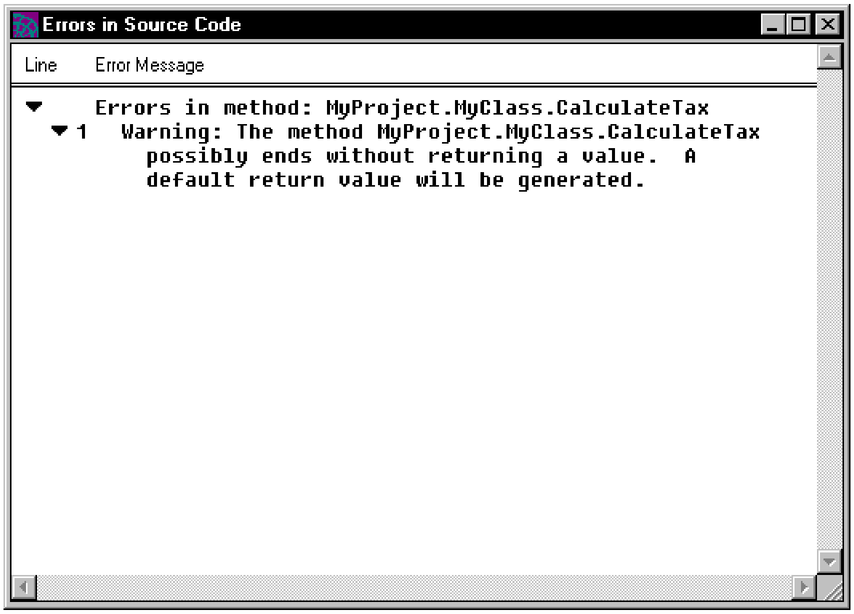
If you specify a new file name, the Export Text… command creates the file. If you specify an existing file, the Export Text… command overwrites the file.

Compiling the Method

After writing the method, you can compile it to check for errors. While it is not necessary to compile your method before you run the project, the Compile command allows you to check for syntax errors without actually executing the code. To compile the method, simply choose the File > Compile command. iPlanet UDS compiles the method and reports any errors in the Error window.

Finding an error The Error window displays the compilation errors messages for the method in an outline field.

Figure 10-4 Error Window



```
▼    Errors in method: MyProject.MyClass.CalculateTax
  ▼ 1  Warning: The method MyProject.MyClass.CalculateTax
         possibly ends without returning a value.  A
         default return value will be generated.
```

You can jump directly from one of these messages to the code that caused the error. In the Error window, double-click the error you wish to find. The Method Workshop will then move the cursor to the line that contains the code that caused the error.

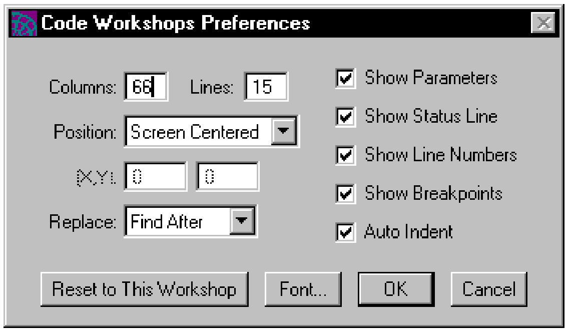
You can keep the Error window open as long as you need it. When you are finished using the window, simply close the window.

Setting Code Preferences

The Method Workshop allows you to set code preferences that are saved as part of your current workspace. The code preferences that you set in the Method Workshop take effect for the current Method Workshop and all Method, Event Handler, and Cursor Workshops that are opened in the future from the same workspace. However if any other Method, Event Handler, or Cursor Workshops are already open, their code preferences will not be changed.

To set the code preferences, choose the File > Workshop Preferences… command. This command opens the Code Workshops Preferences dialog, where you can set any number of preferences.

Figure 10-5 Code Workshops Preferences Dialog



Code Workshops Preferences

Columns: 66 Lines: 15

Position: Screen Centered

(X,Y): 0 0

Replace: Find After

☑ Show Parameters

☑ Show Status Line

☑ Show Line Numbers

☑ Show Breakpoints

☑ Auto Indent

The code preferences fall into the following general categories:

- workshop window size and positions
- replacement mode
- viewing preferences
- automatic indenting
- font preference

The workshop window size and position, viewing, and font preferences are general iPlanet UDS preferences and are described under "Setting Workshop Preferences" on page 136. This section provides information about the preferences specific to the Code Preferences dialog.

Replace Mode

As described under "Editing a Method" on page 582, you can choose one of two modes for the Replace command. The Find Before mode performs the search before the replace. The Find After mode performs the replace on the currently selected text. To change the replace mode for the workspace, simply choose the appropriate mode from the drop list.

Replace Mode	Definition
Find Before	When you give the Replace command, iPlanet UDS moves to the next occurrence of the "Find" string before performing the replacement.
Find After	When you give the Replace command, iPlanet UDS performs the replacement on the currently selected "Find" string and then finds the next occurrence of the string.

Automatic Indenting

By default, the workshop provides automatic indenting for your source code. Each line you type is automatically indented to the same level as the one it follows. If you wish to turn this feature off, set the Auto Indent preference to off. When automatic indenting is off for the workshop, each line you type in the source code field is aligned with the left margin.

Chapter 11

Using the Event Handler Workshop

This chapter provides background information about named event handlers and describes how to use the Event Handler Workshop.

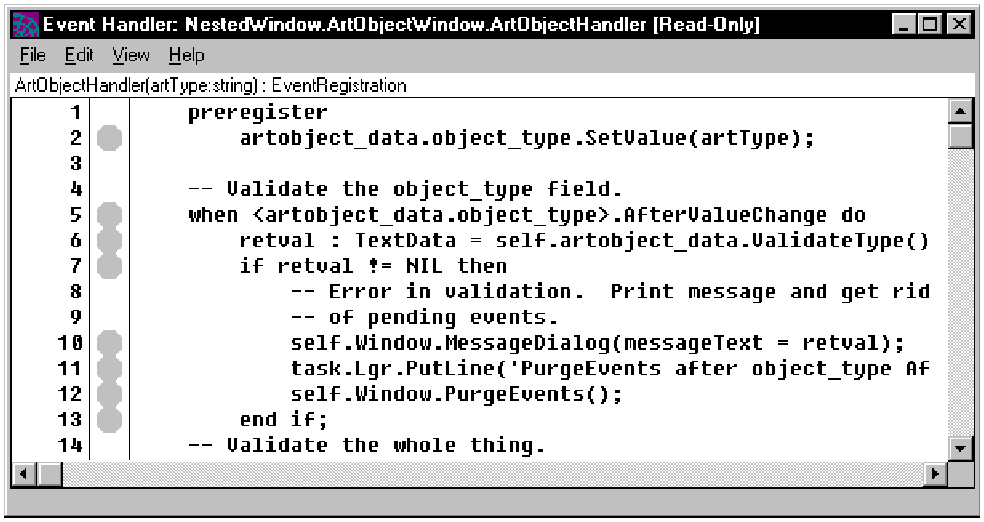
In this chapter, you will learn how to:

- examine an event handler
- create an event handler
- write the event handler source
- edit an event handler
- import and export an event handler
- compile an event handler

About Event Handlers

An event handler is a named block of TOOL code that provides the logic to be executed in response to one or more events. The event handler provides reusable, modular event handling code that you can include in any number of event statements. For example, the following event handler performs field and cross-field validation for some widgets:

Figure 11-1 Example Event Handler



Event Handler: NestedWindow.ArtObjectWindow.ArtObjectHandler [Read-Only]
File Edit View Help
ArtObjectHandler(artType:string) : EventRegistration

```
    preregister
        artobject_data.object_type.SetValue(artType);

    -- Validate the object_type field.
    when <artobject_data.object_type>.AfterValueChange do
        retval : TextData = self.artobject_data.ValidateType()
        if retval != NIL then
            -- Error in validation.  Print message and get rid
            -- of pending events.
            self.Window.MessageDialog(messageText = retval);
            task.Lgr.PutLine('PurgeEvents after object_type Af
            self.Window.PurgeEvents();
        end if;
    -- Validate the whole thing.
```

To see this event handler in context, see the ArtObjectHandler method in the ArtObjectWindow class in the iPlanet UDS example program NestedWindow.

register statement The TOOL `register` statement allows you to include an event handler at any point within an event statement or in another event handler. The following example illustrates using the `register` statement to include the ArtObjectHandler event handler in an `event loop` statement:

Code Example 11-1 Event Handler example

```
event loop

  preregister
    -- Include the ArtObjectWindow's event handler in this
    -- event loop.
  register ArtObjectWindow.ArtObjectHandler
       (artType = 'Sculpture');

  when task.Shutdown do
    exit;
...
end event;
```

Project: NestedWindow • **Class:** SellWindow • **Method:** Display

An event handler is associated with a particular class. Storing named event handlers as part of a class definition allows you to provide event handling code that is inherited by subclasses. This is especially useful when you are creating window classes that inherit part of their physical layout from a superclass. When the event handlers are inherited along with the widgets, the window classes inherit the appropriate event handling code for the widgets they have inherited. For example, the following Display method for an inherited window uses its own event handlers as well as event handlers inherited from its superclass.

Code Example 11-2 Display Method Using Inherited Event Handler

```
-- This window is inherited from DataEntryWindow. The fields
-- related to the ArtObject object were added. This event loop
-- registers event handlers on self (that relate to the
-- art object) and event handlers on super (which handle
-- generic button events).

self.Open();

event loop

  preregister
  register self.ArtObjectHandler();

  register self.ResetHandler();

  -- Use the inherited event handler to handle exit
  -- button events.
  register super.ExitHandler();

end event;
self.Close();
```

Project: InheritedWindow • **Class:** ArtDataEntryWindow • **Method:** Display( )

Event handlers are also useful for storing event handling code in a modular, reusable form. For example, the event loop from the iPlanet UDS example program NestedWindow, shown earlier in this chapter, uses an event handler that is also used elsewhere in that application.

All event handlers consist of a name and TOOL event handling code to be included in one or more event statements. Some handlers also include parameters, which allow the programmer to pass information to the event handler, such as the object for which the events are being handled.

Overriding event handlers iPlanet UDS allows you to override an inherited handler. Overriding a handler means creating a handler for the current class that has the same name and parameters as a handler it inherited from one of its superclasses. By providing different source code for an inherited handler, you are creating a different implementation of the handler for objects of the current class. When you include the handler in method code for the current class, iPlanet UDS uses the handler you defined specifically for the class rather than the inherited handler.

No overloading iPlanet UDS does not allow you to overload event handlers. There can only be one event handler with a given name in the class.

Event handler visibility By default, the event handlers in a class are public, which means that any other classes in the project, or classes that use this project as a supplier, can access them. iPlanet UDS provides the option of defining any event handler as "private." A private event handler can be accessed only by the class that defines it, not by any other classes (even its own subclasses). Only methods in the current class can include a private event handler.

Event handler components An event handler consists of the following components:

Component	Description
name	The name the programmer will use to include the event handler in an event statement using the `register` statement.
parameters	Used by the programmer to provide input values to the event handler, such as the object for which the events are being handled.
source code	The code for handling one or more events plus optional `preregister` and `postregister` clauses.

The following sections provide detailed information about these components.

Event Handler Name

Overriding an inherited event handler The event handler name is the name you must use to reference the event handler in an event statement or another event handler. The name you choose for the event handler is very important because if you use the name of an inherited event handler, you will be overriding the inherited event handler. If you do not wish to override the inherited event handler, you should not use the existing name.

No overloading Unlike method handlers, there is no overloading for event handlers. There can only be one event handler with a given name in a class.

Of course, when you are writing an event handler that overrides another and you wish to register the very event handler that you are overriding, you can do this by using the `super` key word. See the `register` statement in the *TOOL Reference Guide* for information on registering an inherited event handler.

Event Handler Parameters

A event handler may have any number of input parameters. These parameters provide input values for the event handler, such as a reference to the object for which to handle the events.

Parameter name and type Every parameter has a name and a type. The name identifies the parameter and is for use when the handler is included in an event statement or another event handler using the `register` statement. The type specifies the type of value allowed for the parameter, which can be any simple data type or any class.

Class type parameters When the type of a parameter is a class, iPlanet UDS passes a reference to the object, not the object itself. Therefore, even though the parameter is for input only, if the event handler makes changes to the object, these changes are reflected when you exit the event handler. Both the invoking method and the event handler are referencing the same object.

Copy option The copy mechanism for a parameter allows you to prevent the event handler from referencing the same object as the method. When you specify the copy mechanism for a class type parameter, iPlanet UDS makes a copy of the object and passes a reference to that copy as the parameter.

Default value By default, the parameters defined on an event handler are required parameters. However, if you wish to make a parameter optional, you can specify a default value for the parameter. iPlanet UDS uses this value for the parameter when this parameter is excluded from the parameter list. The default value for the parameter must be compatible with the parameter's data type. For parameters with a class data type, the default value must be NIL, which means "no object."

Event Handler Source

The source of the handler is where you provide code to handle one or more events. To write the event handler, you use TOOL, the iPlanet UDS object-oriented programming language. See the *TOOL Reference Guide* for complete information on the language.

The syntax for the event handler source is:

```
[preregister statement_list]...
[[postregister] statement_list]
[when event_specification do statement_block]...
[exception_handler]
```

preregister clause The optional `preregister` clause provides a list of statements to be executed *before* iPlanet UDS registers the events in the `when` clause list. The `preregister` clause is especially useful for including other named event handlers in the current event handler definition using the `register` statement.

postregister clause The optional `postregister` clause provides a list of statements to be executed *after* iPlanet UDS registers the events in the `when` clause list, but before the events are handled by the event handler. The `postregister` clause is useful for ensuring that the code that posts an event is always executed before the event handler attempts to handle the event.

when clause The `when` clause identifies an event that you wish to handle and provides the corresponding code for that particular event. First, you must specify which event that you wish to receive. Second, you can declare a series of variables to receive the parameters that will be passed with the event. Finally, you must provide the statement block to be executed when the event is triggered. An event handler can include any number of `when` clauses for different events.

Exception handler The exception handler for the `event handler` statement provides exception handling for the event handler as a whole.

For a complete discussion of how to write the source for an event handler, see the `event handler` statement in the *TOOL Reference Guide*.

Statement breakpoints When you are entering your event handler source code in the Event Handler Workshop, you can set statement breakpoints in the code for use with the Debugger. The breakpoints that you set in the Event Handler Workshop take effect in the Debugger every time the event handler is executed by the application.

Statement breakpoints added in the Event Handler Workshop are persistent. Persistent breakpoints stay in effect until you remove them from the event handler. You can remove the persistent breakpoints either from the Event Handler Workshop or from the Debugger.

Setting breakpoints for an event handler is exactly the same as setting breakpoints for a method. See "Setting Breakpoints" on page 582 for information about how to set breakpoints.

Using the Event Handler Workshop

You enter the Event Handler Workshop from the Class Workshop either by opening an existing event handler or by creating a new event handler.

Opening an existing event handler If you wish to examine or edit an existing event handler, double-click the event handler name, or click the event handler name and choose the Element > Open command.

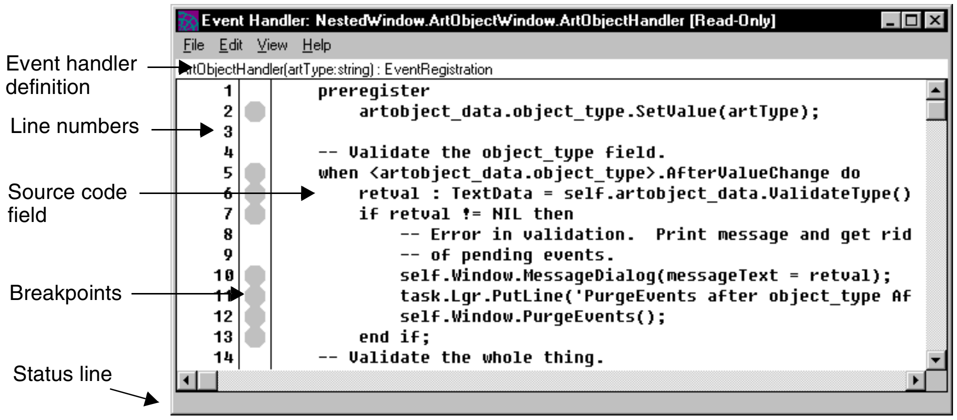
Creating a new event handler If you wish to create a new event handler, choose the Element > New > Event Handler command or single-click the New Event Handler tool.

The Event Handler Workshop Window

The Event Handler Workshop window consists of three parts: the event handler definition line, the event handler source code field, and the status line.

Figure 11-2 illustrates the Event Handler Workshop.

Figure 11-2 Event Handler Workshop



```
ArtObjectHandler(artType:string) : EventRegistration
```

```
preregister
    artobject_data.object_type.SetValue(artType);

-- Validate the object_type field.
when <artobject_data.object_type>.AfterValueChange do
    retval : TextData = self.artobject_data.ValidateType()
    if retval != NIL then
        -- Error in validation.  Print message and get rid
        -- of pending events.
        self.Window.MessageDialog(messageText = retval);
        task.Lgr.PutLine('PurgeEvents after object_type Af
        self.Window.PurgeEvents();
    end if;
-- Validate the whole thing.
```

View menu Note that the View menu in the Event Handler Workshop provides commands that let you specify whether to display the handler definition line and the status line. The event handler source field is always displayed, but you can turn the definition and status lines on or off as you choose.

Access to Other Workshops

From the Event Handler Workshop, you can access one other workshop:

Workshop	How to access it
Class Workshop	The File > Open Class… command opens the Class Workshop to display the definition of the class to which the current event handler belongs. If the class is already being displayed, the Open Class… command moves the input focus to the appropriate Class Workshop window.

Leaving the Event Handler Workshop

To leave the Event Handler Workshop and save all your changes, choose the File > Close command. This closes only the current workshop.

To leave the Event Handler Workshop and erase all changes you have made since your last Save All command, choose the File > Cancel command. This closes only the current workshop.

Examining an Event Handler

The Event Handler Workshop displays the definition for the current event handler. If the event handler you wish to examine is not already displayed, you must return to the Class Workshop to open the event handler.

➤ **To examine an event handler**

1. In the Class Workshop, double-click on the event handler name, or select the event handler name and choose the Element > Open command.
2. When the Event Handler Workshop, you can view the original handler definition, including the handler source code.

Examining Parameters and TOOL Source Code

The Event Handler Workshop displays the event handler's definition and TOOL source directly on the main Workshop window.

Definition The top line of the window displays the definition of the event handler using the syntax for the TOOL `event handler` statement (see *TOOL Reference Guide* for information on the `event handler` statement).

The event handler definition shows the event handler name and the event handler parameters, including the name and type.

Viewing parameters Note that if the event handler definition is not currently displayed in the workshop window, you can turn it on with the View > Parameters command. This command lets you control whether or not the event handler definition is displayed in current workshop. Your code preferences (see "Setting Code Preferences" on page 590) determine whether the event handler definition line is automatically displayed when you enter the workshop.

When the event handler definition is displayed, the parameter list may be too long to be completely displayed. In this case, you can scroll horizontally to view the complete definition.

Note that you can use the text editing commands to copy parameter text from the event handler definition and paste it into your source code.

TOOL source code The lower section of the window displays the source code for the event handler. This is a multiline text field, which you can scroll to view the entire method. Your preferences for the workshop (see "Setting Code Preferences" on page 612) determine whether line numbers and breakpoints are automatically displayed when you enter the workshop.

Viewing line numbers If line numbers are not currently displayed for the source code, you can turn them on with the View > Line Numbers command.

Viewing breakpoints If the breakpoints are not currently displayed for the source code, you can turn them on with the View > Breakpoints command.

Searching for a line If you wish to search for specific lines in the event handler, you can use the Edit > Go to Line or Find commands (see "Editing an Event Handler" on page 605 for information).

Examining Event Handler Properties

To view the event handler's properties, choose the File > Properties… command. This command opens the Event Handler Properties dialog, which displays the event handler's name, parameters, and visibility.

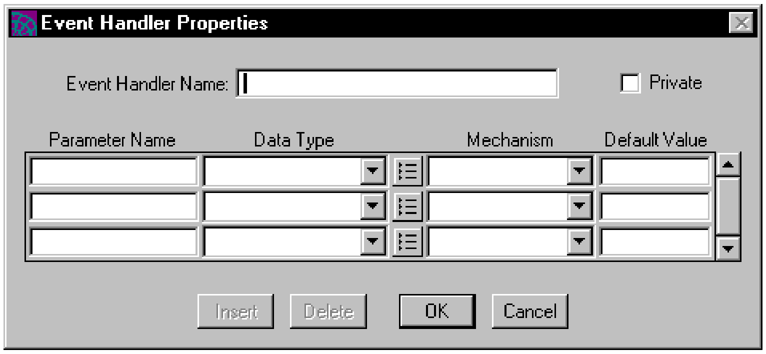
Creating an Event Handler

To create a new event handler, you must start from the Class Workshop. Unless you already have write access to the class, you must use the Project Workshop to check out or branch the class before you can create a new event handler for it. See "Write Access to Project Components" on page 257 for information.

➤ **To create a new event handler**

1. In the Class Workshop, choose the Element > New Event Handler command or single-click the New Event Handler tool.

   The Event Handler Properties dialog opens.



2. In the Event Handler Properties dialog, enter the event handler's name. The parameters and visibility are optional.
3. Click the OK button to add the event handler to the class and open the Event Handler Workshop.

The next section provides information about how to fill in the Event Handler Properties dialog. This is followed by information about how to enter the event handler source code.

Specifying Event Handler Properties

Event Handler Name property The event handler name is any legal iPlanet UDS name. Unless you wish to override the event handler, the name must be unique for the public names of its superclasses.

Overriding an inherited event handler If you specify the name of an inherited event handler, the new event handler you are creating will override the inherited event handler.

Private property By default, an event handler is public, which means all the other classes in the project have access to the event handler. The Private toggle lets you set the event handler's visibility to private. A private event handler can be accessed only by the class that defines it, not by any of the other classes in the project (not even by any of its subclasses). This essentially prevents the event handler from being inherited.

Event handler parameters To specify the event handler's parameters, you fill in an array field that specifies the name, type, mechanism, and optional default value for each parameter. The following table describes how to fill in the array field:

Column	How to fill it in
Parameter Name	The Parameter Name field is a data field. Simply type in any legal iPlanet UDS name.
Type	The Type field is a fillin field that allows you to choose one of the standard data types (described in Chapter 2) or type in a class name.
Mechanism	If you wish to use the copy option for a parameter of a class type, you can choose this option.

Column	How to fill it in
Default Value	If you wish to make a parameter optional, you can specify a default value for it. The default value for the parameter must be compatible with the parameter's data type. For parameters with a class data type, the default value must be "NIL," which means "no object." If you leave the default value blank, the parameter is required.

Event Handler Source Code

To write the event handler source code, you can either type the TOOL code directly into the source field, or you can write the event handler in a text file and then use the Import Text command to copy it into the source field. See "Importing and Exporting an Event Handler" on page 610 for information on the Import Text command.

Typing TOOL Code

Text editing The source code field is an iPlanet UDS text edit field. To type your event handler directly into the field, simply position the cursor in the source code field and start typing. To end a line, use the Return character. To indent a new line, use the tab character. In the Event Handler Workshop, the tab character is always set to four character spaces.

Editing shortcut keys The Event Handler Workshop provides text edit shortcut keys, which you can use to select text and move the cursor. In addition, the Edit menu provides editing commands for modifying the method source. See "Editing the TOOL Source Code" on page 605 for information on the shortcut keys and the Edit menu.

Drag and drop names Another way to enter text into the source code field is to drag the names of project components or class elements displayed in other workshops and drop them into the source code field. This allows you to quickly copy the names rather than keying them in.

Automatic indenting If you have your code preferences set with automatic indenting turned on, each line that you type in the source code field will automatically be indented with the same number of tabs as the line it follows. If you have your code preferences set with automatic indenting turned off, each line that you type will be aligned with the left margin. In this case, you must use tabs or spaces to indent the line or, after entering the code, you can use the Indent and Unindent commands (described under "Editing an Event Handler") to control the indenting of selected blocks of code. See "Setting Code Preferences" on page 590 for information on automatic indenting.

Editing an Event Handler

To edit an event handler, you must start from the Class Workshop. Unless you already have write access to the class, you must use the Project Workshop to check out or branch the class before you can edit the event handler. See "Write Access to Project Components" on page 257 for information.

➤ **To edit an event handler**

1. In the Class Workshop, double-click on the event handler name, or select the event handler name and choose the Element > Open command.
2. When the Event Handler Workshop opens, you can edit the event handler by changing its source code or by changing the event handler's properties.

Editing the TOOL Source Code

Edit shortcut keys The Event Handler Workshop provides the following text editing shortcuts:

Function	Shortcut on Windows/Motif
Select one character to the left	Shift-Left Arrow
Select one character to the right	Shift-Right Arrow
Select one line down	Shift-Down Arrow
Select one line up	Shift-Up Arrow
Select from the cursor position to the beginning of the current word	Shift-Ctrl-Left Arrow
Select from the cursor position to the end of the current word	Shift-Ctrl-Right Arrow
Select from the cursor position to the beginning of the current line	Shift-Home

Function	Shortcut on Windows/Motif
Select from the cursor position to the end of the current line	Shift-End
Select from the cursor position to the beginning of the text	Shift-Ctrl-Home
Select from the cursor position to the end of the text	Shift-Ctrl-End
Select all text	Ctrl-A
Move cursor to the beginning of the line	Home
Move cursor to the end of the line	End
Move cursor one word to the left	Ctrl-Left Arrow
Move cursor one word to the right	Ctrl-Right Arrow
Move cursor to the beginning of the text	Ctrl-Home
Move cursor to the end of the text	Ctrl-End

Edit menu The Edit menu in the Event Handler Workshop provides a set of basic commands for editing the method. These commands allow you to use the window system clipboard to cut, copy, and paste text, to search for and replace text, and to change indenting of selected lines.

Text Editing

The Edit menu provides the following commands for editing the text:

Command	Description
Undo	Removes your last edit.
Redo	Makes the edit you removed with the **Undo** command.
Cut	Removes the selected text and copies it into the clipboard.
Copy	Copies the selected text into the clipboard.
Paste	Inserts the current contents of the clipboard into the method.
Delete	Removes the selected text.

The Cut, Copy, and Paste commands use the clipboard provided by your window system.

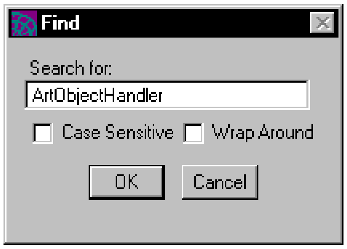
Searching and Replacing

There are two ways to move the cursor to a particular line in your code. You can use the Go to Line… command to go directly to the specified line number or you can use the Find… command to search for a particular string. The Replace… command lets you replace a search string with a new string.

The Go to Line… command prompts you to enter the number of the line to which you wish to move. After you specify the line number, iPlanet UDS moves the cursor to the first character of the line. If your line numbers are not currently displayed, you can use the View > Line Numbers command.

The Edit > Find… command lets you search for a particular string. When you give the Find… command, iPlanet UDS opens the Find dialog, where you specify the search string. The Find… command then searches for the string starting from the current insertion point. When it finds the string, iPlanet UDS highlights the string. Figure 11-3 illustrates the Find dialog:

Figure 11-3 Find Dialog



Search for: ArtObjectHandler

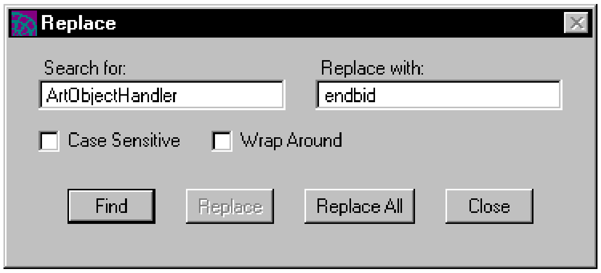
Case Sensitive option By default, the Find… command is case insensitive. If you want to match the case used in the search string, click on the Case Sensitive toggle in the Find dialog.

Wrap Around option By default, the Find… searches from the current insertion point to the end of the method source code. If you wish to search past the end and start from the beginning of the method source code, click on the Wrap Around toggle.

If you wish to move to the next occurrence of the string you searched for in the last Find… command without opening the Find dialog, you can simply give the Find Again command. This command uses the search string you specified in your last Find… command. The Find Again command is especially useful with its speed key.

The Edit > Replace… command lets you search for a particular string and replace that string with a new string. When you give the Replace… command, iPlanet UDS opens the Replace dialog, where you specify the search and replace strings. You then have the choice of using the Replace button to replace the first occurrence of the search string or the Replace All button to replace all occurrences of the search string within the method. Figure 11-4 shows the Replace dialog:

Figure 11-4 Replace Dialog



Depending on your code preferences, the Replace command works in one of two modes: Find After mode or Find Before mode. (See "Setting Code Preferences" on page 590 for information on setting the Replace modes.)

Find After mode By default, the Event Handler Workshop uses Find After mode. In this mode, the Replace command replaces the currently selected text and then automatically finds the next occurrence of the search string. Find After mode allows you to check the search string in context before you actually make the replacement.

Find Before mode If your code preferences are set to use the Find Before mode, the Replace command searches for the next occurrence of the search string and then automatically performs the replacement. Find Before mode allows you to check the replacement in context before moving on to the next occurrence of the search string.

If you wish to replace the next occurrence of a string without opening the Replace dialog, you can simply give the Replace Again command. This command uses the search and replace strings you specified in your last Replace… command, and performs the replacement according to the current Replace mode. In fact, giving the Replace Again command is the same as clicking the Replace button on the Replace dialog. The Replace Again command is especially useful with its speed key.

Indenting

The indenting commands let you move the currently selected lines of code down or up one level.

The Indent command moves the currently selected lines down one level by inserting a tab character at the start of each selected line.

The Unindent command moves the currently selected lines up one level by deleting a tab character from the start of each selected line.

Cancelling Your Changes

If you ever make a set of edits that you want to erase completely, you can use the File > Cancel command to revert to a previous version of the event handler. The Cancel command erases all changes you made to the event handler since you last saved your workspace or, if you have not given a Save All command, since you last opened the event handler.

Editing the Event Handler's Properties

To edit the event handler's properties, choose the File > Properties… command. This opens the Event Handler Properties dialog, where you can make any appropriate changes.

Editing the Parameters

The event handler's parameters are displayed in the array field in the Event Handler Properties dialog. You can change the existing parameters or add new ones to the end of the list.

Insert button If you wish to add a new parameter to the middle of the list, you can use the Insert button.

➤ **To insert a parameter**

1. Select the parameter above which you wish to add the new parameter.
2. Click the Insert button.
3. Fill in the new row in the array field.

Delete button If you wish to delete a parameter from the list, you can use the Delete button.

➤ **To delete a parameter**

1. Select the parameter you wish to delete.
2. Click the Delete button.

Importing and Exporting an Event Handler

iPlanet UDS lets you write an event handler or part of an event handler in an external text file and then import it into the Event Handler Workshop. You can also copy part or all of a event handler to an external text file for use outside of iPlanet UDS.

Importing TOOL Code

Use the Import Text… command to insert the contents of a text file into the event handler source. You can then edit the text as desired.

➤ **To import event handler text**

1. Position the cursor at the point where you wish to insert the imported text.
2. Choose the Edit > Import Text… command.
3. In the File Selection dialog, choose the file that you wish to import.

Exporting TOOL Code

Use the Export Text… command to write an event handler or a section of an event handler to a text file. This command writes the currently selected text into the file you specify. If you do not select any text, the Export Text… command exports the entire event handler.

➤ **To export event handler text**

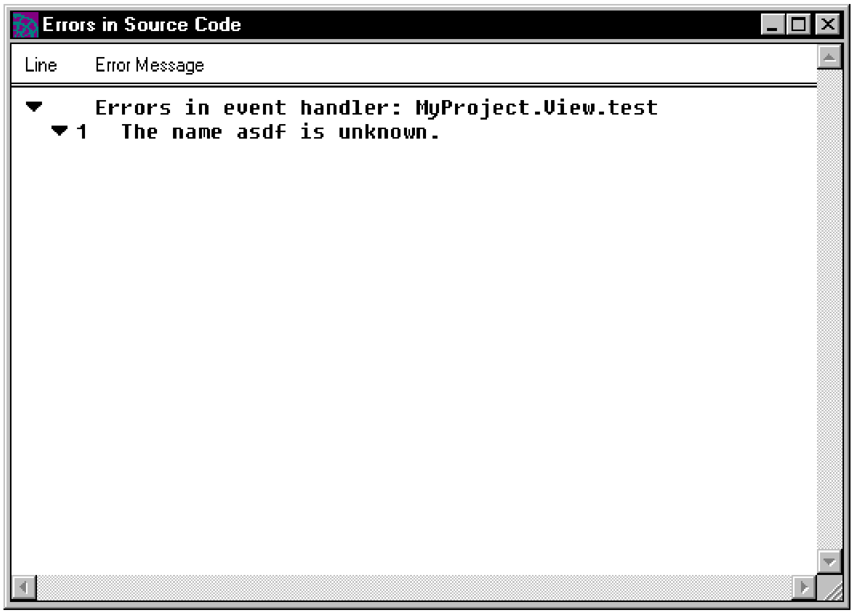
1. Select the text you wish to export.
2. Choose the Edit > Export Text… command.
3. In the File Selection dialog, specify the file name to which you wish to export the text.

If you specify a new file name, the Export Text… command creates the file. If you specify an existing file, the Export Text… command overwrites the file.

Compiling the Event Handler

After writing the event handler, you can compile it to check for errors. While it is not necessary to compile your event handler before you run the project, the Compile command allows you to check for syntax errors without actually executing the code. To compile the event handler, simply choose the File > Compile command. iPlanet UDS compiles the event handler and reports any errors in the Error window.

Finding an error The Error window displays the compilation errors messages for the event handler in an outline field.



```
▼     Errors in event handler: MyProject.View.test
  ▼ 1   The name asdf is unknown.
```

You can jump directly from one of these messages to the code that caused the error. In the Error window, double-click the error you wish to find. The Event Handler Workshop will then move the cursor to the line that contains the code that caused the error.

You can keep the Error window open as long as you need it. When you are finished using the window, simply close the window.

Setting Code Preferences

The Event Handler Workshop allows you to set code preferences that are saved as part of your current workspace. The code preferences that you set in the Event Handler Workshop take effect for the current Event Handler Workshop and all Method, Event Handler, and Cursor Workshops that are opened in the future from the same workspace. However if any other Method, Event Handler, or Cursor Workshops are already open, these will not be changed.

To set the code preferences, choose the File > Workshop Preferences… command. This command opens the Code Workshops Preferences dialog, where you can set any number of preferences. See "Setting Code Preferences" on page 590 for information about setting code preferences.

Chapter 12

Using the Cursor Workshop

This chapter provides background information about cursors and describes how to use the Cursor Workshop.

In this chapter, you will learn how to:

- examine a cursor
- create a cursor
- write the cursor source
- edit a cursor
- import and export a cursor
- test a cursor

About Cursors

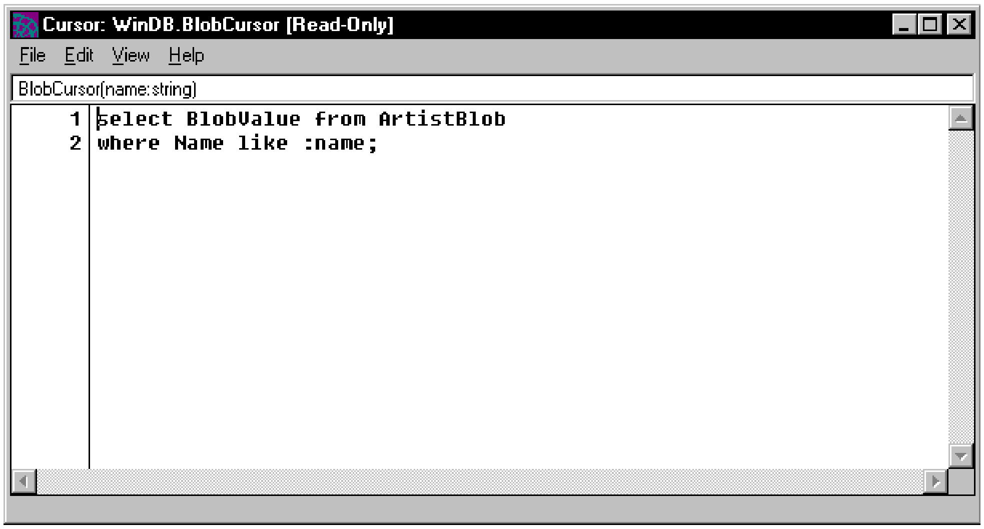
A cursor is a row marker that you can use for selecting and working with a set of rows from a database. The cursor definition consists of a name and a select statement that selects a set of rows from the database. After defining the cursor, there are two different ways you can use it.

First, you can specify the cursor name in the TOOL `for` statement to repeat a statement block for each row in the result set. The `for` statement automatically opens the cursor, fetches the rows one at time as it goes through the loop, and then closes the cursor.

Second, you can use the TOOL `sql open cursor` statement to open the cursor. Opening a cursor selects the rows from the database and positions the cursor before the first row in the result set. You can then use the TOOL `sql fetch cursor` statement to move one row at a time through the result set, and perform any processing you wish on individual rows. When you are finished, you use the TOOL `sql close cursor` command to close the cursor.

Figure 12-1 shows a cursor definition for BlobCursor in the Cursor Workshop. This cursor selects a single column from a database table where the supplied name parameter is like the Name column in the table. This is followed by example code from the example, from the Windb example which shows the cursor being used to fetch the BlobValue column from the ArtistBlob table.

Figure 12-1 Cursor Definition in Cursor Workshop



BlobCursor(name:string)

```
select BlobValue from ArtistBlob
where Name like :name;
```

Code Example 12-1 Using a Cursor Example

```
rowcount : integer = 1;
binData : BinaryData = new;
blobCurs : BlobCursor;

begin transaction

  sql open cursor blobCurs(name) on session self.MGRSession;
  rowcount =  (sql fetch cursor blobCurs into :binData);
  sql close cursor blobCurs;
  if rowcount = 0 then
```

Code Example 12-1 Using a Cursor Example *(Continued)*

```
    return NIL;
  end if;

end transaction;
```

Project: Windb • **Class:** ArtistBlobMgr • **Method:** GetArtist

Read-only cursor By default, a cursor can be used for reading only. When you use the `sql fetch` statement with a read-only cursor, you can access the values in the row, but you cannot update or delete them. Using a read-only cursor keeps the data available to others while you are working with the rows.

Cursor for updating However, the Cursor Workshop lets you specify that the cursor allows updating as well as reading. Declaring a cursor for updating provides a lock on the data to prevent inconsistencies during the update. When you open a cursor that has been declared for updating, other users will not be able to access the result set (and possibly other data) until you close the cursor.

You can either allow updating for all columns or limit updating to a specified list of columns. Although it is not required, we recommend that you specify the particular columns that can be updated. Limiting updating to a specified list of columns allows iPlanet UDS to optimize the code and prevents updating of columns that should not be changed.

A cursor consists of the following components:

Component	Description
Name	The name the caller will use to reference the cursor.
Placeholders	Names that represent values which will be supplied at runtime.
Cursor source	The cursor source consists of a select statement that selects a set of rows from the database and an optional `for` clause lets you allow updating for all columns or limit updating to a specified list of columns.

The following sections provide detailed information about the placeholders and cursor source.

Placeholders

A placeholder is a name that represents a value that will be supplied to the cursor's select statement at runtime. After declaring the placeholders, you can use them as stand-ins in the `where` and `having` clauses of the cursor's select statement. Finally, when you open the cursor, you can set the values of the placeholders as part of the `sql open cursor` statement.

Name and type for placeholders Each placeholder has a name and a type. The name can be any legal iPlanet UDS name. The type can be any iPlanet UDS simple data type or any subclass of the iPlanet UDS `DataValue` class.

Cursor Source

The cursor source consists of a select statement followed by an optional `for` clause

select statement The select statement for the cursor selects the database rows for processing. iPlanet UDS executes this select statement when you use the `sql open cursor` statement to open the cursor, and uses this select statement implicitly in a `for` statement that uses the cursor. You can use the `sql fetch cursor` statement to move the cursor through the result set.

The syntax for the select statement associated with a cursor is:

```
select [all | distinct] (* | column_list) from table_name [, table_name]...
       [where search_expression]
       [group by column_name [, column_name]... ]
       [having search_expression]
       [order by column [asc | desc] [, column [asc | desc] ]... ]
       [for {read only| update [of column [, column]...]}]
```

When you enter the select statement in the Cursor Workshop, you do not need a semicolon at the end.

See *Accessing Databases* for descriptions of the individual clauses.

Using placeholders You can use the placeholders that you declared for the cursor in the search expressions for the `where` and `having` clauses. Later, when you open the cursor, you can set the values for these placeholders. Because the placeholders are iPlanet UDS names, you must preface them with colons to distinguish them from column names. The following example illustrates:

```
select name, age from emptable
where age = :age_val
for update of age
```

For Clause

By default, a cursor is for reading only. The `for update` clause lets you allow updating as well as reading. You can either allow updating for all columns or limit updating to a specified list of columns.

Read-only cursor The `read only` option limits the cursor to reading only, which is the default. When you use the `fetch` statement with a read-only cursor, you can access the values in the row but you cannot update or delete it. Using a read-only cursor keeps the data available for update by others while you are working with the rows. Note that your particular DBMS may not support read-only cursors.

Update cursor The `update` option allows the cursor to be used for updating by providing a lock on the data to prevent inconsistencies during the update. When you open a cursor that has been declared for updating, other users will not be able to access the result set (and possibly other data) until you close the cursor. Although it is not required, we recommend that you use the `of` clause to specify the particular columns that can be updated.

Using the `of` clause to specify the particular columns to be updated allows iPlanet UDS to optimize the code and prevents updating of columns that should not be changed. Any columns that you do not include in the `of` clause are available for read only. Using the `update` option with the `of` clause will lock *all* the database columns.

Using the Cursor Workshop

You enter the Cursor Workshop from the Project Workshop either by opening an existing cursor or by creating a new cursor.

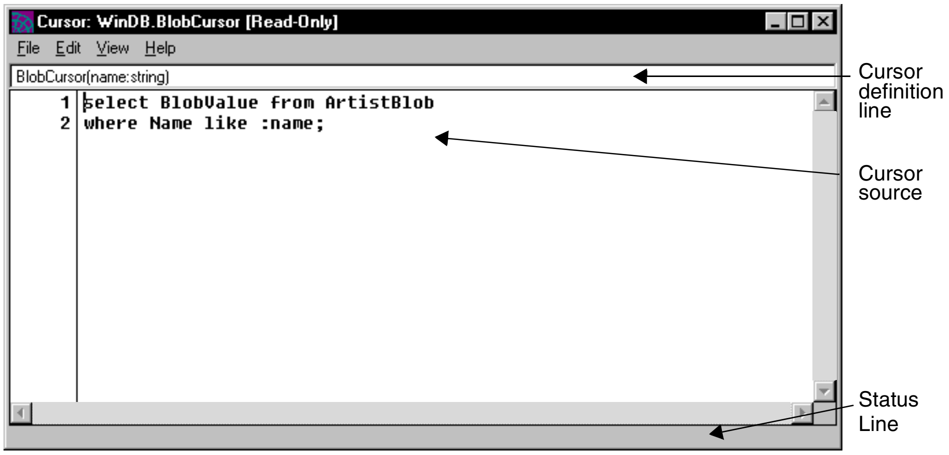
Opening an existing cursor If you wish to examine or edit an existing cursor, double-click the cursor name, or select the cursor name and choose the Component > Open command.

Creating a new cursor If you wish to create a new cursor, choose the Component > New Cursor command, or single-click the New Cursor tool on the toolbar.

The Cursor Workshop Window

The Cursor Workshop window consists of three parts: the cursor definition line, the cursor source code field, and the status line. Figure 12-2 illustrates the Cursor Workshop.

Figure 12-2 The Cursor Workshop



BlobCursor(name:string)

```
select BlobValue from ArtistBlob
where Name like :name;
```

View menu Note that the View menu in the Cursor Workshop provides commands that let you specify whether to display the cursor definition line and the status line. The cursor source code field is always displayed, but you can turn the definition and status lines on or off as you choose.

Access to Other Workshops

From the Cursor Workshop, you can access one other workshop:

Workshop	How to access it
Project Workshop	The File > Open Project… command opens the Project Workshop to display the definition of the project to which the current cursor belongs. If the project is already being displayed, the Open Project… command moves the input focus to the appropriate Project Workshop window.

Leaving the Cursor Workshop

To leave the Cursor Workshop, choose the File > Close command. You can also use the close box on the system menu. This closes only the current workshop.

To leave the Cursor Workshop and erase all changes you have made since your last Save All command, choose the File > Cancel command. This closes only the current workshop.

Examining a Cursor

The Cursor Workshop displays the definition for the current cursor. If the cursor you wish to examine is not already displayed, you must return to the Project Workshop to open the cursor.

➤ **To examine a cursor**

1. In the Project Workshop, select the cursor you wish to examine.
2. Double-click the cursor name, or choose the Component > Open command.

Examining Placeholders and TOOL Source

The Cursor Workshop displays the cursor placeholders and TOOL source directly on the main Workshop window.

Definition The top line of the window displays the cursor name and the cursor placeholders if any.

Viewing placeholders Note that if the cursor definition is not currently displayed in the workshop window, you can turn it on with the View > Placeholders command. This command lets you control whether or not the cursor definition is displayed in current workshop.

TOOL source code The lower section of the window displays the source code for the cursor. This is a multiline text field, which you can scroll to view the entire select statement.

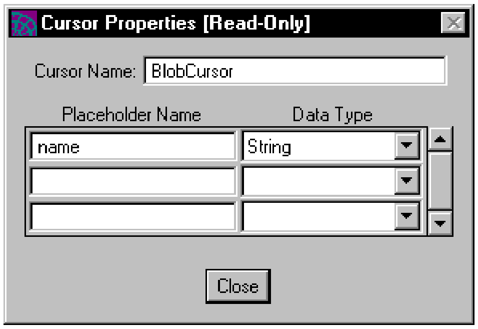
Viewing line numbers If line numbers are not currently displayed for the source code, you can turn them on with the View > Line Numbers command.

Searching for a line If you wish to search for specific lines in the cursor, you can use the Edit > Go to Line or Find commands (see "Editing a Cursor" on page 622 for information on these).

Examining Cursor Properties

To view the cursor's properties, choose the File > Properties… command. This command opens the Cursor Properties dialog, shown in Figure 12-3, which displays the cursor's name and placeholders.

Figure 12-3 Cursor Properties Dialog


Cursor Properties [Read-Only]

Cursor Name: BlobCursor

Placeholder Name	Data Type
name	String

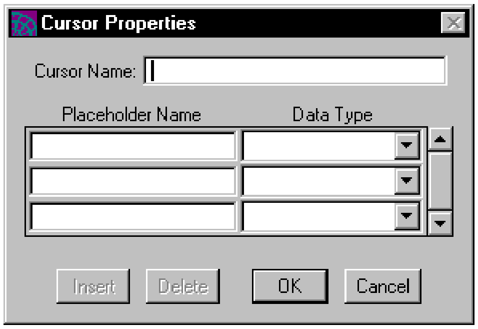
Creating a Cursor

To create a new cursor, you must start from the Project Workshop. Either use the Component > New Cursor command, or single-click the New Cursor tool.

➤ **To create a new cursor**

1. Choose the Component > New Cursor command.

   The Cursor Properties dialog opens.



2. In the Cursor Properties dialog, enter the cursor's name in the Cursor Name field, specify the cursor's placeholders (described below) if any, and click OK.
3. When the Cursor Workshop opens, you can write the cursor source code. See below for complete information on writing the cursor source code.

To specify the cursor's placeholders, you fill in an array field that specifies the name and type for each placeholder.

Column	How to fill it in
Placeholder Name	Type the name into the data field.
Data Type	Select one of the standard data types from the drop list.

Writing the Cursor Source

To write the cursor's source, type the TOOL code directly into the Source field, or you can write the cursor in a text file and then use the Import Text… command to copy it into the Source field. See "Importing Cursor Code" on page 628 for information on the Import Text… command.

Typing the Cursor Source

Text editing The Source field is an iPlanet UDS text edit field. To type your cursor directly into the field, simply position the cursor in the field and start typing. The Edit Menu provides editing commands for modifying the cursor source (see "Editing a Cursor" for information).

Editing shortcut keys The Cursor Workshop provides text edit shortcut keys, which you can use to select text and move the cursor. In addition, the Edit menu provides editing commands for modifying the method source. See "Editing the Cursor Source Code" on page 623 for information on the shortcut keys and the Edit menu.

Automatic indenting If you have your code preferences set with automatic indenting turned on, each line that you type in the source code field will automatically be indented with the same number of tabs as the line it follows. If you have your code preferences set with automatic indenting turned off, each line that you type will be aligned with the left margin. In this case, you must use tabs or spaces to indent the line or, after entering the code, you can use the Indent and Unindent commands (described under "Editing the Cursor Source Code" on page 623) to control the indenting of selected blocks of code. See "Setting Code Preferences" on page 590 for information on automatic indenting.

Editing a Cursor

To edit a cursor, you must start from the Project Workshop.

➤ **To edit a cursor**

1. Select the cursor you wish to edit.

2. Double-click the cursor, or choose the Component > Open command.

This command opens the Cursor Workshop, which displays the selected cursor. You can then edit the cursor by editing its source or modifying its properties.

Editing the Cursor Source Code

Edit shortcut keys The Cursor Workshop provides the following text editing shortcuts:

Function	Shortcut on Windows/Motif
Select one character to the left	Shift-Left Arrow
Select one character to the right	Shift-Right Arrow
Select one line down	Shift-Down Arrow
Select one line up	Shift-Up Arrow
Select from the cursor position to the beginning of the current word	Shift-Ctrl-Left Arrow
Select from the cursor position to the end of the current word	Shift-Ctrl-Right Arrow
Select from the cursor position to the beginning of the current line	Shift-Home
Select from the cursor position to the end of the current line	Shift-End
Select from the cursor position to the beginning of the text	Shift-Ctrl-Home
Select from the cursor position to the end of the text	Shift-Ctrl-End
Select all text	Ctrl-A
Move cursor to the beginning of the line	Home
Move cursor to the end of the line	End
Move cursor one word to the left	Ctrl-Left Arrow
Move cursor one word to the right	Ctrl-Right Arrow
Move cursor to the beginning of the text	Ctrl-Home
Move cursor to the end of the text	Ctrl-End

Edit menu The Edit menu in the Cursor Workshop provides a set of basic commands for editing the method. These commands allow you to use the window system clipboard to cut, copy, and paste text, to search for and replace text, and to change indenting of selected lines.

Text Editing

The Edit menu in the Cursor Workshop provides a set of basic commands for editing the source code. There are six commands for editing the text:

Command	Description
Undo	Removes your last edit.
Redo	Reverses your last Undo command.
Cut	Removes the selected text and copies it into the clipboard.
Copy	Copies the selected text into the clipboard.
Paste	Inserts the current contents of the clipboard into the cursor.
Delete	Removes the selected text.

The Cut, Copy, and Paste commands use the clipboard provided by your window system.

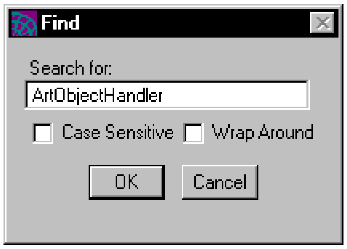
Searching and Replacing

There are two ways to move the cursor to a particular line in your code. You can use the Go to Line… command to go directly to the specified line number or you can use the Find… command to search for a particular string. The Replace… command lets you replace a search string with a new string.

The Go to Line… command prompts you to enter the number of the line to which you wish to move. After you specify the line number, iPlanet UDS moves the cursor to the first character of the line. If your line numbers are not currently displayed, you can use the View > Line Numbers command.

The Edit > Find… command lets you search for a particular string. When you give the Find… command, iPlanet UDS opens the Find dialog, where you specify the search string. The Find… command then searches for the string starting from the current insertion point. When it finds the string, iPlanet UDS highlights the string. Figure 12-4 illustrates the Find dialog:

Figure 12-4 Find Dialog



ArtObjectHandler

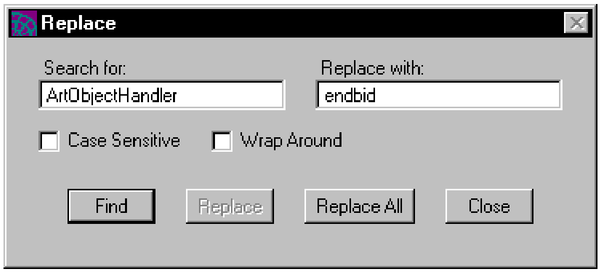
Case Sensitive option By default, the Find… command is case insensitive. If you want to match the case used in the search string, click on the Case Sensitive toggle in the Find dialog.

Wrap Around option By default, the Find… searches from the current insertion point to the end of the method source code. If you wish to search past the end and start from the beginning of the method source code, click on the Wrap Around toggle.

Find Again command If you wish to move to the next occurrence of the string you searched for in the last Find… command without opening the Find dialog, you can simply give the Find Again command. This command uses the search string you specified in your last Find… command. The Find Again command is especially useful with its speed key.

Replace… command The Edit > Replace… command lets you search for a particular string and replace that string with a new string. When you give the Replace… command, iPlanet UDS opens the Replace dialog, where you specify the search and replace strings. You then have the choice of using the Replace button to replace the first occurrence of the search string or the Replace All button to replace all occurrences of the search string within the method. Figure 12-5 shows the Replace dialog:

Figure 12-5 Replace Dialog



ArtObjectHandler
endbid

Depending on your code preferences, the Replace command works in one of two modes: Find After mode or Find Before mode. (See "Setting Code Preferences" on page 590 for information on setting the Replace modes.)

Find After mode By default, the Cursor Workshop uses Find After mode. In this mode, the Replace command replaces the currently selected text and then automatically finds the next occurrence of the search string. Find After mode allows you to check the search string in context before you actually make the replacement.

Find Before mode If your code preferences are set to use the Find Before mode, the Replace command searches for the next occurrence of the search string and then automatically performs the replacement. Find Before mode allows you to check the replacement in context before moving on to the next occurrence of the search string.

Replace Again command If you wish to replace the next occurrence of a string without opening the Replace dialog, you can simply give the Replace Again command. This command uses the search and replace strings you specified in your last Replace… command, and performs the replacement according to the current Replace mode. In fact, giving the Replace Again command is the same as clicking the Replace button on the Replace dialog. The Replace Again command is especially useful with its speed key.

Indenting

The indenting commands let you move the currently selected lines of code down or up one level.

The Indent command moves the currently selected lines down one level by inserting a tab character at the start of each selected line.

The Unindent command moves the currently selected lines up one level by deleting a tab character from the start of each selected line.

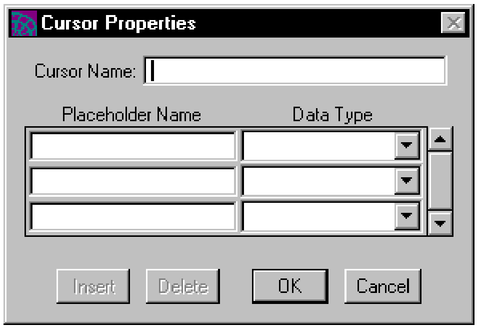
Cancelling Your Changes

If you ever make a set of edits that you want to erase completely, you can use the Cancel command on the File menu to revert to a previous version of the cursor. The Cancel command erases all changes you made to the cursor since you last opened it or since your last Save All command.

Editing the Cursor's Properties

To edit the cursor's properties, choose the File > Properties… command. This command opens the Cursor Properties dialog, where you can change the cursor name or placeholders.

Figure 12-6 Cursor Properties Dialog



Editing the Cursor's Placeholders

The cursor's placeholders are displayed in the array field in the Cursor Properties dialog. You can simply edit the existing placeholders or add new ones to the middle or end of the list.

Insert button If you wish to add a new placeholder to the middle of the list, use the Insert button.

➤ **To insert a placeholder**

1. Select the placeholder above which you wish to add the new placeholder.
2. Click the Insert button.
3. Fill in the new row in the array field.

Delete button If you wish to delete a placeholder from the list, use the Delete button.

➤ **To delete a placeholder**

1. Select the placeholder you wish to delete.
2. Click the Delete button.

Importing and Exporting a Cursor

iPlanet UDS lets you write the cursor code or part of the cursor code in an external text file and then import it into the Cursor Workshop. You can also copy part or all of the cursor code to an external text file.

Importing Cursor Code

Use the Import Text… command to insert the contents of a text file into the cursor source. You can then edit the text as desired.

➤ **To import a cursor**

1. Position the cursor at the point where you wish to insert the imported text.
2. Choose the Edit > Import Text… command.
3. In the File Selection dialog, select the file that you wish to import.

Exporting Cursor Code

Use the Export Text… command to write a cursor or a section of a cursor to a text file. This command writes the currently selected text into the file you specify. If you do not select any text, the Export Text… command exports the entire cursor.

➤ **To export a cursor**

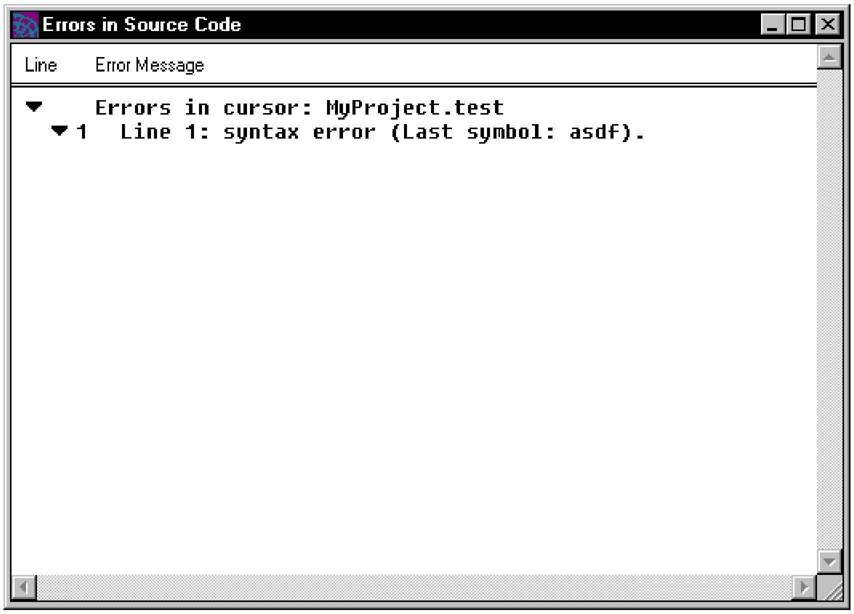
1. Select the text you wish to export.
2. Choose the Edit > Export Text… command.
3. In the File Selection dialog, specify the file name to which you wish to export the text.

If you specify a new file name, the Export Text… command creates the file. If you specify an existing file, the Export Text… command overwrites the file.

Testing the Cursor

After writing the cursor, you can compile it to check for errors. While it is not necessary to compile your cursor before you run the project, the Compile command allows you to check for syntax errors without actually executing the code. To compile the cursor, simply select the Compile command from the File menu. iPlanet UDS compiles the cursor and reports any errors in the Error window.

Finding an error The Error window displays the compilation errors messages for the SQL statement in an outline field.



```
▼    Errors in cursor: MyProject.test
  ▼1   Line 1: syntax error (Last symbol: asdf).
```

You can jump directly from one of these messages to the code that caused the error. In the Error window, double-click the error you wish to find. The Cursor Workshop will then move the cursor to the line that contains the code that caused the error.

You can keep the Error window open as long as you need it. When you are finished using the window, simply close the window.

Setting Preferences

The Cursor Workshop allows you to set code preferences that are saved as part of your current workspace. The code preferences that you set in the Cursor Workshop take effect for the current Cursor Workshop and all Method, Event Handler, and Cursor Workshops that are opened in the future from the same workspace. However if any other Method, Event Handler, or Cursor Workshops are already open, these will not be changed.

To set the code preferences, choose the File > Workshop Preferences… command. This command opens the Code Preferences dialog, where you can set any number of preferences. See "Setting Code Preferences" on page 590 for information about setting code preferences.

Chapter 13

Using the Debugger

This chapter describes how to use the iPlanet UDS Debugger to debug your application.

In this chapter, you will learn how to:

- control program execution
- set breakpoints
- examine the program state
- debug multiple tasks

Because this chapter provides information about how to find bugs in your TOOL code, you must be familiar with the programming language in order to understand the material. This chapter assumes that you have read the *TOOL Reference Guide* and have some experience writing iPlanet UDS methods and event handlers using TOOL.

About the Debugger

The iPlanet UDS Debugger was designed specifically for debugging TOOL code. Special features include the ability to set breakpoints for events and exceptions, and to debug multiple tasks in parallel.

Breakpoints

To help you monitor the flow of the application's execution, iPlanet UDS lets you set breakpoints on the following:

- statements
- method entry and exit
- event posting and delivery
- exception raising

Persistent breakpoints The statement breakpoints you set in the Debugger are persistent; they stay in effect from the time you set them until you explicitly turn them off.

Temporary breakpoints All other breakpoints you set in the Debugger are temporary; they take effect for the current Debugger session only.

Setting breakpoints in other workshops iPlanet UDS lets you set statement breakpoints directly in your TOOL code. In the Method, Cursor, and Event Handler workshops, you can to turn on breakpoints for any statements in the source code. Like the statement breakpoints you set in the Debugger, the statement breakpoints you set in these workshops are persistent. See "Setting Breakpoints" on page 640 for information about setting persistent breakpoints in these workshops.

Multitasking

As an application runs, the Debugger displays each task individually. In each task, you can set breakpoints, step through the code, and view the values of local variables. Although an application can execute any number of tasks concurrently, you may not wish to view them all simultaneously. Therefore, iPlanet UDS lets you decide which tasks to display and which tasks to hide.

This chapter describes how to debug a single task, and includes a special section on working with multiple tasks, "Working with Multiple Tasks" on page 649.

Debugging Distributed Applications

iPlanet UDS applications are designed to be distributed. However, you can only step through code that is executing locally. If a method is invoked on a remote object, iPlanet UDS executes the complete method. You cannot step through the statements individually.

Using the Debugger from the Project Workshop Therefore, when you run the Debugger from the Project Workshop, iPlanet UDS automatically partitions your application so that as much of it as possible will run on your workstation. This allows you to step through code that would normally be executing remotely.

To partition your application for debugging, iPlanet UDS changes the visibility for all your environment-wide service objects to user visibility. The user interface for the application, the application startup code, and all service objects that the client node can support are assigned to the client partition. iPlanet UDS runs the client partition directly on your workstation. (For information about client partitions, see "About Partitions" on page 664.)

Any service objects that the client node cannot support (because the external manager is not available on the client node) are assigned to a special private partition. iPlanet UDS automatically installs this partition on a suitable node in your development environment, ensuring that two people testing the same application will not accidentally share the same service object.

Using the Debugger from the Partition Workshop Debugging the application from the Partition Workshop allows you to run the application using the current configuration and to step through the code running only on the client partition. You cannot use the iPlanet UDS Debugger to monitor code running on remote partitions.

Using the Debugger

This section describes how to start and stop the Debugger, provides usage information about the Debugger windows, and discusses how to control program execution.

Throughout this section, references are made to the state of radio buttons. The following table lists the button states and their physical descriptions.

Button State	Description
Inactive/off	Grey (unfilled)
Selected	Filled (with dark color)
Current	Highlighted (bright color)

Starting the Debugger

You can start the Debugger either from the Project Workshop or the Partition Workshop at any time during the development process. The Debugger executes the application and allows you to monitor the code as it is being executed. When you have located your bug and are ready to fix your code, you must exit the Debugger and return to the appropriate workshop to edit your original source code.

Start class and method To run the Debugger, you must specify a start class and method for the main project in the application. Start classes and methods are described "Start Class and Method" on page 208. The Debugger uses the start class and method as the starting point for running the application.

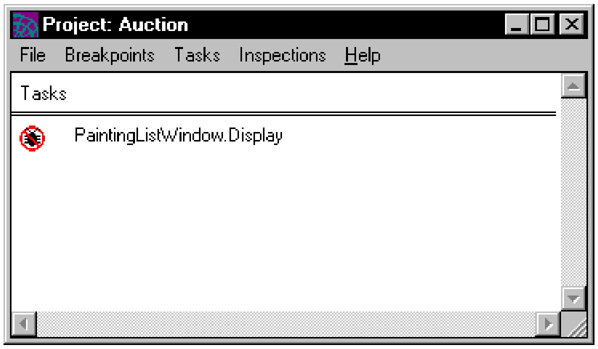
When you start the Debugger, iPlanet UDS begins executing the application by constructing a new object of the start class. The Debugger displays the code for the startup method in the Task window, and suspends execution immediately before the first statement in the startup method. Note that the Init method that is automatically invoked on the starting class is not under the control of the Debugger.

The Debugger Windows

As you enter the Debugger, iPlanet UDS opens two windows: the Debugger window and the Task window.

Debugger Window The Debugger Window, shown in Figure 13-1, displays a list of the tasks in the application currently being executed, and provides access to the Global Breakpoint Manager.

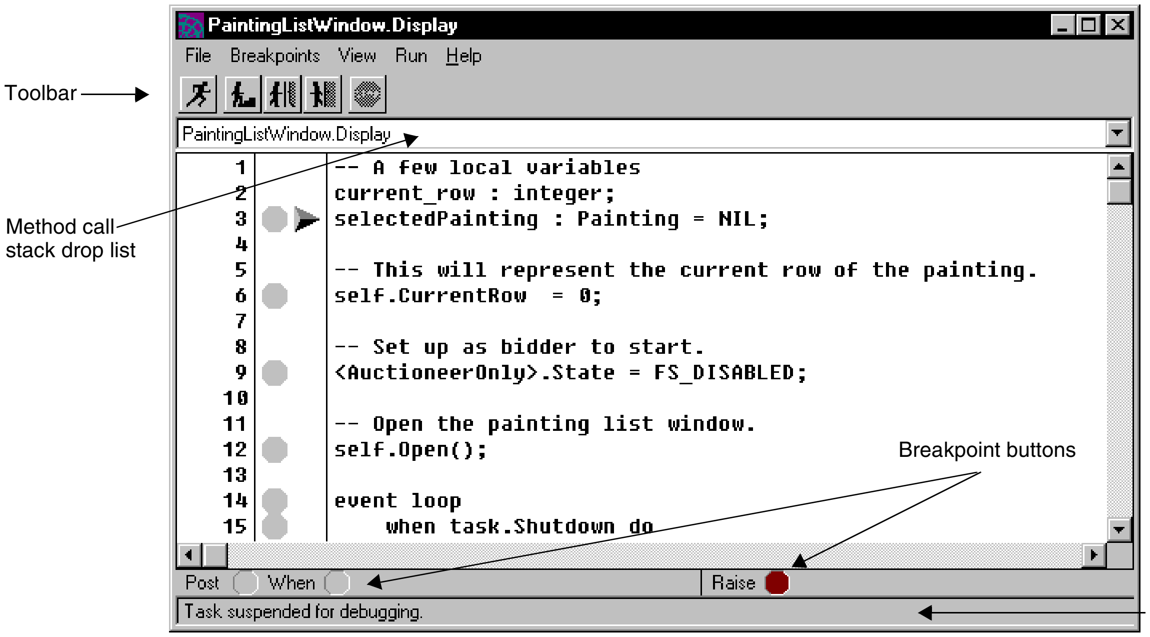
Figure 13-1 Debugger Window



PaintingListWindow.Display

Task Window The Task window, shown in Figure 13-2, displays code for the method in the task that is currently being executed, and allows you to set breakpoints on the code. As you enter the Debugger, this window displays the code for the startup method.

Figure 13-2 Task Window


PaintingListWindow.Display
File
Breakpoints
View
Run
Help
Toolbar
PaintingListWindow.Display
Method call stack drop list
Breakpoint buttons
Post
When
Raise

```
-- A few local variables
current_row : integer;
selectedPainting : Painting = NIL;

-- This will represent the current row of the painting.
self.CurrentRow  = 0;

-- Set up as bidder to start.
<AuctioneerOnly>.State = FS_DISABLED;

-- Open the painting list window.
self.Open();

event loop
    when task.Shutdown do
```

Task suspended for debugging.

Access to Other Workshops

The Debugger does not provide access to any other workshops.

Leaving the Debugger

You can leave the Debugger at any time by choosing the File > Exit Debugger command on either the Debugger window or any of the Task windows. The Exit Debugger command ends your debugging session and closes all the Debugger windows. The global statement breakpoints that you set during your session are saved. All other breakpoints are lost.

Controlling Program Execution

This section describes how to:

- start execution
- suspend execution to examine the program state
- step into and out of methods
- restart the application after it completes

As you debug your application, the Debugger uses the following icons to indicate the current state of the task.

Icon	Description
🏃	Indicates the task is running.
🐞	Indicates the task is suspended.
🧍	Indicates the task is waiting for an event.
🪦	Indicates the task has completed.

Run Menu Commands on Task Window

The Debugger provides the following commands on the Run menu of the Task window for controlling program execution:

Tool	Command	Description
🏃	Run	Executes the application from the startup class and method, or resumes execution after the task is suspended.
🛑	Stop	Suspends execution of the task so you can examine the program state.
🚶	Step	Executes the current statement.
🚪	Step In	Executes the first statement in the invoked method.

Tool	Command	Description
	Step Out	Finishes executing the current method and returns to the statement that originally invoked the method.
	Cancel Task	Stops executing the task and closes associated windows.
	Restart	After the application has completed, executes the application again.

The remainder of this section describes these commands in further detail.

Starting Execution

To start the application from the main project's startup class and method, click the Run button in the Task window or choose the Run > Run command. iPlanet UDS starts the task by executing the statement immediately after the right-pointing arrow. If the application has a user interface, iPlanet UDS displays the windows as they are opened so that you can interact with them.

iPlanet UDS executes the task until one of the following occurs:

- you interrupt execution with the Run > Stop command
- iPlanet UDS reaches a breakpoint
- the task is waiting for an event from a window that it is displaying
- the application terminates
- you cancel the task with the Run > Cancel Task command

Resuming execution When the task is suspended because you gave a Run > Stop command or because iPlanet UDS has reached a breakpoint, you can resume execution with the Run > Run command. iPlanet UDS resumes execution by executing the statement immediately after the right-pointing arrow.

When a task is suspended because it is waiting for an event from a window (that is, the event loop for the Display method for window is waiting for an end user interaction with the window), you will see the waiting icon next to the task name in the Debugger window. To resume execution of a task that is waiting for an event, you must perform the action the event loop is waiting for.

At any point during execution, you can cancel the task by using the l Run > Cancel Task command.

For information about restarting the application after it has completed or you have cancelled the task, see "Application Completion" on page 640.

Suspending Execution

At any point while the application is executing, you can use the Run > Stop command on the Task window to suspend execution. iPlanet UDS suspends execution as soon as possible, and indicates the stopping point with a right-pointing arrow. When a task is suspended, the icon in the Debugger window changes to a debug icon—in other words, the debugger is blocking the task.

Reaching a breakpoint When the application reaches a breakpoint, iPlanet UDS suspends execution and indicates the breakpoint with a right-pointing arrow. The button corresponding to the method, event, or exception breakpoint is highlighted. See "Setting Breakpoints" on page 640 to learn how to set the breakpoints.

Current program state When execution is suspended, either by the Stop command or by a breakpoint, you can examine the current state of the program. You can open the Variables window to display the current values for the method's local variables and the Method Call Stack drop list to view the invoking methods. See "Examining the Program State" on page 642 for information on how to examine this information.

After suspending execution and examining the program state, you have three choices:

- step to the next statement
- step into an invoked method
- click the Run button to resume execution

The following sections describe how to step through your code.

Stepping by Statements

To execute the statement immediately following the last stopping point, use the Run > Step command on the Task window. The Step command executes a single statement. The right-pointing arrow indicates the statement that was executed, and the Variable window displays the current values of the method's local variables.

If the statement that you step to is a method, the Step command executes the entire method. Alternatively, you can step into the method and step through its statements one at a time by using the Step In command.

Stepping In and Out of Methods

When the next statement in your code is a method invocation, you can use the Run > Step In command on the Task window to step to the first statement of that method. While debugging this method, you can execute the rest of the method and return to the invoking method by using the Run > Step Out command on the Task window.

Step In Command

The Run > Step In command on the Task window executes the first statement in the invoked method. The Task window displays the code for the method, and the right-pointing arrow indicates the statement awaiting execution. The Variable window displays the current values for the invoked method's local variables.

If the method is being invoked on a remote object or a 3GL object, iPlanet UDS cannot step into the method, and the Step In command will not be available to you. You also cannot step into a method that is contained in a library.

Step Out Command

Any time while you are debugging a method, you can return to the method that originally invoked it. The Run > Step Out command on the Task window finishes executing the current method and returns to the statement that originally invoked the method.

Even though the Step Out command is designed to finish executing the current method, if there are any remaining breakpoints in the current method, these will cause the Debugger to stop. If you wish to ignore these breakpoints, you must turn them off before giving the Step Out command.

If you give the Step Out command in the first method of a task, this completes the entire task. The Task window closes and the Debugger window displays the RIP icon by the task name.

Application Completion

When the application completes, iPlanet UDS closes the Task window and the Variable window for the task. The Debugger window displays a list of the remaining tasks, with RIP icons indicating that all are now terminated.

At this point, you can start the Debugger again by choosing the File > Restart command on the Debugger window. iPlanet UDS runs the application again, and opens the Task window for the startup method. The terminated tasks from your previous run are no longer displayed in the Debugger window.

Setting Breakpoints

The iPlanet UDS Debugger lets you set the following kinds of breakpoints:

Breakpoint	Description
Statement	Stops execution immediately prior to statement defined as breakpoint.
Method Entry	Stops execution when any method is invoked.
Method Exit	Stops execution when any method ends.
Event Posted	Stops execution when any event is posted.
Event Delivered	Stops execution when any event is about to be handled.
Exception Raised	Stops execution when any exception is raised.

Setting Statement Breakpoints

The Task window displays the method that is currently being executed. You can set persistent statement breakpoints in the current method directly in this window.

➤ **To set a breakpoint in the method currently displayed in the Task window**

1. Scroll to the statement where you want the breakpoint.
2. Click the grey toggle to the left of the statement. A stop sign icon indicates that the statement now has a breakpoint.

To remove a breakpoint, simply click the stop sign icon. A grey toggle indicates that the statement no longer has a breakpoint.

To set breakpoints in a method that will be invoked from the current method, you must either step into the method to display it in the Task window or use the Method Workshop to set persistent breakpoints directly in the method source code.

Global Breakpoint Manager You can view all the statement breakpoints for the project you are debugging by using the Global Breakpoint Manager. To open the Global Breakpoint Manager, use the Breakpoints > Global Breakpoint Manager command on the Task window. The Global Breakpoint Manager window displays a list of the breakpoints that you have already set. Although you cannot set new breakpoints in this window, you can remove them. To remove a breakpoint, simply click the stop sign icon.

Note that when you are debugging an application that uses multitasking, you can set breakpoints for an individual task within the application. See "Working with Multiple Tasks" on page 649 for information on setting task breakpoints.

Setting Method Breakpoints

You can request two different method breakpoints: entry and exit. An entry breakpoint takes effect each time a new method is invoked. An exit breakpoint takes effect each time iPlanet UDS exits a method (either because it completes or because it was terminated due to an exception).

To request entry breakpoints, choose the Breakpoints > Method Entry command on the Task window. When entry breakpoints are on, iPlanet UDS suspends execution every time a method is invoked.

To request exit breakpoints, choose the Breakpoints > Method Exit command on the Task window. When exit breakpoints are on, iPlanet UDS suspends execution every time a method ends.

To turn these breakpoints off, choose the appropriate menu command again to toggle it off.

Setting Event Breakpoints

You can request two different event breakpoints: event delivered and event posted. An event delivered breakpoint takes effect every time an event is going to be handled by a `when` clause of an `event loop` or `event case` statement. An event posted breakpoint takes effect each time an event is posted by any method in the task.

Event delivered breakpoints To request event delivered breakpoints, click the When radio button on the status line. This turns the toggle red, indicating that breakpoints are on. (You can also use the Breakpoints > Event Delivered command on the Task window to request event delivered breakpoints.)

Event posted breakpoints To request event posted breakpoints, click the Post radio button on the status line. This turns the toggle red, indicating that event posted breakpoints are on. (You can also use the Breakpoints > Event Posted command on the Task window to request event posted breakpoints.)

To turn the event breakpoints off, click the corresponding radio button so it turns back to grey.

Setting Exception Raised Breakpoints

Raise breakpoint You can request a breakpoint that takes effect each time an exception is raised by a `raise` statement in any custom method in the task.

To request raise breakpoints, click the Raise radio button on the status line. This turns the toggle red, indicating that raise breakpoints are on. (You can also use the Breakpoints > Exception Raised command on the Task window to request raise breakpoints.)

To turn the raise breakpoints off, click the Raise radio button so it turns back to grey.

Examining the Program State

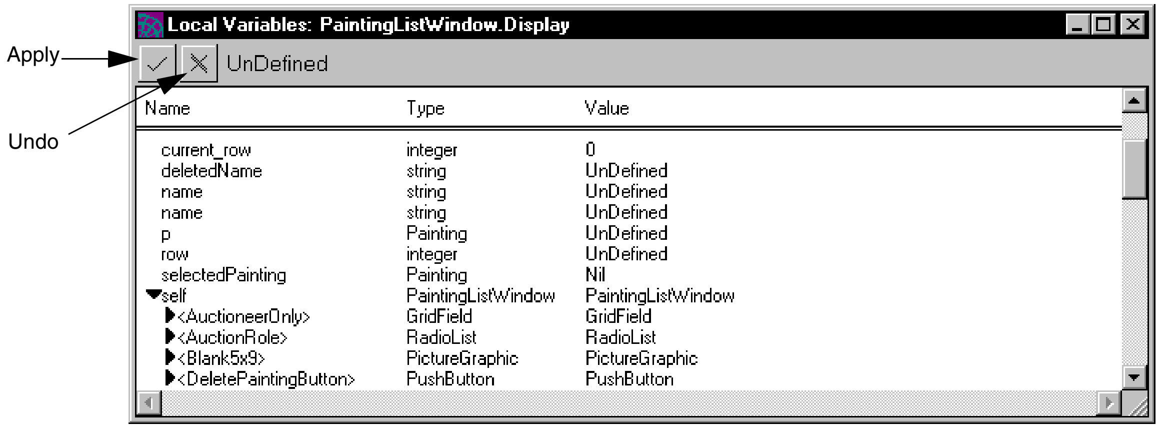
When program execution is suspended, you can examine the current state of the program. The Debugger lets you inspect:

- current values of the method's local variables
- call stack for the current method
- error stack for the task
- event queue for the task

Inspecting Variables

Local Variables command To view the local variables for the method, choose the View > Local Variables command on the Task window. This command opens the Variables window, which displays the current values for each of the method's local variables.

Figure 13-3 Variables Window



Local Variables: PaintingListWindow.Display

UnDefined

Name	Type	Value
current_row	integer	0
deletedName	string	UnDefined
name	string	UnDefined
name	string	UnDefined
p	Painting	UnDefined
row	integer	UnDefined
selectedPainting	Painting	Nil
▼self	PaintingListWindow	PaintingListWindow
▶<AuctioneerOnly>	GridField	GridField
▶<AuctionRole>	RadioList	RadioList
▶<Blank5x9>	PictureGraphic	PictureGraphic
▶<DeletePaintingButton>	PushButton	PushButton

Variables window The Variables window provides a browser that contains the following columns:

Name The variable name.

Type The declared type of the variable.

Value For simple types, the current value of the variable. For class types, the runtime type of the object to which the variable is currently pointing. It can also be NIL for no object or Undefined for not applicable.

Changing the value of a simple variable You can use the Variables window to change the value of a simple variable. Simply edit the value of the variable as shown in the window.

➤ **To change the value of a simple variable**

1. Edit the value.
2. Click the Apply button.

If you decide not to change the value, click the Undo button to revert to the original value.

More information about objects When you need more information about an object, you have two alternatives. First, you can use the expansion arrow in the browser to open the object. This displays the name, types, and values for each of the object's attributes. Second, you can use an Object Inspector to monitor the value of an individual object.

Opening an Object Inspector To open an Object Inspector, double-click the object reference. This opens the Object Inspector window, which displays the object information. The Object Inspector window provides a temporary name for the object, so that if you want to view the same Object Inspector again, you can refer to the particular object by name.

The Object Inspector stays open until you close it by using the window close mechanism provided by your particular window system. After you close the window, the Object Inspector is hidden so you can easily display it again at any time. To display a hidden Object Inspector, use the Inspections menu as described under "“Using the Inspections Menu” on page 646.

As long as the Object Inspector is open, it shows the current values for the same object. Even if the reference in the method changes to another object, the object displayed in the window does not change.

The way you use the Object Inspector depends on the particular class of the object. You use an Object Inspector window to inspect each of the following classes:

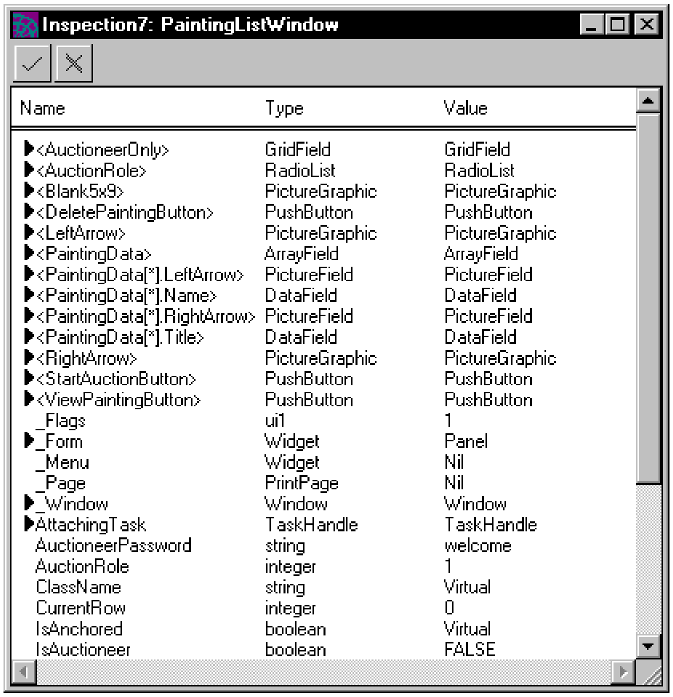
- DataValue subclasses (for example, TextData and IntegerData)
- Array or LargeArray classes
- standard objects (all other classes)

The following sections describe to how use each type of inspector.

Inspecting Standard Objects and Arrays

The Object Inspector provides a browser that displays the name, type, and value of the object's attributes.

Figure 13-4 Object Inspector



Name	Type	Value
▶<AuctioneerOnly>	GridField	GridField
▶<AuctionRole>	RadioList	RadioList
▶<Blank5x9>	PictureGraphic	PictureGraphic
▶<DeletePaintingButton>	PushButton	PushButton
▶<LeftArrow>	PictureGraphic	PictureGraphic
▶<PaintingData>	ArrayField	ArrayField
▶<PaintingData[*].LeftArrow>	PictureField	PictureField
▶<PaintingData[*].Name>	DataField	DataField
▶<PaintingData[*].RightArrow>	PictureField	PictureField
▶<PaintingData[*].Title>	DataField	DataField
▶<RightArrow>	PictureGraphic	PictureGraphic
▶<StartAuctionButton>	PushButton	PushButton
▶<ViewPaintingButton>	PushButton	PushButton
_Flags	ui1	1
▶_Form	Widget	Panel
_Menu	Widget	Nil
_Page	PrintPage	Nil
▶_Window	Window	Window
▶AttachingTask	TaskHandle	TaskHandle
AuctioneerPassword	string	welcome
AuctionRole	integer	1
ClassName	string	Virtual
CurrentRow	integer	0
IsAnchored	boolean	Virtual
IsAuctioneer	boolean	FALSE

The following list defines these columns.

Name The attribute name.

Type The declared type of the attribute.

Value For simple types, the current value of the attribute. For class types, the runtime type of the object to which the attribute is currently pointing. It can also be NIL for no object or Undefined for not applicable.

For attributes with a simple data type, this is all the information you need. For attributes with a class type, the Value column displays the runtime class of the object to which the attribute is currently pointing. You can inspect the object to which the attribute is pointing either by using the expansion arrow to display its attributes or by double-clicking the object reference to open another Object Inspector.

If the object you are inspecting is an array, the columns have a slightly different meaning.

Name The row number for the object in the array.

Type The declared type of the object.

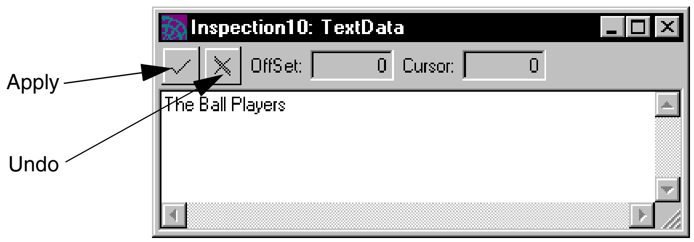
Value The runtime type of the object.

To examine an individual object in the array, you either use the expansion arrow to display its attributes, or you can double-click the object reference to open an Object Inspector.

Inspecting DataValue Objects

The DataValue Object Inspector displays the current value of the DataValue object as a string.

Figure 13-5 DataValue Object Inspector



The Ball Players

Updating the value of a DataValue object A special feature for DataValue objects is that you can change the value from within the inspector.

➤ **To change the value of a DataValue object**

1. Edit the value.

2. Click the Apply button.

If you decide not to change the value, click the Undo button to revert to the original value.

Using the Inspections Menu

The Inspections menu allows you display a hidden Object Inspector or, if the Object Inspector is currently displayed, to move a particular Object Inspector to the front of the window stack.

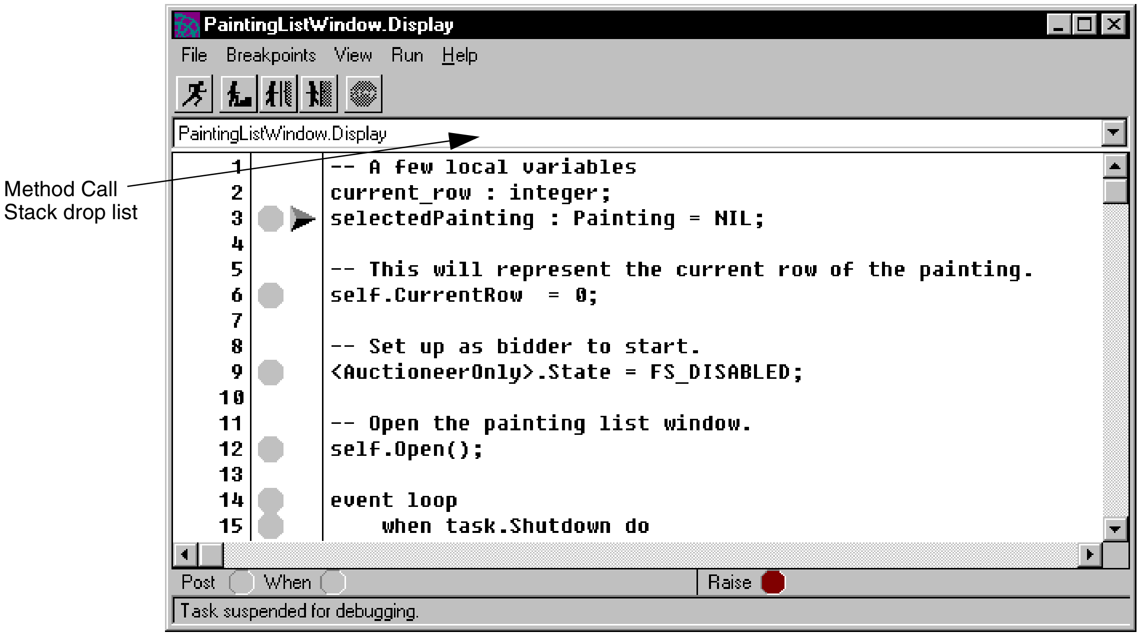
➤ **To view a previously displayed Object Inspector**

1. Pull down the Inspections menu to see a list of the objects you have previously inspected.
2. Choose the command for the object you wish to inspect again.

Viewing the Method Call Stack

To view the call stack for the current method, select the Method Call Stack drop list on the Task window. This drop list displays the complete method call stack for the current task. The first method on the list is the current method. The second method on the list is the method that invoked the current method. The last method on the list is the starting method for the task.

Figure 13-6 Method Call Stack


Method Call Stack drop list
PaintingListWindow.Display
File
Breakpoints
View
Run
Help
PaintingListWindow.Display
Post
When
Raise

```
-- A few local variables
current_row : integer;
selectedPainting : Painting = NIL;

-- This will represent the current row of the painting.
self.CurrentRow  = 0;

-- Set up as bidder to start.
<AuctioneerOnly>.State = FS_DISABLED;

-- Open the painting list window.
self.Open();

event loop
    when task.Shutdown do
```

Task suspended for debugging.

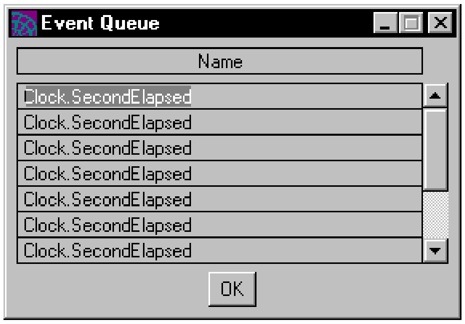
Viewing the Error Stack

To view the error stack for the task, choose the View > Error Stack command on the Task window. This command opens the Error dialog, where you can view the error messages reported for the exceptions raised on the task. (Note that these are the same messages that are produced by the ShowErrors method on the ErrorMgr class. See the <Italic>*Framework Library and AppletSupport Library* for information.)

Viewing the Event Queue

To view the event queue for the task, choose the View > Event Queue command on the Task window. This command opens the Event Queue dialog, which lists the events currently in the queue.

Figure 13-7 Event Queue



Clock.SecondElapsed
Clock.SecondElapsed
Clock.SecondElapsed
Clock.SecondElapsed
Clock.SecondElapsed
Clock.SecondElapsed
Clock.SecondElapsed

The Debugger uses the following format to identify the events in the queue:

classname.*eventname*

The order of the events on the list shows the order in which they are queued.

Working with Multiple Tasks

When an application uses multitasking, the Debugger lets you examine all tasks that are executing concurrently. You can choose which tasks to monitor and which not to monitor. You can control a task's starting state. And you can set breakpoints that take effect only for a single task. A "task breakpoint" takes affect when only when the method is executing as part of the current task.

Task list The Debugger window displays a list of the tasks in the application. Whenever a new task is started, iPlanet UDS adds the task to the list. The name of the new task is indented below the task that started it, providing a task "call stack" for the application.

Icons indicate the current status of each task.

Icon	Description
🏃	Indicates the task is running.
🐞	Indicates the task is suspended.
🧍	Indicates the task is waiting for an event.
🪦	Indicates the task has completed.

You can use the Tasks > Startup Options > Start Suspended command on the Debugger window to make all tasks start in a suspended state. This ensures that each task is brought to your attention, because you will have to start each one.

By default, the first task in an application always starts in a suspended state.

When iPlanet UDS executes a start task statement, the Debugger automatically opens a Task window for the new task. At this point you can:

- run the new task by clicking the Run button in the Debugger window
- leave the task in a suspended state

Hiding and opening a task You can close the Task window without changing the status of the task (it either keeps running or stays suspended). To hide a task, simply close its Task window. iPlanet UDS displays an icon by the task name in the Debugger window. The task continues to run while it is hidden. If the task reaches a breakpoint, a suspended icon in the Debugger window indicates that the task is suspended. As long as the task is not terminated, you can open it again by double-clicking on the task name.

When the task completes or is terminated, iPlanet UDS closes the corresponding Task window (if it is open) and displays an RIP icon by the task name in the Debugger window.

Communication Between Tasks

One of the important issues you need to consider when you are debugging multiple tasks is how the tasks interact with each other. For example, if you are running one task while a concurrent task remains suspended, the running task may reach a point where it is waiting for an event from the suspended task. Because the suspended task cannot post the expected event, the running task will not be able to finish. If there is a problem, you can always check the Debugger window to see the status of each of the tasks in the application.

Setting Task Breakpoints

When you are debugging an application that uses multitasking, you may wish to set statement breakpoints that take effect for the current task only. Normally, the statement breakpoints that you set in the Task window are global breakpoints. Global breakpoints take effect for all tasks in the application. However, the Task window allows you set task breakpoints to take affect for the current task only.

➤ **To set a task breakpoint in the Task window**

1. Scroll to the statement where you want the breakpoint.
2. Shift-click the grey toggle to the left of the statement. A road block icon indicates that the statement now has a task breakpoint.

When the task terminates, all task breakpoints that you have set are automatically removed.

Removing a breakpoint To remove a task breakpoint before the task completes, simply click the road block icon. A grey toggle indicates that the statement no longer has a breakpoint.

Chapter 14

Using the Partition Workshop

This chapter provides conceptual information about distributed applications, libraries, environments, and configurations, and describes how to use the Partition Workshop.

In this chapter, you will learn how to:

- create a configuration
- examine a configuration
- modify a configuration
- combine service objects and partitions
- test a client configuration
- make an application distribution
- make a library distribution

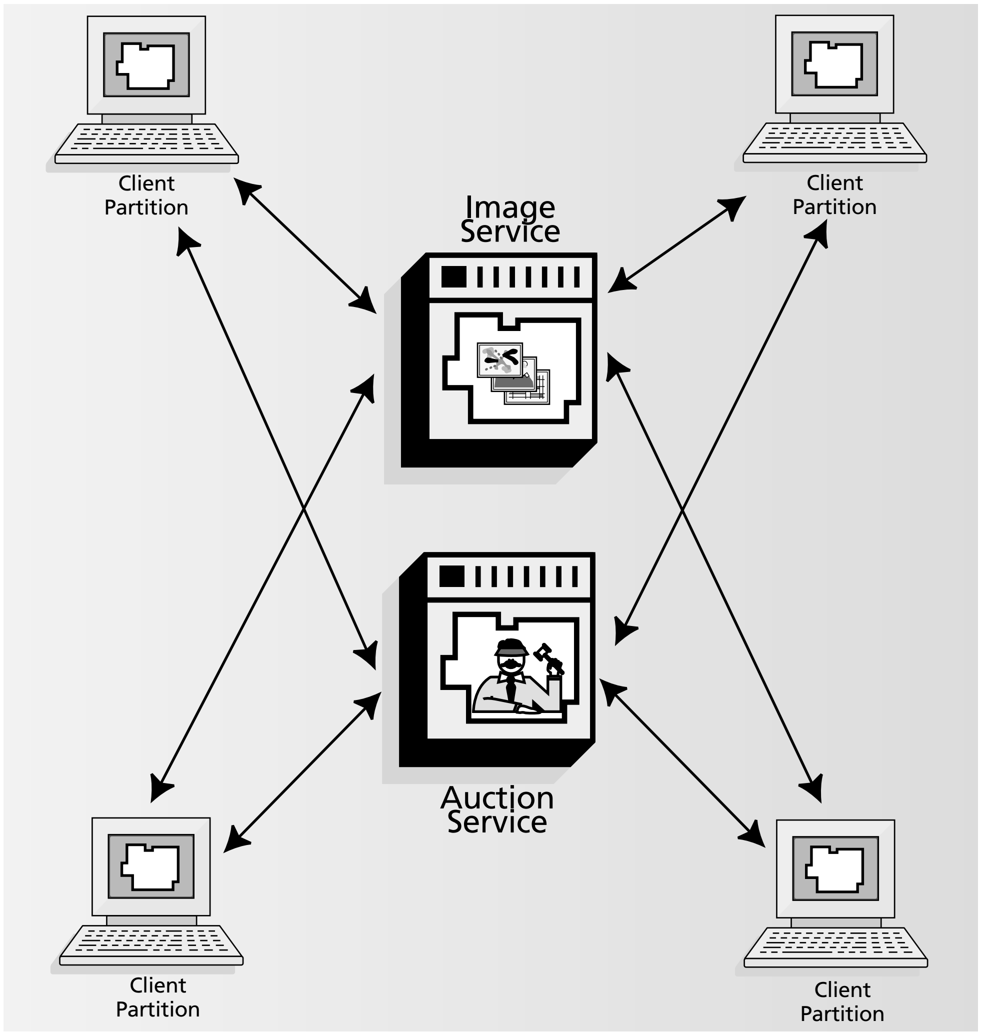
For information on deploying new releases of existing applications, see the *iPlanet UDS Programming Guide*.

About Distributed Applications

To distribute an application, iPlanet UDS divides it into logical sections, called *partitions*. Each partition is an independent process that can run on its own machine.

In iPlanet UDS, a distributed application consists of partitions running on different nodes on a network. The server partitions run on server nodes, which may be serving one client or any number of clients. The client partitions run on client nodes, which may be communicating with any number of servers. Figure 14-1 illustrates the partitions in a distributed application.

Figure 14-1 Partitions in a Distributed Application



The distributed application always runs within a particular environment. Using the Partition Workshop, you partition an application specifically for each environment in which it will run. This partitioning customizes the application for the particular servers and clients on which it will be deployed.

While you are developing an application, you can test it either locally or in the distributed development environment. Typically, you repeat a cycle of testing the application in the distributed development environment and then making appropriate changes.

➤ **To develop and test a distributed application**

1. Design the application and test it locally.
2. Test the application in the distributed development environment.
3. Make appropriate changes.
4. Repeat Steps 2 and 3 until the application is ready for deployment.

When the application has been developed and tested, it is ready for deployment.

➤ **To deploy a distributed application**

1. The system manager defines the deployment environments using the Environment Console.
2. You partition the application for each deployment environment using the Partition Workshop.

   Partitioning an application creates a configuration for each deployment environment.
3. You make an application distribution for each configuration.

   The application distribution is a representation of the application outside of the repository that is used to install the application in an environment.
4. The system manager installs the application distributions in the appropriate environments.

This chapter describes how to use the Partition Workshop to test and modify an application, and to make an application distribution. The *iPlanet UDS System Management Guide* describes how to define environments and install iPlanet UDS applications.

About Libraries

A *library* is a named collection of classes and other component definitions that can be shared by any number of iPlanet UDS applications. iPlanet UDS automatically provides several libraries for your use in developing applications, including Framework, GenericDBMS, and Display. You can also create your own libraries for use within your organization or for distribution or sale to other companies. For example, you could create a library that provides access to an external service, such as a stock market service, or a collection of statistical routines.

A library can be installed in both deployment environments and development environments. When the library is installed in a deployment environment, any number of applications running in the environment can reference it. This provides the ability for multiple applications to share a single library, which makes installation easier and more efficient.

When the library is installed in a development environment, the library can be imported into any of the development repositories in the environment. Then, any projects being developed in those repositories can include the library as a supplier plan. This provides the ability for multiple projects within different development repositories to share a single library.

Compiled libraries The ability to share a library rather than duplicating its definitions provides efficiency gains. The option of providing compiled libraries provides improved performance. A *compiled library* uses C++ code generation to create a compiled shared library. Normally, the libraries running on server machines are compiled. However, by default, all libraries are standard because installation and management of standard libraries is simpler.

Library distribution A *library distribution* is a set of related libraries packaged together for distribution. This provides a convenient way to bundle any number of related libraries together for deployment and installation. When you bundle libraries into a library distribution, you can make a single library distribution to use for deployment.

To create a library, you must define a project that contains the definitions you want to be shared and then configure that project as a library. A project becomes a "library" rather than a project when you include that project in a library distribution. There is a one-to-one correspondence between the *projects* you include in a library distribution and the *libraries* that are available when that library distribution is deployed. Each project in the library distribution gets compiled into a separate shared library.

➤ **To create a library distribution**

1. In the Project Workshop, configure the project as a library.
2. In the Partition Workshop, specify which other projects you wish to bundle with the library project in the library distribution, and indicate the nodes on which they should be installed.

   At this point, you specify which libraries are compiled and which are standard.
3. Make a library distribution.
4. Install the library distribution into one or more iPlanet UDS environments.
5. Within a development environment, import the library definitions into the development repositories. Then, use the libraries as supplier plans for developing iPlanet UDS applications.

This chapter provides information about steps 1 through 3. For information about installing libraries, see the *iPlanet UDS System Management Guide*. For information about creating, deploying, and using libraries, see the *iPlanet UDS Programming Guide*.

Compatibility level A library distribution has a compatibility level based on the compatibility level of the project for which you give the Configure as > Library command. Like an application's compatibility level, the library distribution's compatibility level allows you to release different versions of the same library within a single environment. The rules for when you need to raise the compatibility level of a library distribution are the same as those for an application. See the *iPlanet UDS Programming Guide* for information.

About Environments

An environment is the distributed system on which you run a distributed iPlanet UDS application. An iPlanet UDS environment may consist of one or hundreds of computers. A local- or wide-area network might contain several iPlanet UDS environments. An iPlanet UDS developer or system manager uses the Environment Console to define an environment by describing a set of nodes, each of which corresponds to a particular machine in the environment. Only after an environment has been defined can you partition the application to run in that environment.

There are two kinds of environments in iPlanet UDS: development environments and deployment environments.

Development environment A development environment is an environment in which the development version of iPlanet UDS is installed. You create and test all your iPlanet UDS applications in the development environment. In the Partition Workshop, you can partition your application in the development environment for initial testing of the distributed version of the application.

Deployment environment A deployment environment is an environment in which the runtime version of iPlanet UDS is installed. You can have any number of deployment environments in which you plan to install the production version of your application, and each deployment environment can consist of a different number of nodes, with varying architectures. In the Partition Workshop, you partition your application for each of the deployment environments for final testing. And when your application is complete, you use the Partition Workshop to create an application distribution for each of the deployment environments. An application distribution contains the files necessary for actually deploying the applications.

Simulating deployment environments Because you cannot run the Partition Workshop in the deployment environments (and you may not even have physical access to them), iPlanet UDS simulates the deployment environments by using nodes in your development environment as stand-ins for nodes in the deployment environment. Every node in a deployment environment is assigned to a "testing node," which is the node in the development environment that is used to simulate the deployment node for testing. (Of course, to simulate your deployment environments, your development environment must include at least one of each machine type that will be included in all your deployment environments.)

In the Partition Workshop, you select the environment into which you wish to partition your application. This environment can either be the development environment or a deployment environment; a deployment environment is simulated within the development environment. While you are working in the Partition Workshop, you can examine the node definitions for the environment in which you are partitioning your application. However, you cannot change the node definitions or any other part of the environment definition from the Partition Workshop. The Environment Console or the Escript utility, its command-line counterpart, are the only utilities that allow creation and modification of environments.

Connected environments iPlanet UDS environments can be connected, that is, partitions in one environment can find service objects in other iPlanet UDS environments and access them. See "About Connected Environments" on page 659.

The following section provides information about the properties of a node, which you can examine from the Partition Workshop.

About Nodes

A node is a machine in your network that is capable of running an iPlanet UDS partition. In the Partition Workshop, you can examine the nodes in the environment by viewing the node's properties dialog. You cannot, however, change node definitions from the Partition Workshop.

Every node has the following properties:

Node Property	Description
Name	The name assigned to the node when the environment was defined.
Architecture	The node type.
Testing Node	The name of the node in the development environment used to simulate the current node for testing.
Client	Specifies that you plan to install client partitions on the current node. A node where you plan to install only server partitions is considered a server node.
Use as Model	Specifies that the current node represents a number of identical client nodes.
Use for Testing	Specifies that the node can be used for simulating a node in a deployment environment.
Resource Managers	The list of resource managers available on the current node.
Installed Protocols	The list of communication protocols that the node can use to communicate with other iPlanet UDS nodes.
Installed Libraries	The list of restricted, user-defined external projects installed on the current node.

Each property is discussed in the following sections.

Node Name

The node name for the current node is the name that was assigned to it by the system manager who defined the environment. You cannot change this name from the Partition Workshop.

Node Architecture

The architecture of the node describes the node type. For the most current list of node types, see the *iPlanet UDS System Installation Guide*.

Windows 95 is client node only All platforms that iPlanet UDS supports are available for both client and server nodes with the exception of Windows 95, which is a client node only.

Testing Node

The Partition Workshop allows you to test your configuration by using the testing nodes in your development environment to simulate your deployment environment. When the system manager defined your deployment environments, she assigned each deployment node to a testing node in your development environment. iPlanet UDS uses this testing node (instead of the deployment node) whenever you test a configuration. The testing nodes allow the Partition Workshop to simulate the deployment environment.

On a given node's properties dialog, the testing node is the node in the development environment that is being used to simulate the current node.

Client Node

The client partition in your configuration contains the user interface for the application (if there is one) and the startup code. This partition allows the end user to run the application from his or her workstation. The client partition may also include user-visible service objects.

With the exception of Windows 95 nodes, which support only client partitions, a client node can support both client and server partitions. The Partition Workshop automatically assigns your client partitions to client nodes, however, it is possible in the Partition Workshop to manually assign a client partition to a server node. See "Modifying Node Assignments" on page 699 for information.

Use as Model

iPlanet UDS provides a special option for client nodes that lets the system manager define one node to represent multiple client nodes. This features enables the system manager to avoid having to add identical nodes to the environment for every specific physical node. For example, if the site has 500 identically configured (from iPlanet UDS's perspective) PCs, the system manager can define a single model PC node to represent them.

When the Use as Model toggle is turned on for a node, you can assign the client partition to it and have that single assignment represent a number of identical client nodes.

Use for Testing

When the system manager is setting up your development environment, she can specify which nodes in the development environment can be used for testing (that is, simulating nodes in a deployment environment) and which cannot be used for testing. If the Use for Testing toggle is on for a node, the node can be used for simulating a node in deployment configuration.

Resource Managers

A resource manager is a database management system that can be accessed from iPlanet UDS. When defining a node, the system manager assigns a name to each of the external resource managers on the node that developers will be accessing from iPlanet UDS applications. You need to use the external resource name in the properties dialog for a DBSession or DBResourceMgr service object. When you partition a service object, you use the resource manager name to identify the particular DBMS installation used for the service object in that configuration.

Installed Protocols

For the current node, you can view a list of the communication protocols that the node can use to communicate with other iPlanet UDS nodes. Your system manager selects the installed protocols for the node when he or she is setting up the environment. See the *iPlanet UDS System Management Guide* for complete information on the installed protocols for a node.

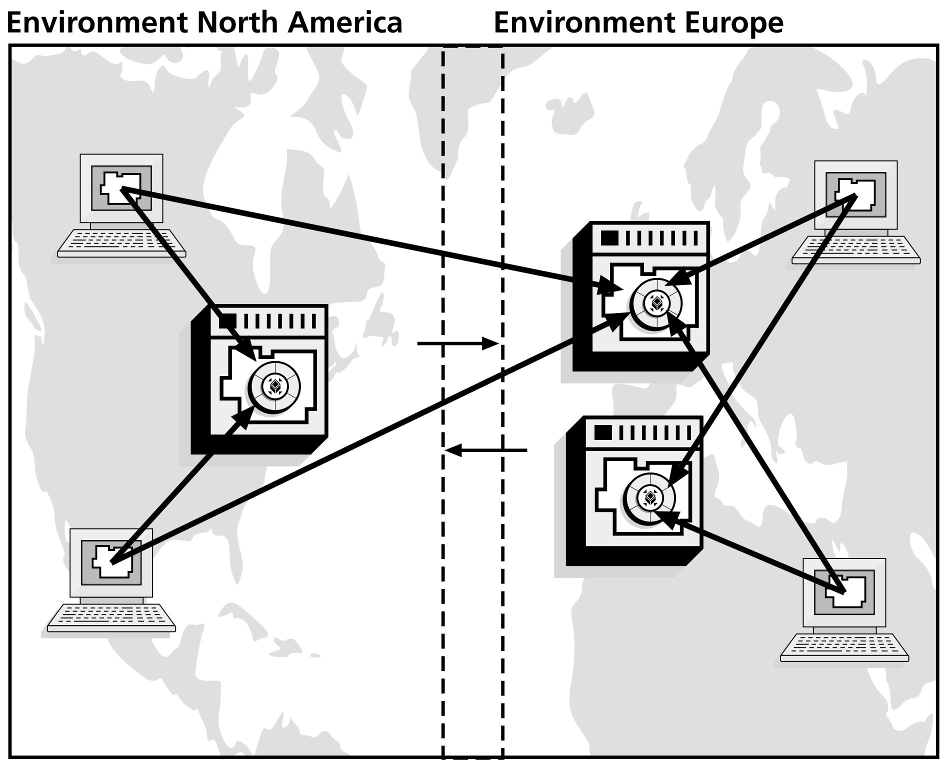
Installed Libraries

For the current node, you can view a list of the restricted, user-defined external libraries that are installed on the node. Partitions that use restricted external libraries must be assigned to the nodes where the appropriate external libraries are actually installed. See *Integrating with External Systems* for information about external libraries.

About Connected Environments

If your system manager has connected your iPlanet UDS environments, the environment in which you are partitioning a project may have one or more other environments connected to it. Figure 14-2 illustrates connected environments.

Figure 14-2 Connected Environments



When the current environment is connected to other environments, you can use service objects from other environments in the current configuration. There are two ways to use a service object that is in a different environment:

- You can share an existing service object in a connected environment rather than starting that same service object in the current environment.

  You do this by creating a reference partition (described under "Making a Reference Partition" on page 689) and by setting the environment search path for the service object in the reference partition (described under "Modifying a Service Object Definition" on page 695).

- You can provide a list of service objects in connected environments to be used for failover for a service object in the current environment.

  You do this by specifying an environment search path for the service object (described under "Modifying a Service Object Definition" on page 695).

If your environment is not connected to other environments, you can only use service objects within the current environment. To see if a given environment is connected to any other environments, you must use the Environment Console or Escript. For information on how to do so, see the *iPlanet UDS System Management Guide*.

See the *iPlanet UDS Programming Guide* for complete information on connected environments and for instructions on using reference partitions and failover with connected environments.

About Application Configurations

To distribute an application, iPlanet UDS divides the application into logical sections, called partitions. Each partition is an independent process, which can run on its own machine. A *configuration* customizes your application for the particular hardware and software in that environment by assigning the partitions to nodes in the environment. A configuration also allows you to replicate certain partitions for load balancing or failover. The result is a distributed application from which you can generate the files to be installed in the environment.

There are two basic kinds of application configurations: a client configuration and a server configuration.

Client configuration A *client configuration* defines a client application, which is an application that the end user can run from his or her workstation. A client configuration contains one client partition, which includes the user interface for the application (if there is one) and the startup code. A client configuration may also contain one or more server partitions.

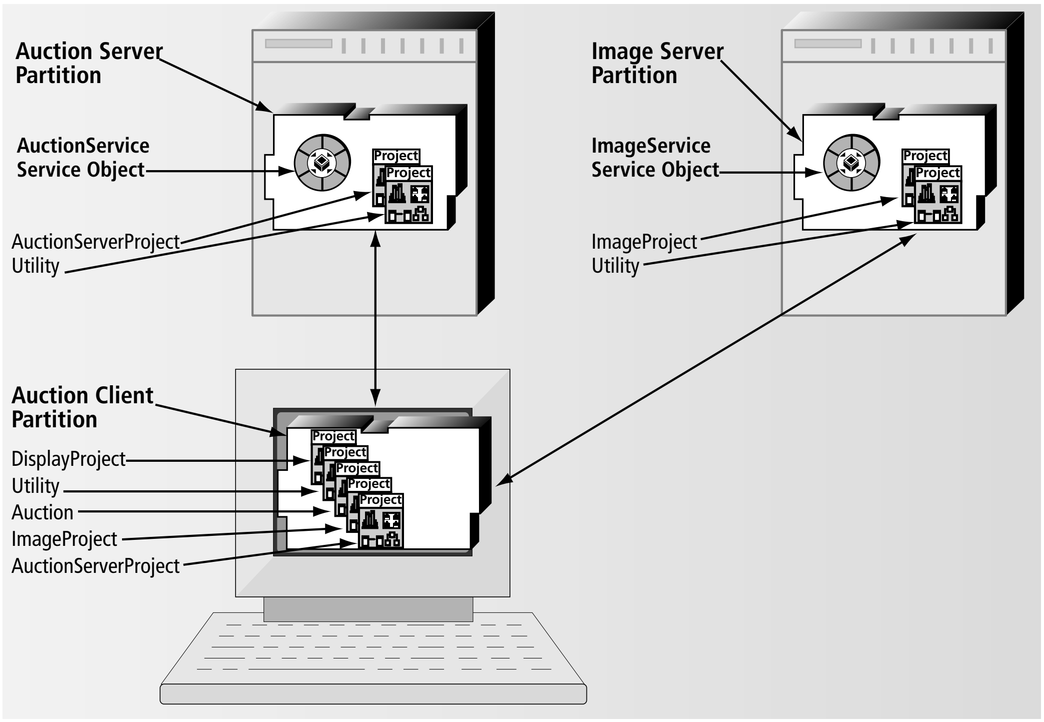
Applets In the Partition Workshop, any client configuration can be configured as an applet, rather than as a client application. An *applet* is an application that can be launched from another application using the AppletSupport library, but that cannot be run as an independent client application. Except for this limitation in the way it can be started, an applet is the same in every way as a client application. For complete information on applets, see the *iPlanet UDS Programming Guide*.

Server configuration A *server configuration* defines an iPlanet UDS server application, which provides processing for one or more client applications. A server configuration has one or more server partitions, and no client partition.

Relationship between partitions and projects As described under "Supplier Plans" on page 209, an application consists of a main project and all of its supplier plans. When iPlanet UDS partitions your application, it assigns all of the service objects in the main project and its supplier projects to partitions. If any class within a project is needed by a service object on a partition, iPlanet UDS also includes that project definition on the partition. Therefore, a single project may appear on any number of partitions.

For example, in the Art Auction application, the `Painting` class needs to be available on the client partition because it is displayed to the end user on the window; it also needs to be available on the server partition where the Image Server is running because that is where the paintings are stored. Therefore, the project that defines the Painting class needs to be on both partitions. Figure 14-3 illustrates the relationship between partitions, service objects, and projects:

Figure 14-3 Relationship between Partitions and Projects



Relationship between partitions and libraries The supplier libraries for the application must also be installed on the partitions that need to access them. However, the application configuration does not include the supplier libraries (only the supplier projects). Therefore, you must make a separate library distribution that contains the libraries needed by the application, and install the library distribution along with the application distribution. See "About Library Configurations" on page 673 for information.

Automatic partitioning iPlanet UDS automatically partitions your application for each of your environments. Most of your service objects can run only on a certain node. Therefore, iPlanet UDS automatically assigns the service objects to specific partitions and assigns the partitions to the nodes in the environment that have the appropriate resources and capabilities. For further details on automatic partitioning, see "Default Configuration" on page 672.

After iPlanet UDS partitions the application, you can examine it and make adjustments. If necessary, you can assign service objects to different partitions or assign partitions to different nodes. For partitions that contain replicated service objects, you can replicate the partitions to provide load balancing or failover.

When you examine a partition in the Partition Workshop, you see only the service objects assigned to the partition, not the projects. However, iPlanet UDS ensures that the project definitions needed by each service object are always included on the partition. If you move a service object, the projects it needs are moved along with it.

(Note that you can use the ShowApp command in Fscript to see which projects are included on a partition. See the *Fscript Reference Guide* for information on the ShowApp command.)

Testing a configuration After you adjust your configuration, you can test it by running the partitioned application in the environment. This lets you test the distributed application as it will appear to end users when it is installed.

Deploying applications The completed configuration is your "distributed application." When your configuration is ready, you can generate the *distribution*, that is, the files necessary to install the application for production deployment. The Partition Workshop provides a Make Distribution command for this purpose. See "Making an Application Distribution" on page 716 for information.

About Partitions

A partition is a logical section of an application that represents either the client portion of the application or one of the servers. When iPlanet UDS partitions an application, it assigns all the service objects in the application to partitions; these are the *logical partitions*. It then assigns the logical partitions to appropriate nodes in the environment; these are the *assigned partitions*.

The Partition Workshop shows both the logical partitions into which your application is divided and the assigned partitions that represent the processes that will be installed in the environment. The following sections provide information about the properties of logical and assigned partitions.

Logical Partitions

When iPlanet UDS partitions your application, it divides the application into one client partition and any number of server partitions.

Client partition A client partition contains the user interface for the application (if there is one) and the application startup code. In addition, if the application contains any service objects with user visibility, iPlanet UDS will put them on the client partition if the client nodes in the environment can support them. If you used the Configure as > Server command to create a server configuration, iPlanet UDS does not create a client partition for the application.

Server partition A server partition contains one or more service objects and the projects accessed by the service objects. There are two types of server partitions: replicated and non-replicated.

Replicated server partition A replicated server partition is a partition that contains a service object that was defined as being replicated for load balancing or failover. When a logical partition is replicated, you can assign it to any number of nodes in the environment. For each assigned partition, you can specify a startup replicate count, as described under “Assigning Partitions” on page 699.

Router partition When the service object in a partition is replicated for load balancing, iPlanet UDS automatically creates an extra partition called a router partition. The purpose of a router partition is to route the traffic between the partitions that are load balancing work for the service object. Although the router partition is usually assigned to the same node as one of the server partitions that it is managing, it can be on any server node in the environment.

If the service object is replicated for both failover and load balancing, iPlanet UDS replicates the router partition for failover and replicates the server partitions for load balancing. As a result, if the first router partition fails, the second router partition can take over and manage the partitions that are sharing the work load, and so on for each replicated router partition. See the *iPlanet UDS Programming Guide* for further information about the router.

Do not add service objects to router partition Although the router partition appears in the Partition Workshop as a standard partition, we strongly recommend that you do not move any service objects onto this partition.

Non-replicated server partition A non-replicated server partition is a partition containing service objects that were not defined as being replicated. When a logical partition is non-replicated, you can assign only one enabled partition in the environment. Other copies you make of the non-replicated server partition will be automatically disabled, as described under "Assigned Server Partition Properties" on page 670.

In the Partition Workshop, you can change whether a partition is replicated or not by modifying the definition of the service object that it contains. See "Modifying a Service Object Definition" on page 695 for information.

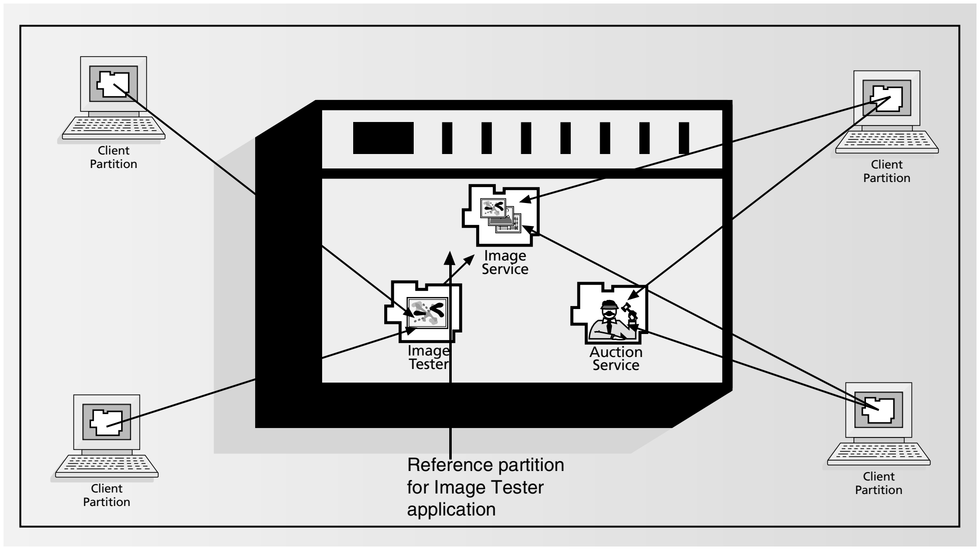
Reference partition A special kind of logical partition is a *reference partition*. A reference partition allows you to share an existing service object in another application (or in another environment) rather than starting a new instance of the server partition in the current application.

To share a service object between applications, you first deploy the service object in one application. The application that deploys the service object can be either a server application or a client application—it makes no difference.

Then, in the other applications that need to access that deployed service object, you create a *reference partition*. Instead of containing a new service object, the reference partition points to the existing service object that was originally deployed as part of the first application.

Sharing services objects in the current environment Reference partitions allow you to create business services that are shared by any number of applications in the current environment. For example, if you want your application to interact with an image server that is already provided by another application in the environment, you can include the project that defines that service object as a supplier and then create a reference partition that points to the existing service object. This way your application will be sharing the existing service object with any other applications that are using it, rather than creating a new instance of the service object. Figure 14-4 illustrates a reference partition:

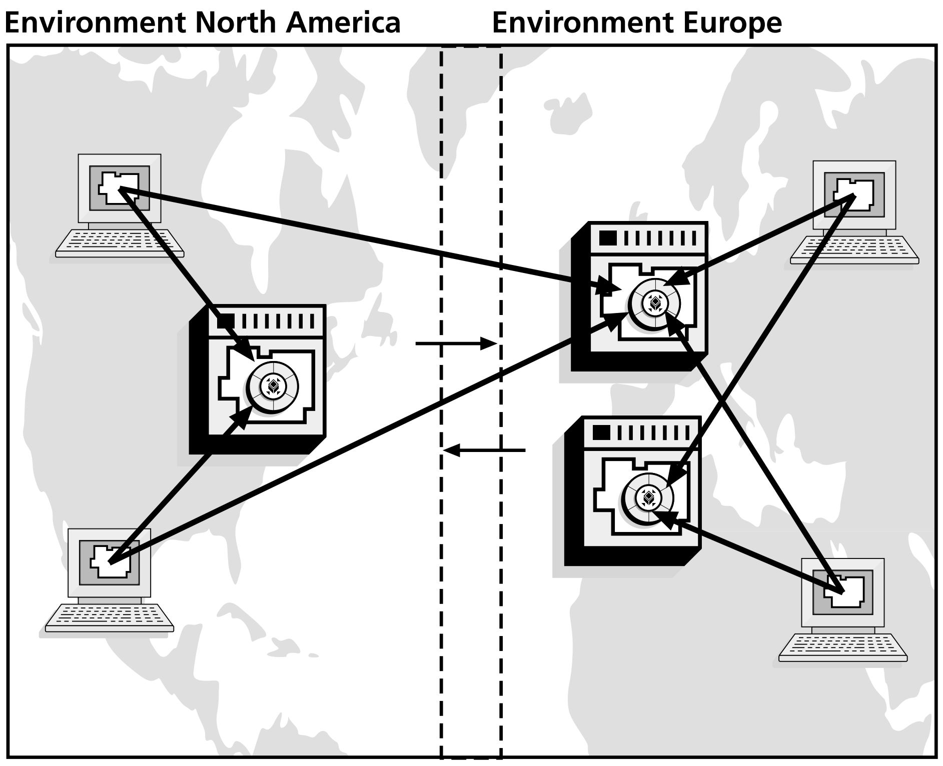
Figure 14-4 Reference Partition for Current Environment



Sharing services objects between connected environments If your deployment environments are connected, you can create business services that are shared across environments. For example, you may have a service object that can run only in one environment, such as a service object that uses a specialized satellite feed or stock ticker that can be accessed through a callout to an external service running in one location. Or, imagine you have a service object that handles personnel data. This service object only resides in the headquarters environment because that is where the personnel database is. All other applications installed in their own local environments must access the personnel database located at headquarters.

Applications in other environments can access the specialized service object in the connected environment by using a reference partition. The following figure illustrates:

Figure 14-5 Reference Partition for Connected Environment



The advantages of using reference partitions include:

- modularity

  The ability to share a single service object between multiple applications means that you need to create and manage only that single service object. Without reference partitions, you would need to create more than one version of the service object.

- efficient use of resources

  Only one service object needs to be running in order to provide services needed by multiple applications. Without reference partitions, you would need to run the service object within every application that needs its services.

For complete information about reference partitions, see the *iPlanet UDS Programming Guide*.

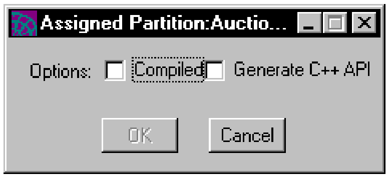
Assigned Partitions

An assigned partition represents a client or server partition that will be installed on a particular node in the environment. The following two sections describe the different properties available for the assigned client and server partitions.

Assigned Client Partition Properties

Figure 14-6 shows the properties available for an assigned client partition:

Figure 14-6 Assigned Partition Properties Dialog (Client Partition)



Compiled property A client partition can either be standard or compiled. A *standard partition* runs with the iPlanet UDS runtime system using an image repository and an iPlanet UDS interpreter (see the *iPlanet UDS System Management Guide* for definitions of these terms). A *compiled partition* is a partition for which you are going to use C++ code generation to create a compiled version. A compiled partition can provide the advantage of improved performance, which is especially significant for high-volume partitions.

When to use compiled partitions Compiled partitions provide improved performance for process-intensive partitions, which are typically server partitions. Because client partitions are not typically process-intensive, you may not see the same performance gains for client partitions that you see for server partitions. One good strategy might be to experiment to find out what works most effectively for your particular application.

Because the compilation process can take some time, you may wish to use standard partitions when you are developing and testing, and request compiled partitions only when you are making the distribution for a deployed application. By default, all partitions are standard.

Restrictions for compiled client partitions Compiled client partitions are not supported on all the client platforms. The following table summarizes:

Platform	Client Code Generation Available?	Autocompile Support?
Windows 95/NT	Yes	Yes
UNIX	Yes	Yes
OpenVMS	Yes	Yes

Windows NT and Windows 95 compiled clients Because Windows 95 code and Windows NT code are binary compatible, iPlanet UDS compiled client partitions built on these two operating systems are actually interchangeable. For purposes of administration and processing throughput, we recommend that system administrators configure Windows NT machines for the auto-compile service to build compiled Windows 95 clients. However, for customers who do not have Windows NT machines available, the AutocompileSvc_Part2 can actually be placed onto a Windows95 node. See the *iPlanet UDS System Management Guide* for information on setting up the auto-compile service. See "Auto-Compile Option" on page 723 for information on using automatic compilation for Windows 95 compiled clients.

Compiled clients and applets Compiled client partitions cannot be launched by the iPlanet UDS Launcher. Therefore, you cannot mark a client partition as an applet in the Logical Partition dialog, and mark the same partition as Compiled in the Assigned Partition dialog. See "Making an Applet" on page 694 for information about making client applications into applets.

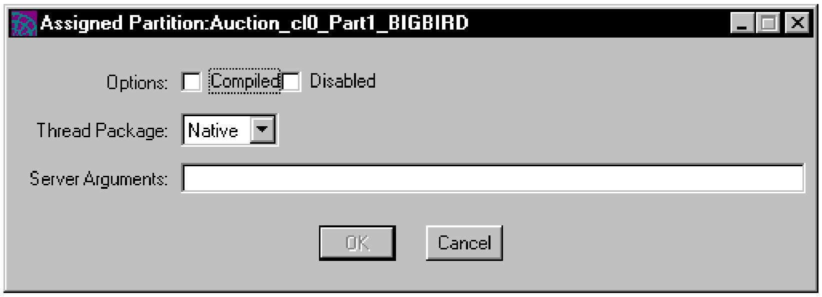
See "Making an Application Distribution" on page 716 for information about creating the distribution for compiled partitions. See the *iPlanet UDS System Management Guide* for information about installing compiled partitions.

Generate C++ API property If you are planning to access your application using C++, you must generate a C++ API for each client partition that needs one. To generate the C++ API for a client partition, on each client node for which you want the API generated, you must turn on both the Compiled and Generate C++ API properties for the assigned partition. When you later use the automatic compilation feature of the Make Distribution command or the `fcompile` command to compile the partitions, iPlanet UDS generates and compiles the files for the C++API. See *Integrating with External Systems* for step-by-step instructions for generating a C++ API for an iPlanet UDS application.

Assigned Server Partition Properties

Figure 14-7 shows the properties available for an assigned server partition.

Figure 14-7 Assigned Partition Properties Dialog (Server Partition)



Assigned Partition:Auction_cl0_Part1_BIGBIRD

Options: ☐ Compiled ☐ Disabled

Thread Package: Native

Server Arguments:

The exact properties available on this dialog depend whether or not the server partition is replicated.

Compiled property A server partition can either be standard or compiled. A *standard partition* runs with the iPlanet UDS runtime system using an image repository and an iPlanet UDS interpreter (see *iPlanet UDS System Management Guide* for definitions of these terms). A *compiled partition* is a partition for which you are going to use C++ code generation to create a compiled version. A compiled partition can provide the advantage of improved performance, which is especially significant for high-volume partitions.

When to use compiled partitions Compiled partitions provide improved performance for process-intensive partitions, which are typically server partitions. One good strategy might be to experiment to find out what works most effectively for your particular application.

Because the compilation process can take some time, you may wish to use standard partitions when you are developing and testing, and request compiled partitions only when you are making the distribution for a deployed application. By default, all partitions are standard.

See "Making an Application Distribution" on page 716 for information about creating the distribution for compiled partitions. See the *iPlanet UDS System Management Guide* for information about installing compiled partitions.

Disabled property Every assigned server partition is either enabled or disabled. An enabled server partition starts up automatically when the application starts. Note that if the auto-start property for the individual service objects on the enabled partition is turned off (it is on by default), the enabled partition will not automatically start up. See the *iPlanet UDS Programming Guide* for complete information on automatic startup for service objects.

Replicated partitions For a replicated partition, at least one copy of the partition must be enabled. By default, all assigned replicated partitions are enabled. However, you can disable an individual replicated partition, if appropriate.

Non-replicated partitions For a non-replicated partition, one and only one copy of the partition must be enabled. If you enable one copy of a non-replicated partition, the previously enabled copy will be automatically disabled. The only way to disable an enabled non-replicated partition is to explicitly *enable* another copy of the partition.

A disabled partition does not start up automatically when the application starts, but simply provides an extra copy of the partition for installation that you can use for manual backup. If no other copies of the logical partition are currently running, you can start the disabled partition explicitly from the iPlanet UDS Environment Console.

Service objects on disabled partitions not auto-started Disabled partitions by definition are not auto-started. Therefore, if a service object on a disabled partition is defined as "auto-started" using the "(a)" feature in the service object's environment search path, it will not be auto-started on the disabled partition. See the *iPlanet UDS Programming Guide* for complete information on automatic startup for service objects and on using the environment search path to auto-start them.

Thread Package property The Thread Package property specifies the thread package used by the partition. The default thread package depends on the particular platform (described in the *iPlanet UDS System Installation Guide*). If you do not want to use the default thread package, you can use the Thread Package property to specify one of the other thread packages supported for the particular platform. See the *iPlanet UDS System Installation Guide* for information about which thread packages are supported for each platform.

Server arguments The server arguments for server partitions specify the startup flags to use for the assigned partition. These include the `-fl` flag to specify filter settings for the partition's log messages and the `-fm` flag to specify memory use for the partition. You can also include your own application-specific flags. Note that these take effect only when the partition is started using the Environment Console or Escript; these arguments are ignored when the partition is starting manually using the `ftexec` command.

Replication Count property For replicated server partitions, you can specify the number of replicates for the partition on each node where it is enabled. When the application starts, iPlanet UDS automatically starts the specified number of replicates of the partition, providing the automatic load balancing or failover for the service objects on that partition. Note that when the system manager installs the application, he or she can specify a different value for the number of automatic startup replicates.

Default Configuration

When iPlanet UDS partitions an application, it assigns all the service objects in the main project and all its supplier projects to logical partitions.

Client partition If the application has a client (that is, if you used the Configure as > Client command to create the configuration), iPlanet UDS creates a client partition to contain the user interface for the application and the application startup code. In addition, if the application contains any user-visible service objects, iPlanet UDS will put them on the client partition if the client nodes in the environment can support them. The client partition is then assigned to every client node in the environment.

Server partitions If a service object has environment visibility, iPlanet UDS assigns it to a server partition. If a service object has user visibility and cannot run on the client partition, iPlanet UDS assigns it to a different server partition than the one that contains the environment-visible service objects. Each server partition is then assigned to an appropriate node in the environment. If the partition requires a database resource manager or a restricted C project, iPlanet UDS assigns the partition to the node where the required resource or library is installed. If you have previously specified a default node and/or excluded nodes for the configuration (described in "Configuration Properties" on page 673), this will be taken into account when the application is partitioned.

By default, iPlanet UDS assigns all compatible service objects to the same partition. However, when you are ready to deploy the application, you may need to move some of the service objects to partitions on different nodes in the deployment environment. You should do this before making the application distribution for the deployment environment. See "Modifying a Client or Server Configuration" on page 686 for information on how to make new partitions and move service objects.

Unassigned service objects If you have any service objects in the application that cannot be supported in the current environment (for example, because a DBMS resource manager service object is defined for an external manager that is not present in the environment, or because a restricted 3GL project has not been installed in the environment), these will be unassigned. To run your application in this environment, you must either update the service object definition or the system manager must update the environment definition so the service object can be assigned to a partition. You can then assign the partition to the appropriate node.

Replication for load balancing and failover iPlanet UDS does not automatically replicate your service objects for load balancing or failover. Instead, in the Partition Workshop, you must replicate the partitions to which the service objects are assigned. You can replicate a partition either by assigning it to an additional node (as described under "Assigning Partitions" on page 699) or by setting the replication count for an individual assigned partition on a single node (described under "Setting Assigned Partition Properties" on page 700).

For each service object that is replicated for load balancing, iPlanet UDS automatically creates a router partition. The router partitions are assigned to the default server node for the environment.

Configuration Properties

For the configuration in general, you can set the following two properties:

Property	Description
Default node	iPlanet UDS automatically assigns all partitions that can run on it to the default node.
Excluded nodes	iPlanet UDS will not use these nodes in the configuration.

Changing the node settings causes the application to be automatically repartitioned after you click the OK button on the properties dialog.

About Library Configurations

A library distribution consists of one or more projects that you bundle together using the Partition Workshop. When it is time to make the library distribution, you must take one of the projects it will contain and configure it as a library with the Configure as > Library command. The project you choose to configure provides the name and compatibility level for the library distribution as a whole.

A *library configuration* consists of projects assigned to nodes in the environment. The default configuration places the project you configured as a library on all nodes in the environment. When you modify the configuration, you can add other projects to it and specify the nodes on which the libraries will be deployed.

When you are examining a library configuration, the left side of the Partition Workshop shows a list of the projects included in the library configuration and the right side of the Partition Workshop shows the assigned libraries, indicating the specific nodes in the environment on which each project will be installed as a library.

Standard or compiled libraries For assigned libraries, you can specify whether the library on a given node is standard or compiled. The option of providing compiled libraries provides improved performance. A *compiled library* uses C++ code generation to create a compiled shared library. Normally, the libraries running on server machines are compiled. However, by default, all libraries are standard because installation and management of standard libraries is simpler.

Compiled suppliers for compiled libraries If a given library has supplier libraries, you must be sure to set the compilation options correctly for the main library and its suppliers. Once you designate a given library on a particular node as compiled, you must ensure that all its supplier libraries are also compiled. For a standard library, the supplier libraries can either be standard or compiled.

Compiled libraries for compiled partitions When deciding whether or not a library is compiled, you also need to take into account whether or not the partition that will be using the library is compiled. If the partition is compiled, the library that it accesses must also be compiled. If the partition is standard, the library that it access can be either standard or compiled.

Configuration property A library configuration has only one property:

Property	Description
Excluded nodes	iPlanet UDS will not use these nodes in the configuration.

Using the Partition Workshop

There are two ways to enter the Partition Workshop. First, if you wish to test a client application in a distributed environment, you can use the Partition… command in the Repository or Project Workshop. Second, when you are ready to deploy a client application, a server application, or a library, you can use the Configure as command in the Project Workshop.

The Plan > Run > Partition… command in the Repository and Project Workshops opens the Partition Workshop, allowing you to run a client application using any of its configurations. See "Testing a Client Configuration" on page 712 for information about using the Partition Workshop to test a client application.

The File > Configure as command in the Project Workshop creates a configuration for the current project (or opens the existing configuration for the current project if there is one). You can then modify the configuration and, when ready, make the final distribution for deployment. You can also open and modify the configurations for other environments. See "Creating a Configuration" on page 677 for information about using the Configure as command.

To use the Partition Workshop, you must be running the iPlanet UDS Workshops in distributed mode. If you are running in standalone mode, the Partition… and Configure as commands will not be available.

Current environment is locked When you use the Partition Workshop for a given environment, iPlanet UDS locks the environment so that no one using the Environment Console or the Escript utility can modify the environment definition. Therefore, you should be careful not to keep the environment locked for long periods of time by leaving the Partition Workshop open unnecessarily.

The Partition Workshop Window

For a client or server configuration, the Partition Workshop window displays the logical and assigned partitions in the environment. For a library configuration, the Partition Workshop displays the assigned libraries in the environment. Figure 14-8 illustrates the Partition Workshop and Figure 14-9 shows the Partition Workshop toolbar icons.

Figure 14-8 Partition Workshop

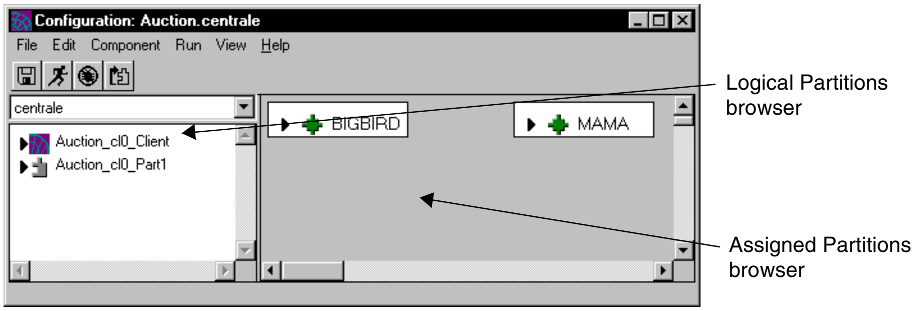
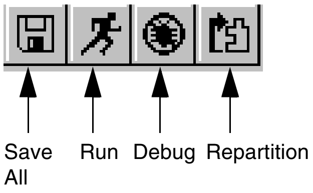

Configuration: Auction.centrale
File Edit Component Run View Help
centrale
Auction_cl0_Client
Auction_cl0_Part1
BIGBIRD
MAMA
Logical Partitions browser
Assigned Partitions browser

Figure 14-9 Partition Workshop Toolbar



Access to Other Workshops

From the Partition Workshop, if you are working with a client configuration, you can access the Debugger by clicking the Debugger button or selecting the Run > Debug command.

Leaving the Partition Workshop

To leave the Partition Workshop, use the File > Close command to close the workshop. This command closes only the current workshop.

Creating a Configuration

The Configure as command creates a configuration, and determines the configuration type: client, server, or library.

The Configure as > Client command creates a client configuration, which defines a client application, an application that the end user can run from his or her workstation. A client configuration contains one client partition, which includes the user interface for the application (if there is one) and the startup code. A client configuration may also contain one or more server partitions.

Applets In the Partition Workshop, any client configuration can be configured as an applet, rather than as a client application. An *applet* is an application that can be launched from another application using the AppletSupport library, but that cannot be run as an independent client application.

Because the client configuration defines a client application, you can use the Partition Workshop to test or debug the client configuration. Even applets can be tested or debugged using the Partition Workshop.

The Configure as > Server command creates a server configuration, which includes one or more server partitions. A server configuration defines an iPlanet UDS service, which provides services for one or more client applications. A server configuration has one or more server partitions, and no client partition.

Because the server configuration defines a server that cannot run on a client, you cannot run or debug it from the Partition Workshop, even if the project that was configured defines a start class and method.

The Configure as > Library command creates a library configuration, which consists of projects assigned to nodes in the environment. The default configuration places the project you configured as a library on all nodes in the environment. When you modify the library configuration, you can add other projects to it and specify the nodes on which the libraries will be deployed.

Because the library configuration defines sets of libraries, not applications, you cannot run or debug it from the Partition Workshop, even if the project that was configured defines a start class and method.

Using the Configure as Command

You can use the Configure as command to create a new configuration or to open an existing configuration.

➤ **To create a configuration or open an existing one**

1. In the Project Workshop, select the project you wish to configure, or whose configuration you wish to open.
2. Choose the File > Configure as command.
3. On the Configure as submenu, select the configuration type you wish to create or open: Client, Server, or Library.

   The Partition Workshop opens.
4. In the Partition Workshop, select the environment from the environment drop list.

Selecting an environment The environment drop list shows all the environments in your environment repository (see the *iPlanet UDS System Management Guide* for information about the environment repository and about defining environments). Note that if any environments are currently locked by the Environment Console or the Escript utility, you will not be able to use them.

Automatic partitioning If you select an environment that does not yet have a configuration, iPlanet UDS automatically partitions the application or libraries. The automatic partitioning may take some time.

Incremental partitioning If you select an environment that already has a configuration and you have changed any of the projects in the application in such a way as to effect the configuration, iPlanet UDS automatically repartitions the application. This automatic repartitioning is an *incremental* partitioning, that is, iPlanet UDS only changes assignments that have become invalid, and does not make new assignments. For example, if you have added new client nodes to the environment, the incremental partitioning will not assign client partitions to the new nodes. To force a complete repartitioning, you can use the File > Repartition command (described under "Repartition Command" on page 704).

When the partitioning (or repartitioning) is complete, iPlanet UDS displays the current configuration.

Examining a Configuration

To examine a configuration, you can start from the Project Workshop or from the Partition Workshop.

Configure as command From the Project Workshop you can examine the configuration for any project in your workspace. In the Project Components browser, select the project whose configuration you wish to examine and then choose the appropriate File > Configure as command (see "Using the Configure as Command" on page 678). When the Partition Workshop opens, select the environment from the environment drop list.

From the Partition Workshop, you can examine the configuration for the current project in any environment in your environment repository. Simply select the appropriate environment from the environment drop list.

Read-only workspace If your workspace is open for reading only, you can use the Partition Workshop to examine a configuration, test it, and make temporary modifications to it. However, since you cannot save your workspace, any changes you make to the configuration are only temporary.

When you enter the Partition Workshop, the workshop displays the configuration that was open the last time you used the workshop (or the configuration for the active environment if the last configuration is unavailable). If it is the first time you have opened the Partition Workshop, the workshop displays the configuration for the active environment. If you wish to select another configuration, you can select the environment name from the environment drop list.

The following sections provide detailed information on examining the configuration components in an application configuration, examining the configuration components in a library configuration, and examining the configuration properties (applies to both application and library configurations).

Examining an Application Configuration

In an application configurations, you can examine the following:

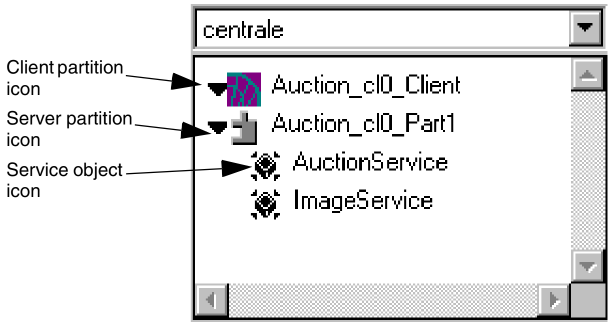
- the logical partitions
- the nodes in the environment
- the assigned partitions on specific nodes in the environment

Examining the Logical Partitions

The Logical Partition browser portion of the Partition Workshop, shown in Figure 14-10, displays a two-level hierarchy that lists the names of the logical partitions for the environment and the service objects contained in each partition.

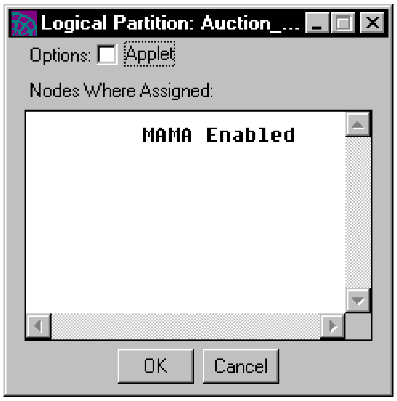
Logical partition icons For logical partitions, the browser displays the logical partition names. Icons indicate whether the partition is a client partition or a server partition. You can identify a router partition by its name.

Figure 14-10 Logical Partition Browser



Double-click partition name to view assignments To view a list of the nodes to which the partition is assigned, double-click on the partition name. A dialog appears, as shown in Figure 14-11, that lists the nodes where the partition is assigned and indicates whether the partition is enabled or disabled.

Figure 14-11 Logical Partitions Dialog



```
MAMA Enabled
```

Examining a service object For service objects, the Logical Partition browser displays the service objects' names. To examine the service object definition, double-click on the service object name. The Service Object Properties dialog opens, displaying the full definition of the service object.

Reference partitions If the service object is in a reference partition, the Service Object Properties dialog does not display the full definition of the service object. Instead, the dialog displays the name of the referenced application and referenced partition. (If the service object in the reference partition is in another environment, the Service Object Properties dialog shows the environment search path.)

To display the full definition of the service object, you must examine the configuration for the project where the service object was originally defined.

Unassigned service objects If there is a mismatch between the service objects defined in the application and the external managers or installed C projects provided by the environment, iPlanet UDS will not be able to assign all the service objects to logical partitions. In this case, the unassigned service objects will be displayed on the "Unassigned" list. Because you cannot run the application with unassigned service objects, you must fix this problem either by modifying the service object definition or having the system manager modify the environment definition. See "Modifying a Service Object Definition" on page 695 for information on modifying the service object definition.

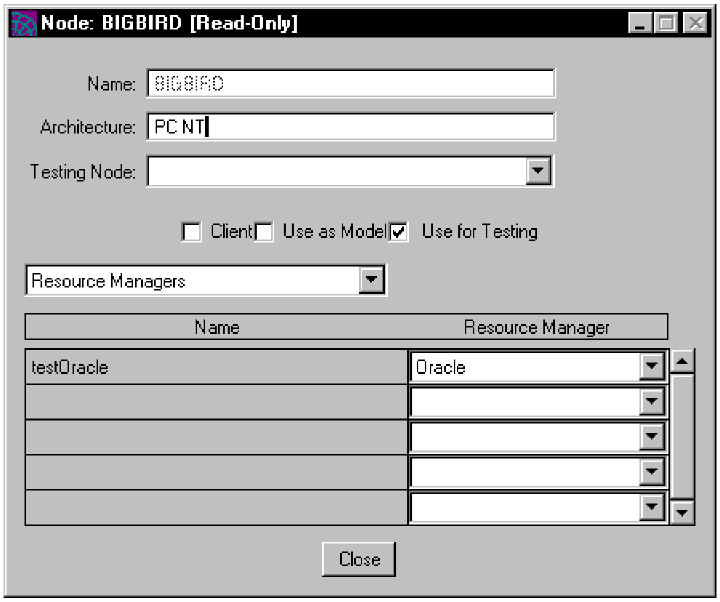
Examining Nodes in an Application Configuration

View menu The Nodes browser displays the assigned partitions within the environment. By default, the browser displays a topological view of the environment. The View > Node Outline command lets you display the same information in an outline form.

For each node in the environment, the browser displays a three-level hierarchy. At the top level is the node name. At the second level are the partitions assigned on that node. At the third level are the service objects assigned to each partition.

Examining node properties To examine the node properties, double-click the node name, or select the node and give the Component > Properties… command. The Node dialog appears, shown in Figure 14-12, displaying the node properties described under "About Nodes" on page 657. These properties are the settings that were specified for the node when the environment was created in the Environment Console, and you cannot change them from the Partition Workshop. See the *iPlanet UDS System Management Guide* for further information on the properties of a node.

Figure 14-12 Node Properties Dialog


Node: BIGBIRD [Read-Only]

Name: BIGBIRD

Architecture: PC NT

Testing Node:

[ ] Client [ ] Use as Model [✓] Use for Testing

Resource Managers

Name	Resource Manager
testOracle	Oracle

Examining the Assigned Partitions

For each assigned partition, you can examine information about the partition as well as information about any DBSession service objects on the partition. To open the Assigned Partition Properties dialog, double-click the assigned partition name or select the partition and choose the Component > Properties… command. See "Assigned Partitions" on page 668 for information about the properties available for assigned client partitions and for assigned server partitions.

Projects assigned to partitions You cannot use the Partition Workshop to see which projects in your application are assigned to a particular partition. However, you can use Fscript to do so. The `ShowApp` command in Fscript shows the current configuration information for the current application in the current environment. Seethe *Fscript Reference Guide* for information on the ShowApp command.

Examining Library Configurations

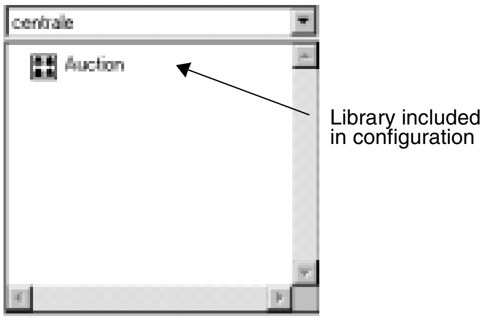
For library configurations, you can examine:

- the projects that are being configured as libraries
- the nodes in the environment
- the assigned libraries on specific nodes in the environment

Examining the Projects

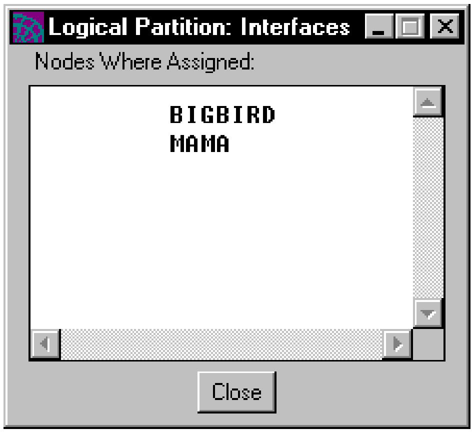
For library configurations, the Project browser portion of the Partition Workshop, shown in Figure 14-13, lists the projects included in the library configuration.

Figure 14-13 Project Browser



Double-click project name to view assignments To view a list of the nodes to which the project is assigned, double-click on the project name. A dialog opens, as shown in Figure 14-14, that lists the nodes where the project is assigned.

Figure 14-14 Logical Partition Dialog for a Project



BIGBIRD
MAMA

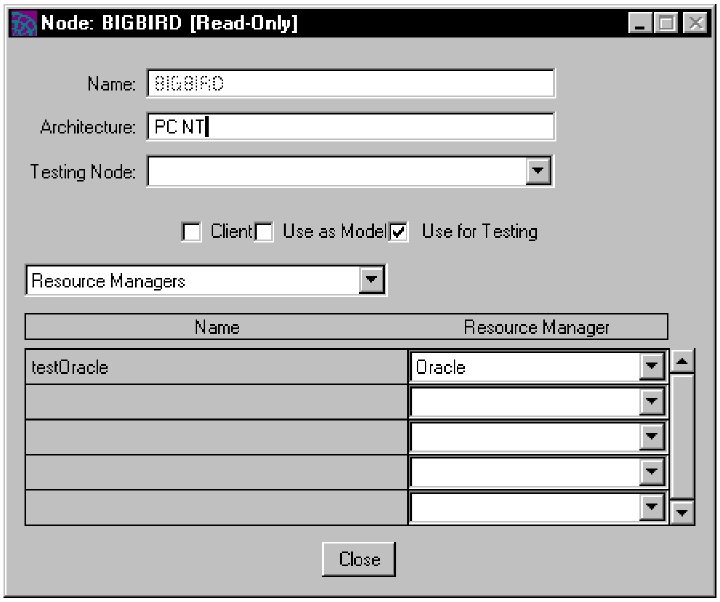
Examining Nodes in a Library Configuration

For library configurations, the Nodes browser displays the assigned libraries within the environment. By default, the browser displays a topological view of the environment. The View > Node Outline command lets you display the same information in an outline form.

For each node in the environment, the browser displays a two-level hierarchy. At the top level is the node name. At the second level are the libraries assigned on that node. If the libraries are not currently displayed, open the node by clicking the expansion arrow.

Examining node properties To examine the node properties, double-click the node name, or select the node and give the Component > Properties… command. The Node dialog appears, shown in Figure 14-15, displaying the node properties described under "About Nodes" on page 657. These properties are the settings that were specified for the node when the environment was created in the Environment Console, and you cannot change them from the Partition Workshop. See the *iPlanet UDS System Management Guide* for further information on the properties of a node.

Figure 14-15 Node Properties Dialog



Node: BIGBIRD [Read-Only]

Name: BIGBIRD

Architecture: PC NT

Testing Node:

☐ Client ☐ Use as Model ☑ Use for Testing

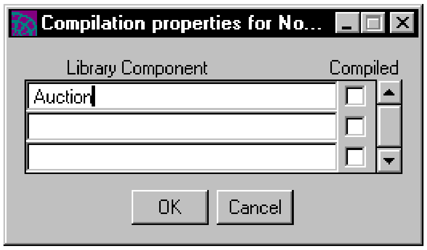
Resource Managers

Name	Resource Manager
testOracle	Oracle

Examining Assigned Libraries

For each assigned library, you can open a dialog that shows which libraries on the node are compiled. To open the Compilation Properties for Node dialog, shown in Figure 14-16, double-click the assigned library name or select the library and choose the Component > Properties… command. A toggle next to the library's name indicates that the library is compiled.

Figure 14-16 Compilation Properties for Node Dialog



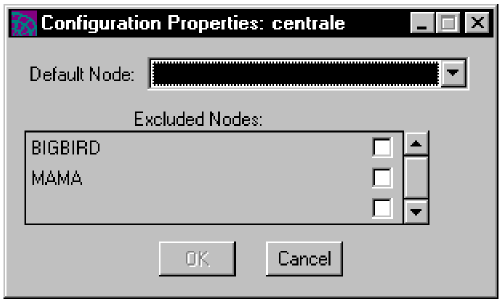
Compilation properties for No…

Library Component	Compiled
Auction	☐
	☐
	☐

Viewing the Configuration Properties

To view the default node and excluded nodes for the current configuration, choose the File > Properties… command. The Properties… command opens the Configuration Properties dialog, shown in Figure 14-17, displaying all the nodes in the environment. If the toggle next to the node name is checked, this indicates that the node is excluded from the configuration.

Figure 14-17 Configuration Properties Dialog



Modifying a Client or Server Configuration

This section describes how to modify a client or server configuration. Because a library configuration is so different, modifying a library configuration is described separately under "Modifying a Library Configuration" on page 708.

Libraries in a separate distribution! Note that the application configuration does not include the libraries that are needed by the application. If your application has supplier libraries, you must create a separate library configuration for the libraries, as described under "Modifying a Library Configuration" on page 708 and then create a separate library distribution for them as described under "Making a Library Distribution" on page 732. Both the application distribution and library distribution must be installed in order for the application to run.

In the Partition Workshop, you can modify your client or server configuration by:

- changing a logical partition, which affects all assigned copies of that partition

  You can modify your logical partitions by moving service objects from one logical partition to another, creating new logical partitions, and changing the service object definitions. Any changes you make to the logical partitions are immediately reflected in the assigned copies of that partition.

- changing the individual assigned partitions
- making reference partitions

  A reference partition points to an existing service object in the installed environment, allowing applications to share an existing service object.
- making a client application into an applet
- modifying a node

  You can modify the nodes in the configuration by assigning partitions to them, moving partitions from one node to another, deleting partitions, and setting the properties of an individual assigned partition.
- setting the properties for the configuration itself, which causes iPlanet UDS to repartition the application

Opening the configuration To modify a configuration, you must first open it (described under "Creating a Configuration" on page 677). If you select an environment that does not yet have a configuration, iPlanet UDS automatically partitions the application. If you select an environment that already has a configuration and you have changed any of your projects in such a way as to affect the configuration, iPlanet UDS automatically performs an incremental partitioning on the application, however, this may take some time.

Permanent changes Note that to make permanent changes to a configuration, your workspace must be open for updating. If your workspace is open for reading only, you can use the Partition Workshop to examine a configuration, test it, and make temporary modifications to it. However, because you cannot save your workspace, any changes you make to the configuration are only temporary.

Modifying Logical Partitions

You can make the following changes to a logical partition:

- move a service object from one partition to another
- create a new logical partition to contain the selected service object
- make a reference partition
- make an applet
- modify the definition of an individual service object

Moving Service Objects

You can move a service object to any compatible partition. For the partition to be compatible, the service objects in the target partition must meet the following conditions:

- If the service object you want to move is replicated, the service objects in the target partition must be replicated the same way.
- If the service object you want to move is associated with a resource manager, the service objects in the target partition must be associated with the same resource manager, if any.
- If the service object you want to move belongs to a restricted project, the service objects in the target partition can belong to another restricted project only if both restricted projects can run on the same node.

If you do try to move a service object to an incompatible partition, you will get an error message that explains why it is incompatible.

Mixing visibility You can include a user-visible service object in the same partition as an environment-visible service object. However, when a user-visible service object is in the same partition with an environment-visible service object, the user-visible service object becomes accessible only within that partition. This limited accessibility can be a useful optimization technique. See "Combining Service Objects and Partitions" on page 705 for more information.

➤ **To move a service object**

1. In the Logical Partition browser, select the service object you wish to move.
2. Drag the service object on top of the target partition name.

   If the partition is incompatible, iPlanet UDS will display an error dialog explaining why the service object cannot be moved.

Creating a New Logical Partition

New Logical Partition command To create a new logical partition, you must have at least one existing service object that you wish to assign to it.

➤ **To create a logical partition**

1. Select the service object you wish to move to the new partition.
2. Choose the Component > New Logical Partition command.

Making a Reference Partition

Before you can make a reference partition, the application that includes the service object must be deployed. Then, you must obtain the project that defines the service object and include it as a supplier plan for your main project.

Project that defines the service object The project that you include as a supplier plan for your main project must be the *same* project that was used to define the service object in the deployed application. The supplier project in the current application must match the project in the deployed application in the following ways:

- the names of the projects must be the same
- the compatibility levels of the projects must be the same
- all runtime IDs must match

CAUTION If the project that defines the service object was created in another repository, you need to export the project from the other repository using the Fscript `ExportPlan` with the `ids` option. Then, you need to import this exported plan into your repository.

Any time that multiple repositories are using the same project, the project must have been imported with IDs into all the repositories (except for an originating repository) even when the project was originally imported into a repository without IDs. Whenever a project without IDs is imported into a repository, new IDs are automatically created for the project, its components, and its service objects. If you simply import the same exported project without IDs into different repositories, applications generated by these separate repositories will not recognize that the projects, service objects, and components are the same because the IDs are different.

If you have received an exported project that does not include unique IDs, you need to get another copy of the exported project that includes these IDs; otherwise, your application will not be able to use the deployed partition. The iPlanet UDS runtime system uses the IDs to determine whether the requested service object and the available service objects are the same.

There are four steps you must follow to include a service object from a deployed application in your current application.

➤ **To include the service object**

1. Deploy the application that defines the service object that is to be shared.
2. If the service object is in a connected environment, you must make a distribution for the server application that contains it in the current environment (although you do not install it).
3. In the application that needs to access the deployed service object, include as a *supplier plan* the project from the deployed application that originally defined the service object.
4. Use the Partition Workshop to create a reference partition that points to the existing service object.

   If the service object is in a connected environment, the environment search path for the service object in the reference partition must specify the environment where the original server application that contains the service object is deployed.

The remainder of this section describes how to use the Partition Workshop to create a reference partition. See the *iPlanet UDS Programming Guide* for detailed information on steps 1 through 3.

When you partition your application in the Partition Workshop, iPlanet UDS creates new partitions containing all the service objects in the main project and all of its supplier projects, just as usual. Therefore, the service object that you are planning to include in the reference partition will be on a new partition (or it may be sharing a new partition with other service objects).

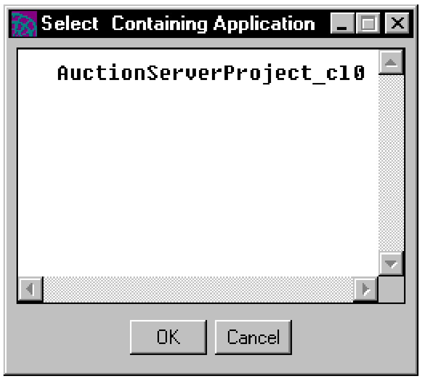
At this point, you must create the reference partition to point to the deployed service object, and move the service object from the new partition into the reference partition. This tells iPlanet UDS to use the service object that the reference partition points to, rather than creating a new instance of the service object.

New Reference Partition command To create the reference partition, use the New Reference Partition command. When you choose the New Reference Partition command, a dialog displays a list of the applications from which you can select an existing service object. This list consists of all those applications for which application distributions have already been made. Therefore, you should make sure that the distribution that includes the service object you want to reference was made before you try to reference it.

➤ **To make a reference partition**

1. In the Logical Partition browser, select the service object name that you wish to access from the reference partition.
2. Choose the Component > New Reference Partition… command.

   The Select Containing Application dialog opens.



3. In the Select Containing Application dialog, select the application that contains the existing service object you wish to reference, and click the OK button.

Automatic startup for reference partitions If automatic starting is specified for the environment (the default), the reference partition will automatically start the service object it references when this is necessary. If the automatic starting property is turned off for the environment (check with your system manager), you can turn on automatic startup for the service object in the reference partition by using the (a) option in its environment search path. However, before using this auto-start feature, please check with your system managers because auto-starting an individual service object affects the startup behavior of the entire application.

For complete information about auto-starting service objects, see the *iPlanet UDS Programming Guide*.

When the application that contains the reference partition is installed, the application that contains the service object that is being referenced must also be installed. If the application that contains the service object is not installed, the application that contains the reference partition will fail at runtime when it tries to access the service object.

If, in the future, the compatibility level of the application that defines the service object referenced by the reference partition is raised, and you want to use the new release of the service object, you must update your application appropriately. For information about using new compatibility levels for service objects, see the *iPlanet UDS Programming Guide*.

Using Reference Partitions with Connected Environments

If your deployment environments are connected and a service object that you need to access can run only in one of the environments, you can access that service object from other environments by using a reference partition combined with an *environment search path*. (To see if a given environment is connected to any other environments, you must use the Environment Console or Escript. For information on how to do so, see the *iPlanet UDS System Management Guide*.)

Environment search path In a reference partition to a connected environment, you use the environment search path on the service object to specify the name of the environment that contains the service object you want to access. When the environment search path for a service object specifies an environment name other than the current environment, every time the service object is referenced, iPlanet UDS always uses the service object in the specified environment.

Search path excludes the local environment When you have applications in multiple environments that need to use a specific service object in a specific environment, you should deploy the application that contains the service object in the environment in which it can run. Then, each application in the other environments can create a reference partition to the service object in that application. The search path from each application would *only* include the specific environment where that service object is deployed. (It must *exclude* the local environment).

Making a reference partition to a connected environment Making a reference partition to a service object in a connected environment is similar to making a reference partition to a service object within the current environment, however, you must follow some extra steps. First, you must make a distribution for the application that contains the service object.

➤ **To make the distribution**

1. Import the project that defines the service object into the development repository.

   Remember, the project that defines the service object must have been exported from the repository where it was defined using the Fscript `ExportPlan` command with the ids option. If it was not, the application with the reference partition will not be able to inter-operate with the application that is running the service object. If your copy of the exported project does not include unique IDs, you need to get another copy of the project that does.

2. In the Project Workshop, choose the File > Configure as > Server command to partition the application that contains the service object.

3. In the Partition Workshop, partition the application appropriately and choose the File > Make Distribution command to make a distribution for the application.

   Making the distribution makes the service object available for referencing in the current environment. However, you should *not* install the application, because the service object is intended to run outside of your environment and cannot be installed in your current environment.

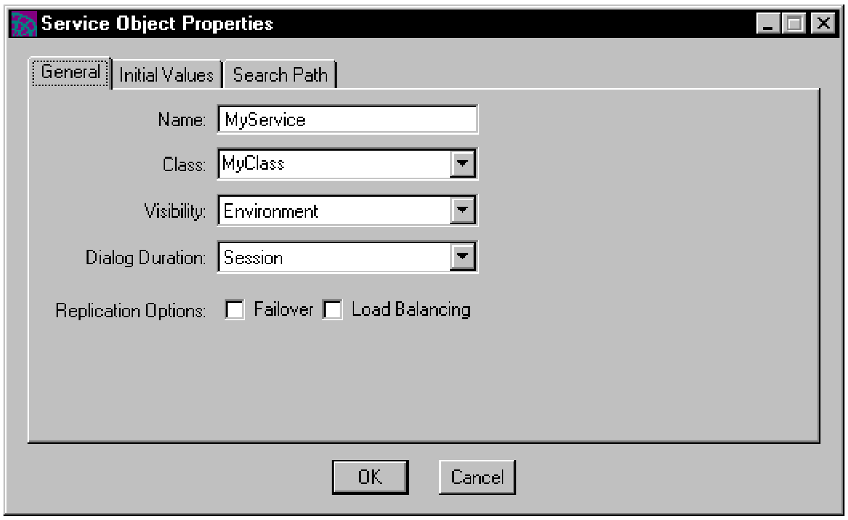
Second, you must use the service object's environment search path to specify which environment actually contains the service object to be accessed.

When a service object is in a reference partition and you provide an environment search path for the service object, iPlanet UDS searches the environments in the environment search path for a service object to use within the current application. iPlanet UDS uses the first service object it finds in the search path.

➤ **To create the reference partition with an environment search path**

1. Use the Component > New Reference Partition command to make a reference partition for the service object and assign it.
2. In the assigned reference partition, double-click the service object name.

   The Service Object Properties dialog opens.



Name: MyService
Class: MyClass
Visibility: Environment
Dialog Duration: Session
Replication Options: ☐ Failover ☐ Load Balancing

3. Click the tab for the Search Path tab page.
4. On the Search Path tab page, enter the environment search path for the service object (see "Specifying the Environment Search Path" on page 698 for the environment search path syntax).

   Because you want to use a service object in a different environment, the environment search path should *not* include the current environment.

Making an Applet

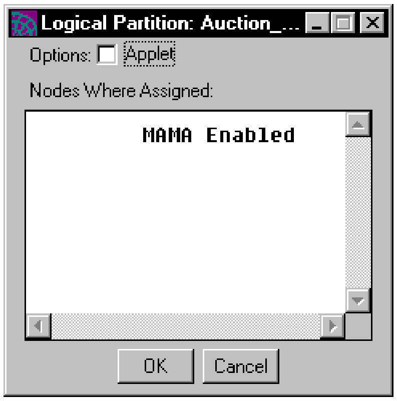
An applet is a client application that can be launched from another application using the AppletSupport library, but that cannot be run as an independent application. See the *iPlanet UDS Programming Guide* for complete information on writing and configuring an application that launches applets.

Applet property You can configure any client application as an applet by turning on the Applet toggle in the Logical Partition Properties dialog for the client partition.

➤ **To create an applet**

1. In the Logical Partition browser, double-click the client partition.

   The Logical Partition dialog opens.



MAMA Enabled

2. In the Logical Partition dialog, turn on the Applet toggle.

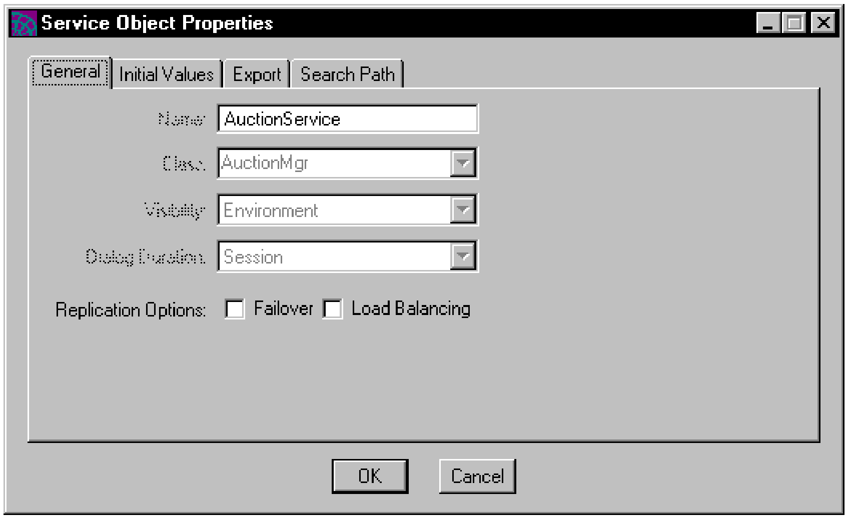
Modifying a Service Object Definition

In the logical partition, you can modify certain properties of the service object. Note, however, that changing the service object definition in the Partition Workshop applies only to the current configuration. Changes do not affect the original service object definition that you specified in the Project Workshop.

In addition, if the service object is in a reference partition, you cannot modify its definition in the current configuration. You can only make changes to the service object definition from the configuration that contains the original service object definition. The exception to this is the environment search path. You must specify a new environment search path for the service object in a reference partition; the environment search path in the original service object definition is ignored in the current configuration.

To modify a service object definition, double-click the service object name to open the Service Object Properties dialog, as shown in Figure 14-18.

Figure 14-18 Service Object Properties Dialog



Name: AuctionService
Class: AuctionMgr
Visibility: Environment
Dialog Duration: Session
Replication Options: ☐ Failover ☐ Load Balancing

In the Service Object Properties dialog, you can update only certain properties. The following table describes the properties on each tab page that you can update for each kind of service object.

Service object	Properties you can update
DB Resource Manager	General Tab page: Failover (see below), Load Balancing (see below) Database Tab page: Database Manager Search Path Tab page: Search Path (see below)
DB Session	General Tab page: Failover (see below), Load Balancing (see below) Database Tab page: Database Manager, Database Name, User Name, Password Search Path Tab page: Search Path (see below).
User	General Tab page: Failover (see below), Load Balancing (see below) Export Tab page: Export Name (see below), External Type (see below) Initial Values Tab page: initial values for attributes Search Path Tab page: Search Path (see below)

Unassigned service objects Use the Service Object properties dialog to modify the definition of an unassigned service object. After you change the external manager name to a name that is available in the current environment, iPlanet UDS will automatically reassign the service object to the appropriate partition. And when all unassigned service objects are assigned appropriately, you can run the application.

Failover and Load Balancing

You can use the Failover and Load Balancing options to turn on replication for a service object. Note, however, that when you turn on failover and load balancing for a service object, iPlanet UDS does not automatically replicate your service objects for load balancing or failover. Instead, you must replicate the partitions to which the service objects are assigned. You can replicate a partition either by assigning it to an additional node (as described under "Assigning Partitions" on page 699) or by setting the startup replicates for an individual assigned partition on a single node (described under "Setting Assigned Partition Properties" on page 700).

Setting the Export Name and External Type

The Export Name and External Type properties are for use with the following external systems:

- IIOP
- OLE

If your service object is not going to function as a server for one of these external systems, you should ignore these properties. See *Integrating with External Systems* for information about integrating OLE with iPlanet UDS applications. See the documentation for the *iPlanet UDS Java Interoperability Guide* for information about integrating with IIOP.

Export name The export name is the name to be used by the client to identify the service object.

External type The external type specifies the type of external client application that can access this service object, either DCE, ObjectBroker, OLE, IIOP, Encina, or none. This external type tells iPlanet UDS to export the files that you need to set this service object as a server for the specified external system.

For example, specifying DCE as the external type tells iPlanet UDS to export an interface definition file when you make a distribution. You will later use the file to set up this service object as a DCE server. The programmer writing DCE client applications that access this service object will also read this file to understand how to access the methods in this service object.

Specifying the Environment Search Path

The environment search path for the service object specifies the connected environments in which iPlanet UDS searches for the service object (see the *iPlanet UDS Programming Guide* for complete information on connected environments).

The environment search path you specify for an individual service object within a configuration overrides the environment search path in the original service object definition itself specified in the Project Workshop (it does not add to it). You can use this feature only if the current environment is connected to other iPlanet UDS environments (see the *iPlanet UDS System Management Guide* for information on this).

Document your settings! If you use the environment search path for a service object, you should be sure to document exactly what you specified for your system manager. Setting the environment search path for an individual service object overrides what the system manager specifies for the environment as a whole, and the system manager needs to be aware of your changes.

Search path syntax To specify the environment search path, enter a string that includes one or more environment paths. iPlanet UDS searches for the service object in the same order as the paths in the string. A blank environment search path specifies using the default search path.

The syntax of the search path string is:

path [**(a)**] [**:** *path* [**(a)**]...

path is:

(@ | @*environment_name*)

(a) option A special (a) option allows you to specify that the service object identified by a specific path should automatically be started if necessary. However, before using this auto-start feature, please check with your system managers because auto-starting an individual service object affects the startup behavior of the entire application. For complete information about auto-starting service objects, see the *iPlanet UDS Programming Guide*.

You can use an environment variable You can use an environment variable to specify the contents of the environment search path. In fact, we recommend using an environment variable because this makes it possible to change the environment search path for the service object after the application is deployed.

The value for the environment variable is set on first access to the service object, using the value of the environment variable as set on the service object's partition. The syntax is:

${*environment_variable_name***}**

Be sure to include the braces!

The following example illustrates a search path that looks first in the current environment, second in the "la" environment, and last in the "sf" environment:

```
@:@la:@sf
```

If environments have the same name If two or more connected environments share the same environment name, and these environments are specified in the environment search path, then you must use the environment UUID to specify each environment. Specify this environment UUID in place of the @environment_name, for example, B763E430-22FF-11D0-A5AA-5BC569EDAA77. For more information about the environment UUID, see the *iPlanet UDS System Management Guide*.

Modifying Node Assignments

You can modify a node assignment in the configuration by:

- assigning partitions to the node
- moving an assigned partition from one node to another
- deleting an assigned partition from the node
- setting properties of an individual assigned partition

Assigning Partitions

Drag logical partition onto a node To assign a logical partition to a node, you simply drag the logical partition onto the appropriate node. The node must provide the resources necessary to run the particular partition.

If the logical partition is replicated, the new assigned partition will be enabled. If the logical partition is non-replicated and there is already a copy of the logical partition assigned in the environment, the new assigned partition will be disabled.

➤ **To assign a logical partition**

1. In the Logical Partitions browser, select the logical partition you wish to assign.
2. Drag the logical partition to the node to which you wish to assign it.

Moving Partitions

You can move an assigned partition to any compatible node. To be compatible, the node must provide the resources required by the service objects on the assigned partition.

Drag assigned partition to new node To move a partition from one node to another, simply drag the partition to its new location. If the node is incompatible, iPlanet UDS will display an error dialog explaining why the service object cannot be moved.

Deleting Partitions

You can delete any disabled, assigned partition from any node. (The last enabled copy cannot be deleted.)

➤ **To delete a disabled, assigned partition**

1. Select the assigned partition.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the partition.

Setting Assigned Partition Properties

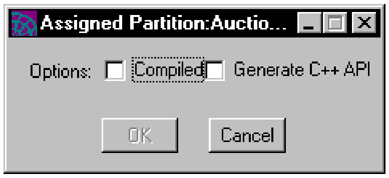
Double-click partition name To set the properties for an assigned partition, double-click the assigned partition name, or select the assigned partition and choose the Component > Properties… command. This command opens the Assigned Partition dialog, where you can set the properties that are appropriate for the particular partition.

The following two sections describe the different properties available for the assigned client and server partitions.

Assigned Client Partition Properties

Figure 14-19 shows the properties available for an assigned client partition:

Figure 14-19 Assigned Partition Properties Dialog (Client Partition)



Assigned Partition:Auctio...

Options: ☐ Compiled ☐ Generate C++ API

Compiled property By default, a client partition is a standard partition. If you wish to make the partition a compiled partition, turn on the Compiled toggle. Remember, after specifying that the partition is compiled, you may need to perform extra steps to produce the application distribution. See "Compiling Partitions" on page 727 for information.

Restrictions for compiled clients Note that even though the Assigned Partition dialog shows the Compiled option for all client platforms, code generation for client partitions is not supported for all platforms. The following table summarizes:

Platform	Client Code Generation Available?	Autocompile Support?
Windows 95/NT	Yes	Yes
UNIX	Yes	Yes
OpenVMS	Yes	Yes

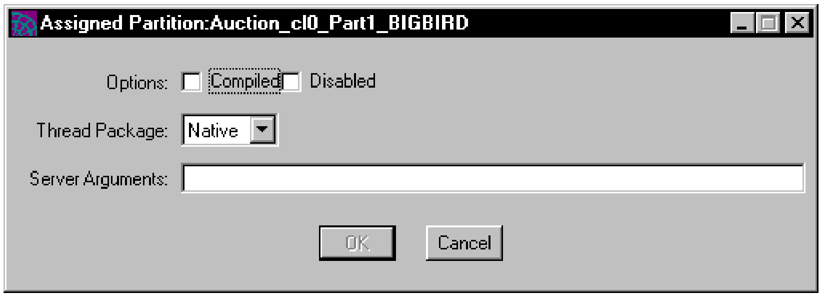
Generate C++ API property For an assigned client partition, you can request that iPlanet UDS generate and compile files for a C++ API. To generate the C++ API for a particular assigned client partition, turn on both the `Compiled` and `Generate C++ API` properties on the Assigned Partition Properties dialog.

When you later use the automatic compilation feature of the Make Distribution command or the `fcompile` command to compile the partitions, iPlanet UDS generates and compiles the files for the C++API. See *Integrating with External Systems* for step-by-step instructions for generating a C++ API for an iPlanet UDS application.

Assigned Server Partition Properties

Figure 14-20 shows the properties available for an assigned server partition:

Figure 14-20 Assigned Partition Properties Dialog (Server Partition)


Assigned Partition:Auction_cl0_Part1_BIGBIRD

Options: ☐ Compiled ☐ Disabled

Thread Package: Native

Server Arguments:

The exact properties available on this dialog depend on whether or not the partition is replicated.

Compiled property By default, a server partition is a standard partition. If you wish to make the partition a compiled partition, click the Compiled toggle to on. Remember, after specifying that the partition is compiled, you may need to perform extra steps to produce the application distribution. See "Compiling Partitions" on page 727 for information.

Disabled property The Disabled toggle indicates whether the assigned partition is currently disabled. To enable a disabled partition, you can turn off this toggle. To disable an enabled replicated partition, turn on this toggle.

Only one copy of a non-replicated partition can be enabled. Therefore, if you enable a non-replicated partition, the other assigned copies of that partition will automatically be disabled. To disable a non-replicated partition, you must explicitly *enable* another copy of that partition.

Thread package property The `Thread Package` property specifies the thread package used by the partition. The default thread package depends on the particular platform (described in the *iPlanet UDS System Installation Guide*). If you do not want to use the default thread package, use the Thread Package property to specify one of the other thread packages supported for the particular platform. See the *iPlanet UDS System Installation Guide* for information about which thread packages are supported for each platform.

Server Arguments property To specify startup flags for the server partition, enter a list of flags in the Server Arguments field. These can be the following:

Flag	Description
-fm *memory _flags*	Specifies the space to use for the memory manager. See Appendix B for information.
-fl *logger_flags*	Specifies the logger flags to use for the session. This has no effect for standard partitions. See Appendix B for information on logger flags.

You can also include your own application-specific flags. See the `CmdLineArgs` attribute of the `Partition` class in the Framework Library online Help for information.

The syntax for the server arguments is platform dependent. For example, any argument with parentheses needs to be enclosed in double quotes or there will be a runtime error.

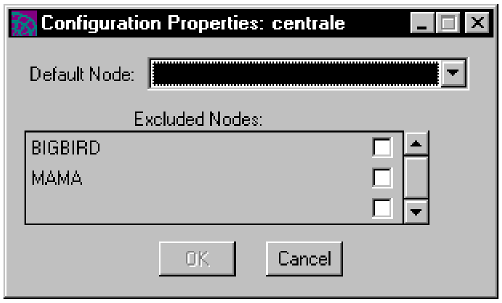
NOTE The server arguments take effect only when the partition is starting using the Partition Workshop, the Environment Console, or Escript; these arguments are ignored when the partition is starting manually using the `ftexec` command.

Replication Count property To provide startup replicates for a replicated partition, enter the total number of partitions to be started in the Replication Count field. When the application starts, iPlanet UDS automatically creates the number of replicates you specify (unless the system manager overrides your setting).

Setting Configuration Properties

To set the configuration properties for the current configuration, select the File > Properties… command. This command opens the Configuration Properties dialog.

Figure 14-21 Configuration Properties Dialog



To specify a default node, select the node name from the drop list. To exclude a node, turn on the toggle next to the node name.

Repartition Command

The File > Repartition… command erases all changes you have made to the configuration and forces iPlanet UDS to perform a complete automatic partitioning on the application. A complete partitioning removes all current assignments, and then recreates the default partitioning scheme from scratch.

Normally, you do not need to use the Repartition… command. Repartitioning is useful mainly when you have made changes to the configuration that you wish to remove.

When you choose the Repartition… command, iPlanet UDS displays a confirmation dialog to ensure that you do not accidentally erase your changes. When you confirm that you want to repartition the application, iPlanet UDS performs the repartitioning and displays the default configuration in your workshop window.

Combining Service Objects and Partitions

When you partition your application for the first time, iPlanet UDS assigns the service objects to logical partitions according to the rules described under "Default Configuration" on page 672. The Partition Workshop then allows you to move service objects between compatible partitions in many different ways.

The way you combine service objects onto partitions has a significant effect on the performance and behavior of your application. Some particularly useful combinations of service object are:

- combining user-visible service objects onto one partition

  This combination minimizes server process resources and network connection resources on clients and servers.

- combining environment-visible service objects onto one partition

  This combination minimizes server process resources and network connection resources on clients and servers.

- combining an environment-visible service object and a user-visible service object

  This combination make the user visible service object private to the partition containing the environment-visible service object.

- combining load-balanced service objects on a single partition.

  This combination allows you to minimize process resources and network connection resources in some cases.

The following sections provide detailed information about these basic combinations.

Combining User-Visible Service Objects On Partitions

Private partition A partition that includes only user-visible service objects is a *private partition* for the client partition (or for any other server partition that is referencing one of the user-visible service objects). iPlanet UDS starts a new copy of the private partition each time a client or another server accesses the service object in the application.

By default, the Partition Workshop places all your user-visible service objects on your client partition, if possible. Otherwise, the user-visible service objects are on a private partition on whichever server node has the necessary resources.

By combining a number of user-visible service objects onto a single private partition, you minimize the number of server processes needed to support each client, as well as the network connection resources needed both on the client and the server. In addition, any access between the service objects within the partition is executed within the same process and is therefore very efficient.

Combining Environment-Visible Service Objects On Partitions

Shared partition A partition that includes only environment-visible service objects is a s*hared partition*. All clients and servers that access the service objects in the partition share the partition. There is only one copy of a shared partition per application (unless it is replicated for load balancing or failover). The shared partition can be started up manually from Environment Console or Escript, or it will be started automatically on first access.

By default, the Partition Workshop places all your environment-visible service objects in the same partition (to minimize the number of partitions) on a server node.

By combining a number of environment-visible service objects onto a single shared partition, you minimize the number of server processes needed to support the clients of the application, as well as the network connection resources needed both on the client and on the server. In addition, any access between the service objects within the partition is executed within the same process and is therefore very efficient.

However, if any of the environment-visible service objects are accessing a resource that supports only single-threaded access, such as a relational DBMS or a C project with the multi-threading option disabled, combining environment-visible service objects into a single partition can cause poor interactive performance in the clients. This is because access from *one* client to the single-threaded resource will delay processing for *all other* clients that are accessing *any* of the service objects in the shared partition. Therefore, to provide the best performance for a shared service object that accesses single-threaded resource, we recommend placing the shared service object in its own partition, possibly a partition that is replicated for load balancing.

Combining Environment-Visible with User-Visible Service Objects

A partition that includes both an environment-visible service object and a user-visible service object makes the environment-visible service object shared for the application and the user-visible service object *private for that partition*. The user-visible service object can be accessed indirectly *through* the environment visible service object—neither the client partition nor any other service objects can access it directly.

The Partition Workshop does not include environment-visible and user-visible service objects within the same partition by default. However, you can move a user-visible service object onto the same partition with an environment-visible service object.

This combination is very useful when you have an environment-visible TOOL class service object that provides DBMS access through a user-visible DBSession or DBResourceMgr service object. We recommend placing the TOOL class service object on the same partition with the DBSession or DBResourceMgr service object because this way a client can access the database only through the TOOL service object. The DBSession or DBResourceMgr service object is protected from inappropriate access, because no other partitions can access it.

Combining Load-Balanced Service Objects

When you combine more than one load-balanced service object on a single partition, iPlanet UDS automatically places all the routers for the load-balanced service objects onto a single router partition.

This architecture is useful for minimizing the number of server processes needed to support an application, because only one process is needed to support both routers for the application. You can use this architecture if you have several service objects that need to access multi-threaded resources, such as:

- Service objects that access true multi-threaded C projects. The C projects must have the multi-threaded option enabled, and must be implemented so they support the multi-threaded conventions on the platform.
- TOOL class service objects that do not access single-threaded resources (such as a relational DBMS).

However, if any of the load-balanced service objects access a resource that supports only single-threaded access, such as a relational DBMS or a C project that has not enabled the multi-threaded option, combining load-balanced service objects onto a single partition can lead to uneven and unpredictable interactive performance in the clients. This is because the two routers perform their routing completely independently from one another. A client request on *one* of the routed service objects can delay a concurrent client request on the *other* routed service object, even though it is from an unrelated client. Because of this behavior, we do not recommend this architecture for service objects that access relational DBMSs or single-threaded external services.

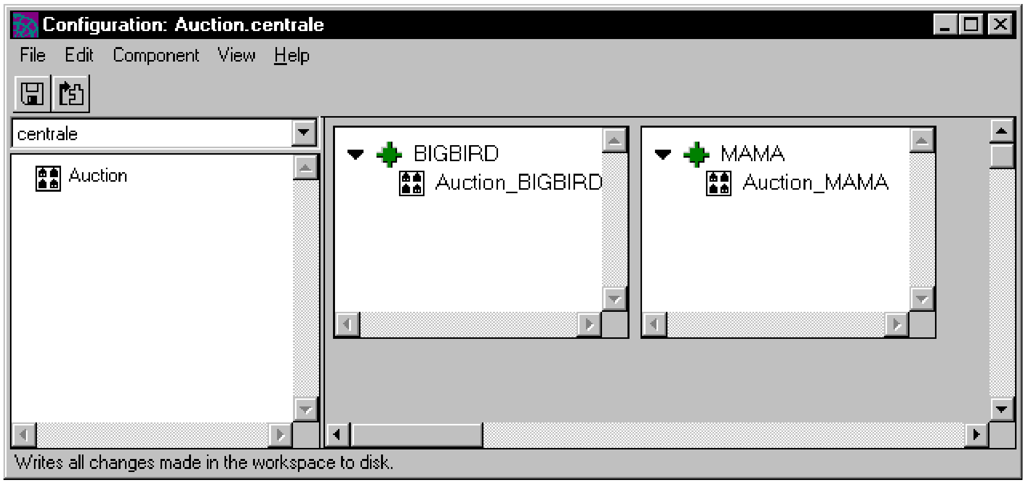
Modifying a Library Configuration

Partitioning a library configuration is very different than partitioning a client or server configuration. When you partition a client or server configuration, you move partitions containing service objects onto nodes in the environment. When you partition a library configuration, you move *projects* onto nodes, not partitions.

A library configuration consists of projects assigned to nodes in the environment. Non-restricted projects can be assigned to any node in the environment. Restricted projects can be assigned to any node on which they will run.

When you configure the project as a library, there is only one project in the shared library, the project that you have “partitioned” by giving the Configure as > Library command. This project you “partition” differs from other projects you later add to the library configuration only in that its name is used to name the library distribution and its compatibility level is used for the library distribution. Otherwise, all projects in the configuration are exactly the same. Figure 14-22 illustrates a default library configuration:

Figure 14-22 Default Library Configuration


Configuration: Auction.centrale
File Edit Component View Help
centrale
Auction
BIGBIRD
Auction_BIGBIRD
MAMA
Auction_MAMA

Writes all changes made in the workspace to disk.

At this point, if you wish to package other projects in the library distribution, you can add them to the library configuration.

➤ To partition a library configuration

1. Specify which projects should be included into the library distribution.
2. Indicate which libraries should be on which nodes in the environment.
3. Specify which libraries should be compiled and which should be standard.

Opening the configuration To modify a configuration, you must first open it (described under "Creating a Configuration" on page 677). If you select an environment that does not yet have a configuration, iPlanet UDS automatically partitions the library configuration.

Permanent changes Note that to make permanent changes to a configuration, your workspace must be open for updating. If your workspace is open for reading only, you can use the Partition Workshop to examine a configuration and make temporary modifications to it. However, because you cannot save your workspace, any changes you make to the configuration are only temporary.

This following sections provide information adding projects to the library configuration, deleting libraries from nodes, and turning on compilation for a library. Setting the configuration properties for a library configuration is the same as setting them for an application configuration. See "Setting Configuration Properties" on page 704.

Adding Projects to the Configuration

You can add any projects in your workspace to the library configuration.

Library names When your library configuration contains more than one library, you must ensure that each library has a unique name. By default, iPlanet UDS uses the first eight characters of the project name as the library name. If two or more projects in the library configuration have names that start with the same first eight characters, you must use the specify unique library names for the projects. See "Setting Project Properties" on page 247 for information on setting the library names.

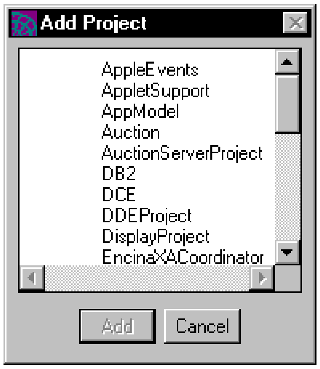
➤ **To add a project to the configuration**

1. Drag the project from the Repository Workshop to the Partition Workshop.
2. In the Partition Workshop, drop the project on any node where you wish to install it.

or

1. In the Partition Workshop, choose the Component > Add Project command.

   The Add Project dialog opens.



2. On the Add Project dialog, select the project you wish to add and click the Add button.

When you add a new project to the library configuration, iPlanet UDS automatically assigns it to all the nodes where the original project is.

The following section describes how to modify this default configuration by removing projects from a node.

Modifying the Default Configuration

You can modify a library configuration by:

- removing individual restricted external library from a node
- removing all libraries from a node

Because restricted external projects cannot run on all nodes in the environment, you must remove them from the nodes where they cannot run.

➤ **To remove a restricted external library**

1. Select the assigned library.
2. Choose the Edit > Delete command.
3. Confirm that you wish to delete the assigned library.

If there is a node where you do not wish the libraries to be installed, you can remove them from the node by deleting any non-restricted project or restricted TOOL project. When you remove one of these projects from a node, all other projects on that node will also be removed.

Standard or Compiled Libraries

By default, compilation for all libraries in the library configuration is turned off.

CAUTION Remember, if a given library has supplier libraries, you must be sure to set the compilation options correctly for the main library and its suppliers. Once you designate a given library on a particular node as compiled, you must ensure that all its supplier libraries are also compiled. For a standard library, the supplier libraries can either be standard or compiled.

You also need to take into account whether or not the partition that will be using the library is compiled. If the partition is compiled, the library that it accesses must also be compiled. If the partition is standard, the library that it accesses can be either standard or compiled.

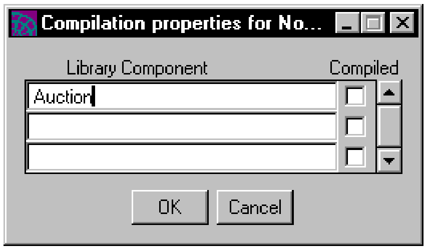
Furthermore, if any of the libraries will be used as suppliers to client partitions with C++ APIs, you need to compile the libraries and generate the handle classes for them. For information about integrating with C++, see *Integrating with External Systems* for complete in

To turn on compilation, you open the assigned library.

➤ **To turn on compilation for a library**

1. Double-click the assigned library.

   The Compilation Properties for Node dialog opens.



2. On the Compilation Properties for Node dialog, click on the Compiled toggle next to the library name.

Remember, after specifying that the library is compiled, you may need to perform extra steps to produce the library distribution. See "Compiling Libraries" on page 735 for information.

Testing a Client Configuration

You test a client configuration in the Partition Workshop by selecting an environment in which to run the client application. If you are in the Repository or Project Workshops, you use the Partition command to enter the Partition Workshop, and test the current client application.

➤ **To test the client application from the Repository Workshop or Project Workshop**

1. Choose the Run > Partition… command or single-click the Partition Workshop tool.

   If the main project for the application was previously configured as a server or library, you will be prompted to change it to a client.

2. In the Partition Workshop, select the environment from the environment drop list.

If you are already in the Partition Workshop, you can simply select the environment whose configuration you wish to test.

Selecting an environment The environment drop list shows all the environments in your environment repository (see *iPlanet UDS System Management Guide* for information about the environment repository and about defining environments). Note that if any environments are currently locked by the Environment Console or the Escript utility, you will not be able to use them.

Automatic partitioning If you select an environment that does not yet have a configuration, iPlanet UDS automatically partitions the application. The automatic partitioning may take some time.

Incremental partitioning If you select an environment that already has a configuration and you have changed any of the projects in the application in such a way as to effect the configuration, iPlanet UDS automatically repartitions the application. This automatic repartitioning is an *incremental* partitioning, that is, iPlanet UDS only changes assignments that have become invalid, and does not make new assignments. For example, if you have added new client nodes to the environment, the incremental partitioning will not assign client partitions to the new nodes. To force a complete repartitioning, you can use the Repartition command (described under "Repartition Command" on page 704).

When the partitioning (or repartitioning) process is complete, iPlanet UDS displays the current configuration.

The Partition Workshop provides two different ways to test the configuration for an application:

Test	Description
Run the application	Runs the application from the startup method and reports compilation errors.
Debug the application	Starts the Debugger for the application, which allows you to monitor the client code as it is being executed.

Testing nodes Running or debugging the application from the Partition Workshop runs the application using the current configuration. To simulate the deployment environment, iPlanet UDS uses the testing nodes that were assigned to each of the deployment nodes. If your environment definition does not assign the appropriate testing nodes, you will not be able to run the application from the Partition Workshop. See the *iPlanet UDS System Management Guide* for information about specifying testing nodes for deployment environments.

When you test your configuration, iPlanet UDS automatically starts up the server partitions on all the appropriate nodes. Therefore, all the appropriate node managers and the environment manager must be running in the environment before you can test. See the *iPlanet UDS System Management Guide* for information about node and environment managers.

iPlanet UDS ensures that your testing does not interact with any installed applications or with any applications being tested by other developers. For your test run, iPlanet UDS provides separate copies of all service objects, even those in reference partitions, so that your test does not interfere with concurrent use or testing of the service object.

When applets are being launched When you test an application that launches applets and client applications using the AppletSupport library, the applets or client applications being launched must be available in the development environment (installed or publicly available). If they are not installed or publicly available, you will get an error. However, the main application itself does not have to be installed. See the *iPlanet UDS Programming Guide* for complete information on writing and configuring applications that launch applets or other applications.

Testing server applications Note that because a project that defines a server application does not have a startup class and method, you cannot run or debug the configuration for an application whose main project defines a server application. However, if you run a client application that includes the project that defines the server, you can test the server as part of that application.

TestClient utility If you wish to test shared service objects in the application by running multiple clients, you can use the iPlanet UDS TestClient utility, which allows any number of clients to participate in a test run from the Partition Workshop. See the *iPlanet UDS Programming Guide* for information.

Running the Application

To run the application, choose the Run > Run command, or single-click the Run tool on the toolbar. This works exactly like the Run Distributed command in the Repository and Project Workshops.

iPlanet UDS starts executing the application by invoking the start method on an object of the start class, and it runs until the application exits. All necessary server partitions are started on the appropriate nodes. Any errors are reported in the Error window (see "Using the Error Window" on page 269).

The first time you run the application using the Run command, iPlanet UDS starts up the remote server partitions for the application and leaves them running, even when you exit from the application. This is for efficiency reasons, so that the next time you run the application, it is not necessary to restart the remote server partitions.

iPlanet UDS stops the remote partitions in the following cases:

- when you exit from the Partition Workshop
- when you run another application
- if you make changes to the a project that make the remote partition out of date
- if you change the configuration (in this case, it only stops the partitions that are effected)

If necessary, you can stop the remote partitions explicitly at any time by using the Run > Stop Remote Partitions command. This command allows you to recheck the start logic of the remote partitions the next time you test the application.

If you are unable to exit the application, you can use the Cancel Run command at any time to cancel execution. This cancels the client partition for the application. The remote partitions will still continue to run; use the Stop Remote Partitions command to stop these.

Debugging the Configuration

Debugging the application from the Partition Workshop allows you to run the application using the current configuration and to step through the code running on the client partition. You cannot use the iPlanet UDS Debugger to monitor code running on remote partitions.

To debug the application, choose the Run > Debug command, or click the Debugger tool. The Debugger uses the startup class and method as the starting point for running the application. iPlanet UDS begins executing the application by constructing a new object of the startup class. It displays the code for the startup method in the Task Window, and suspends execution immediately before the first statement in the startup method. See Chapter 13, “Using the Debugger,” for information about using the Debugger.

Making an Application Distribution

This section describes how to create an iPlanet UDS application distribution for installation into an active iPlanet UDS environment. You create an application distribution from a configuration using the Make Distribution command.

Libraries for application are in a separate distribution! The application configuration does not include the libraries that are needed by the application. If your application uses one or more libraries, you must be sure that they are installed in all intended deployment environments. First, you must create a separate library configuration for the libraries, as described under "Modifying a Library Configuration" on page 708 and then create a separate library distribution for them as described under "Making a Library Distribution" on page 732. Both the application distribution and library distribution must be installed in order for the application to run. See the *iPlanet UDS System Management Guide* for information about installing applications and libraries.

Understanding Application Distributions

As the last step in the partitioning process, you create an *application distribution*. An application distribution is a collection of files outside of the development repository that represent an application intended for deployment. Once you create an application distribution, you load the distribution into a target environment, and install it using iPlanet UDS system management tools. Additional steps are required if your configuration contains compiled partitions and you do not or cannot use automatic compilation when you give the Make Distribution command.

Standard Partitions

A standard partition, with its companion runtime repository, runs against the `ftexec` executor (or one of its variants), which is part of every iPlanet UDS development or runtime system. Any partition, whether client or server, may be a standard partition. See the *iPlanet UDS System Management Guide* for information about `ftexec` and its variants.

If your application uses only standard application partitions, your application distribution is complete when you create it (that is, after you give the Make Distribution command). You can install the application immediately into an active iPlanet UDS environment.

Compiled Partitions

A compiled partition runs as its own independent executable in the context of an iPlanet UDS application. You designate a partition as compiled by opening the assigned partition's properties dialog (see "Setting Assigned Partition Properties" on page 700).

Restrictions for compiled client partitions Compiled client partitions are not supported on all the client platforms. The following table summarizes which platforms support client code generation and indicates whether automatic compilation (described below) is available:

Platform	Client Code Generation Available?	Autocompile Support?
Windows 95/NT	Yes	Yes
UNIX	Yes	Yes
OpenVMS	Yes	Yes

Automatic compilation If your configuration contains compiled partitions, you may wish to select the automatic compilation option when you give the Make Distribution command. Automatic compilation will be available if your system manager has set up your iPlanet UDS installation with the appropriate applications. See the *iPlanet UDS System Management Guide* for information on this.

If automatic compilation is not available to you, or if for some reason you choose not to use it, after giving the Make Distribution command, you must use the iPlanet UDS `fcompile` command to generate C++ code for the partitions and compile the code. This process is described under "Compiling Partitions" on page 727.

Launching Applets and Other Applications

If your client application launches one or more applets, you must create a separate application distribution for each applet that the main application launches. Likewise, if your client application launches other client applications, you must create a separate distribution for each application that is launched (if the distributions have not already been created).

To ensure that all the applications and applets required by a main client application are installed when the main application is deployed:

- include all the application distributions with the main client applications
- write an Escript script that installs all the applications and applets when the main client application is installed

See the *iPlanet UDS Programming Guide* and the *iPlanet UDS System Management Guide* for complete information on writing and configuring applications that use the Launch Server and applets.

Adding an Icon File for Windows to the Distribution

You can include in your application distribution icon files that iPlanet UDS will use when creating the client icon on Windows 3.1, 95, and NT. After using the Make Distribution command to generate the application distribution, add the appropriate .ico files to the port-specific directories in the application distribution directory structure shown in the following section.

Application Distribution Directory

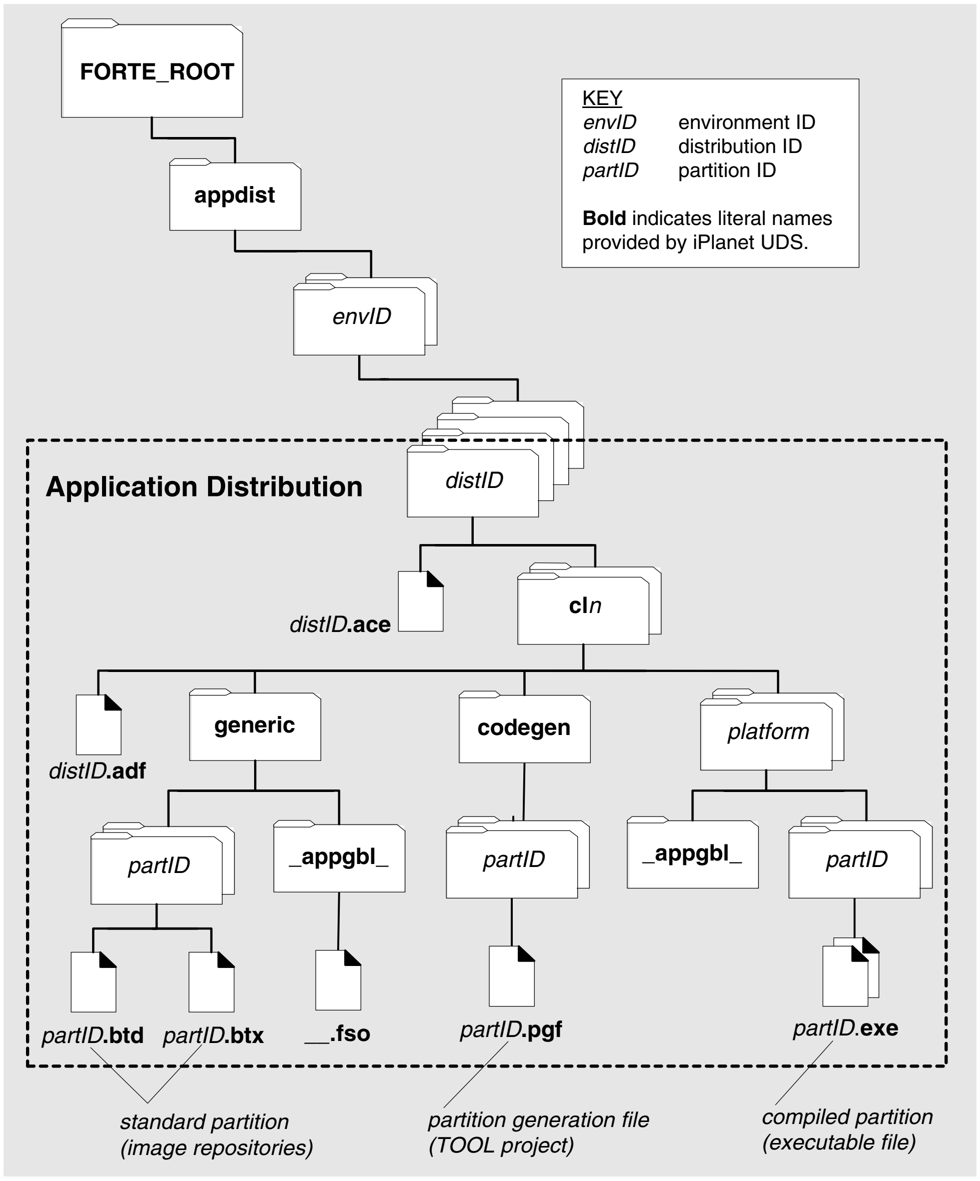
An application distribution consists several elements, including the actual partitions you assigned, and a group of files describing the application.

Distribution directory The distribution is contained in the following directory:

`FORTE_ROOT/appdist/`*environment_ID*`/`*distribution_ID*`/cl`*n*

The *environment _ID* is the ID for the deployment environment, the *distribution_ID* is the first eight characters of the name of the project that was configured. The *n* in the cl*n* is the compatibility level of the project that was configured.

Figure 14-23 Application Distribution Directory Structure



The following table describes these files.

Name	Purpose
—.ace	File which maps iPlanet UDS application component names to unique identifier names
—.adf	Application distribution file. Contains information about the application (partitioning) configuration—what partitions are assigned to what nodes—for use in the deployment process. (The .adf extension is added to the application's unique identifier name.)
generic	Directory that contains all portable files. The partition directories below the generic directory contain all the standard partitions.
codegen	Directory that contains all source files used for generating C++ compiled partitions.
platform1…	Directories that contain non-portable compiled partitions for each supported platform.
appgbl	Directory in which you or iPlanet UDS can place files that would be installed along with any partition (generic directory) or set of partitions assigned to a node (platform directories). For example, this file is used for message files that are used for internationalization.
—.btd, —.btx	Standard partition image repositories. (The .btd and .btx extensions are added to the partition's unique identifier name.)
—.pgf	Partition generation file. Contains the source code used to make compiled partitions for a single logical partition. (The .pgf extension is added to the partition's unique identifier name.)
—.exe, …	Compiled partition executable file.(The name is platform dependent and based on partition's unique identifier name.)

File Naming Conventions

In iPlanet UDS distributions, all names are based on the name of the main project for the application. The following table describes how the names of various distribution components are determined:

Name	Determined by
Application name	The name of the project that was configured.
Full project and application names	The name of the project that was configured with the compatibility level appended to the end. For example, if a main project named "Auction" has a compatibility level of 1, its full name is "Auction_cl1". The full application name is the same.
Distribution ID	The first 8 characters of the name of the project that was configured, not the entire project name (because name length is limited on some platforms).
	Standard partitions, runtime repositories, compiled partitions, distribution information files, application distribution files, and even some of the directory structure that defines the application distribution all use the distribution ID as a basis for their names.
Partition names	The full application name plus an iPlanet UDS system-generated number. For example, the name of a partition for the "Auction_cl1" application is "Auction_cl1_part# (where # is the system-generated number).
Partition unique identifier	The first 6 characters of the application name plus the partition number. The partition generation file (.pgf) is derived from the partition's unique identifier.

For information about project names, see "Using iPlanet UDS Names" on page 131. For information about environment names, see the *iPlanet UDS System Management Guide*.

Using the Make Distribution Command

The Make Distribution command produces an application distribution from the current configuration.

The following table briefly describes the options available for the Make Distribution command:

Make Distribution Option	Description
Local/Remote	Specifies whether the distribution should be made on the local node or a remote node. If you choose remote, you can select the specific node where you wish to create the distribution. See "Local/Remote Option" on page 723 for further information.
Auto-Compile	Specifies whether partitions marked as compiled should be automatically compiled while making the distribution. If auto-compile is off, you must compile the partitions yourself as described under "Compiling Partitions" on page 727. Automatic compilation will work only if your system manager has set up your iPlanet UDS installation correctly, and is not available on all platforms for client partitions. See"Auto-Compile Option" on page 723 for special information about using automatic compilation for Windows 95 compiled clients.
Full or Partial Make	If a distribution already exists for the configuration, a partial make creates the distribution only for those components that have changed since the last make. See "Full or Partial Make" on page 725 for further information.
Install in Current Environment	If the configuration is for the current environment, you can automatically install. However, you should avoid installing too many versions of the same application. See"Install in Current Environment Option" on page 726 for further information.
Include Source	When you are making a library distribution, this option specifies that the library source code will be included in the export files.

When the Make Distribution completes, a message in the status line indicates that the distribution was complete. You will be notified if automatic compilation or installation fails.

The following sections provide more detailed information about some of the Make Distribution command's options.

Local/Remote Option

When you make an application distribution from the Partition Workshop, the first option on the Make Distribution dialog is where to store the application distribution: on the local machine or on a remote server.

Do not use a model node If you are running the Partition Workshop on a model node, do not select Local as the node where the application distribution is placed. The Environment Manager never searches for application distributions on model nodes, therefore, you should not make an application distribution on a model node. Instead, select the remote node where you chose to store all application distributions. Otherwise, the Environment Manager cannot locate the application distribution.

Auto-Compile Option

In general, if your system manager has set up your environment to support automatic compilation, you can use the Auto-Compile option to compile your partitions automatically. However, for Windows 95 compiled client partitions, there are special considerations.

Windows 95 and auto-compile If you want to use the auto-compile feature to compile partitions that run on the Windows 95 platform, you have two choices:

- Compile partitions as Windows NT partitions, then include the compiled executables or libraries as part of the Windows 95 distribution.

  This approach is recommended, because most Windows NT machines are usually more powerful than Windows 95 machines, which makes compilations more efficient. See "Compiling Partitions as Windows NT partitions" on page 724 for detailed instructions.

- Set up a Windows 95 node to act as a server that can auto-compile Windows 95 partitions.

  This option is recommended if your environment does not include any Windows NT nodes. See the *iPlanet UDS System Management Guide* for complete information on how to set up a Windows 95 node to support automatic compilation.

Compiling Partitions as Windows NT partitions

You can compile partitions as Windows NT partitions, then include the compiled executables or libraries as part of a Windows 95 distribution because Windows 95 code and Windows NT code are binary compatible. You might want to use this approach if you have Windows NT nodes already available with auto-compile services installed. Compiled Windows NT executables and libraries also run on Windows 95.

➤ **To compile partitions in Windows NT and use them in Windows 95 distributions**

1. In the Partition Workshop, assign to Windows NT nodes the partitions that you want to compile and run on Windows 95.
2. Make a distribution for the application using auto-compile, but not auto-install.
3. Copy the compiled executable or library for the partition from pc_nt to the pc_win95 directory for that partition in the `FORTE_ROOT/appdist` directory.

   For example, if you have assigned the client partition for the Banking_cl0 application to a Windows NT node to auto-compile it, you need to copy the following file:

   FORTE_ROOT/appdist/*environment_ID*/banking/cl0/pc_nt/bankin0/bank in0.exe

   Copy this file to the following directory (make the pc_win95 and client partition directories first, if they does not already exist):

   FORTE_ROOT/appdist/*environment_ID*/banking/cl0/pc_win95 /banking0

➤ **To install the Windows 95 distributions**

1. In the Environment Console or Escript, load your application, lock the active environment, then assign the client partitions to Windows 95 nodes and set the partitions as compiled. Unlock the active environment when you are done.

   You might get a warning on this step that says that the distribution does not contain the compiled partition for Windows 95. You can ignore this warning and continue.
2. Install the application using the **Install** command for the Application agent.

For more information about the Environment Console, see the *iPlanet UDS System Management Guide*. For information about Escript and iPlanet UDS system agent commands, see *Escript and System Agent Reference Guide*.

Full or Partial Make

The first time you make a distribution for a given configuration in a given workspace, iPlanet UDS creates a full distribution, whether or not you select the Full or Partial Make. The next time you make a distribution for the same configuration in the same workspace and you have not yet given and Update Workspace command, you can request a partial make. For a partial make, the Make Distribution command changes the existing distribution by updating only those components that have changed. Naturally, you should only choose the Partial Make option after you have already created a full distribution.

In order to use a partial make, you must not only make the distribution from the same workspace that you used to make the original distribution but you must also use the same *node*. When you request a partial make, iPlanet UDS checks the node to make sure that the appropriate full distribution is already present, and if it is not, you will get an error.

Using Partial Make or Full Make with Auto-Compile If you have already made a distribution that contains a compiled partition, and you want to make a distribution again for the same application, you should consider the following situations when you decide whether to make a partial distribution or a full distribution.

- You have changed a server partition from a standard, or interpreted, partition to a compiled partition.

  You can use the Partial Make option with the Auto-Compile option, and the new distribution will contain the newly-compiled server partition.

- You tried to make a distribution with the Auto-Compile option, and the compile failed.

  After you have corrected the cause of the compilation error, you need to select the Full Make option with the Auto-Compile option to have the compiled server partitions of the distribution auto-compiled.

- You made a distribution that contained a compiled server partition without using the Auto-Compile option, and you now want to make a distribution using the Auto-Compile option.

  You need to select the Full Make option with the Auto-Compile option to have the compiled server partitions of the distribution auto-compiled.

Install in Current Environment Option

The Install in Current Environment Option lets you automatically install the application in the development environment after the distribution is created.

Installing on Windows 95 If you are making this application distribution on Windows 95 and you did not start the iPlanet UDS Workshops using the Launch Server, the application is not automatically installed on the node running the Partition Workshop, because the client node is not running a node manager. The Environment Manager can only install applications on nodes that have a node manager running. (See "Starting the iPlanet UDS Workshops" on page 110 for information about starting the iPlanet UDS Workshops with and without the iPlanet UDS Launch Server.)

To have the application installed on PC, start the Launch Server provided as described under "Running the iPlanet UDS Launch Server" on page 121. The Launch Server starts a node manager and prompts the Environment Manager to install any application partitions that are assigned to the PC.

You can also start the Environment Console or Escript, which starts a node manager on the client node. You can then install the application on the client node using the **Install** command for the Environment Console or Escript.

For more information about the Launch Server, the Environment Console, Escript, and how applications are deployed in the environment, see the *iPlanet UDS System Management Guide*.

Using Auto-Install with Installed Applications By default, iPlanet UDS assumes that, after you have installed an application, the configuration of the installed application is the correct configuration for the application.

This assumption can affect whether your changes to the application configuration in the Partition Workshop are used when installing the application or not.

If you change the contents of any logical partition, iPlanet UDS cannot re-install the application because the new configuration is incompatible with the configuration of the installed application.

Un-installing the installed application If you have changed only the partition assignments of an application that is installed in the current environment, the installed configuration will remain the same as for the original installation of the application, no matter how you assign the logical partitions in the environment after you have installed the application. Therefore, to see the changes, you must first shutdown and un-install the installed application.

➤ **To change the configuration of an installed application without changing the contents of any logical partitions**

1. *Shutdown* and *un-install* the application using the Environment Console or Escript.

   You might need to ask your system administrator to perform this task, depending on who is permitted to use the Environment Console or Escript in your environment. For information about how to un-install an application, see the *iPlanet UDS System Management Guide*.

2. In the Partition Workshop, configure the application the way you want, then choose the File > Make Distribution command.

   In the Make Distribution dialog, choose the Full Make and Install in Current Environment options. You can also choose the Auto-Compile option, if appropriate. Click the Make button.

Compiling Partitions

If your configuration contains one or more compiled partitions and you did not use automatic compilation with the Make Distribution command, the process for making a distribution has some extra steps.

➤ **To make a distribution for a configuration containing compiled partitions**

1. Give the appropriate Make Distribution command.

   For each compiled partition, iPlanet UDS creates a partition generation file (.pgf file), which you use with the `fcompile` command to generate a completed, compiled partition.

2. Set your environment variables and path as described under "Environment Variables and Path".

3. Run the `fcompile` command, as described under "Using the fcompile Command for Partitions" on page 729.

Environment Variables and Path

Depending on certain characteristics of your system, you may need to set a few environment variables before you make the distribution for a compiled partition. You also have to make sure you have a valid C++ compiler installed and in your path (see *iPlanet UDS System Installation Guide* for information about which version of the C++ compiler you need).

The following table describes the environment variables that affect the `fcompile` command:

Environment Variable	Description
ORACLE_HOME	Set the ORACLE_HOME environment variable if you are running Oracle in the partition. Set the variable to the root directory of the Oracle installation. If you do not set this environment variable, you will receive an error message.
SYBASE	Set the SYBASE environment variable if you are running Sybase in the partition. Set the variable to the root directory of the Sybase installation. If you do not set this variable when necessary, you will receive an error message.
FORTE_X_LIBDIRS	Set this variable only if the Display library is contained in the partition and if you are unsure if your X Window header files reside in a standard location. Generally, you should set this variable if you are running on a Sparc; if your hardware vendor also supplied your windowing software, then you probably do not need to set the variable. The syntax is: FORTE_X_LIBDIRS -L/*dir_name*/lib
FORTE_X_HEADERDIRS	Set this variable if the Display library is contained in the partition. This variable points the C++ compiler to the directory in which the header files for X Windows and Motif are stored. The syntax is: FORTE_X_HEADERDIRS -I/*dir_name*/include To find out if the Display library is in the server partition, use the `FindProj`, `FindApp`, and `ShowApp` commands in Fscript.
FORTE_CG_RESERVED	Set this variable to provide a file containing reserved words to supplement the list of iPlanet UDS reserved words. You should provide this file when your project uses names for class components that are already reserved by your C++ compiler.The iPlanet UDS code generator uses the reserved words file to rename the class components in order to avoid conflicts. The reserved word file is strictly optional—you do not need to provide one. If you do not set the variable, the default is: $FORTE_ROOT/install/scripts/cgreserv.lst. The syntax is: `FORTE_CG_RESERVED` *file_specification*

Using the fcompile Command for Partitions

The `fcompile` command generates code, compiles, and links a compiled partition from a .pgf file. The process for compiling a client partition and a server partition is the same.

The portable syntax is:

```
fcompile [-c component_generation_file] [-d target_directory] [-o output_file]
         [-cflags compiler_flags] [-lflags linking_flags]
         [-fns name_server_address] [-fm memory_flags] [-fst integer]
         [-fl logger_flags] [-cleanup]
```

The OpenVMS syntax is:

```
VFORTE FCOMPILE
         [/COMPONENT=component_generation_file]
         [/DIRECTORY=target_directory]
         [/OUTPUT=output_file]
         [/COMPILER=compiler_flags]
         [/LINKING=linking_flags]
         [/NAMESERVER=name_server_address]
         [/REPOSITORY=repository_name]
         [/MEMORY=memory_flags]
         [/STACK=integer]
         [/LOGGER=logger_flags]
         [/CLEANUP]
```

The `fcompile` command has other flags that are for use only when integrating with external systems. See *Integrating with External Systems* for information on these flags.

The following table describes the command line flags for the `fcompile` command:

Flag	Description
`-c` *component_generation_file* `/COMPONENT =` *component_generation_file*	Specifies the file that iPlanet UDS compiles. This value includes the path where the file resides if the file is not in the current directory. By default, iPlanet UDS compiles all files in the current directory.

Flag	Description
`-d` *target_directory* `/DIRECTORY =` *target_directory*	Specifies where the compiled directories will be placed. By default, `fcompile` compiles files in the current directory, and places the compiled files in the current directory. *target_directory* is a directory specification in local syntax. If the `-c` (`/COMPONENT`) flag is also specified, the `-d` flag specifies where the compiled component files will be placed. Otherwise, the directory specified by the `-d` (`/DIRECTORY`) flag specifies both the directory containing the files to be compiled and the directory where the compiled files will be placed.
`-o` *output_file* `/OUTPUT =` *output_file*	The log output file used by the `fcompile` command.
`-cflags` *compiler_flags* `/COMPILER =` *compiler_flags*	Specifies any C++ compiler options. Any header file specifications included here are used before the specifications included in the C project definition. For more information about these options, see *Integrating with External Systems*.
`-lflags` *linking_flags* */LINKING = linking_flags*	Specifies any linking flags. Any files included here are linked before files specified in the extended properties of the C project definition. For more information about specifying linking flags in the C project, see *Integrating with External Systems*.
`-fm` *memory_flags* `/MEMORY =` *memory_flags*	Specifies the space to use for the memory manager. See "-Fm Flag (Memory Manager)" on page 799 for information.
`-fst` *integer* `/STACK=`*integer*	Specifies the thread stack size in bytes for iPlanet UDS and POSIX threads. This specification overrides default stack size allocation. Typically, you use the default specification. For compiling large applications, you may want to specify a larger thread stack size. For more information on this flag, refer to the *iPlanet UDS System Management Guide*.
`-fl` *logger_flags* `/LOGGER =` *logger_flags*	Specifies the logger flags to use for the command. See "-Fl Flag (iPlanet UDS Logger)" on page 795 for information.
`-cleanup` `/CLEANUP`	Deletes all the files except for the newly compiled partitions.

Where to run fcompile When you give the `fcompile` command for a given partition, you must do so on machine with the same architecture as the node to which the partition is assigned. For example, if you are building a compiled partition for an RS6000, you must run `fcompile` on an RS6000.

Do not use a model node You cannot use a model node to create a distribution. Therefore, you should not run the `fcompile` command on a machine that is defined as a model node within the current environment.

➤ **To run fcompile for a compiled partition**

1. If necessary, move the .pgf file to the machine with the same architecture as the intended partition.

2. Set the necessary environment variables, as described in "Environment Variables and Path" on page 727.

3. Give the `fcompile` command as described above.

   ```
   fcompile -c aucio1.pgf -d ./rs6000
   ```

   This creates an executable partition file in the format used by the specific platform.

4. Return the executable partition file to the component's subdirectory of in the appropriate port in the distribution directory.

File format for OpenVMS For OpenVMS only, you must ensure that the .pgf file's record format is stream_lf (you can check the format with the dir/full command). If the record format is not stream_lf, or if you get an error reading the file, use the following command to change the file format:

```
set file/attrib=rfm=stmlf 'filename'
```

Your application distribution is now ready to install. For instructions on installing an application distribution, see the *iPlanet UDS System Management Guide*.

Compiling a Partition for Use on Several Computing Platforms

You may need to compile a server for use on more than one computing platform. For example, let's say you are compiling a server partition for an application that is load balancing across server platforms of varying architectures: one server is an IBM RS/6000, one is a Sun SPARCStation running UNIX, and a third is a VAXStation running VMS. In this case, you would compile three separate executable partition files using the same .pgf file.

Making a Library Distribution

This section describes how to create an iPlanet UDS library distribution for installation in iPlanet UDS deployment and/or development environments.

The process of making a library distribution is similar to making an application distribution. The major difference is that you can specify whether or not to include source code in the library distribution.

By default, source code is not included in the library distribution. When the library distribution is created without source code, iPlanet UDS developers using the library cannot view the source code for the methods, cursors, and event handlers in the library. A special option on the Make Distribution command allows you to request that source code be included in the distribution. When the library distribution is created with source code, iPlanet UDS developers using the library can view (but not modify) all the source code. (If you need to distribute source code that *can* be modified, you should simply ship the project export files.)

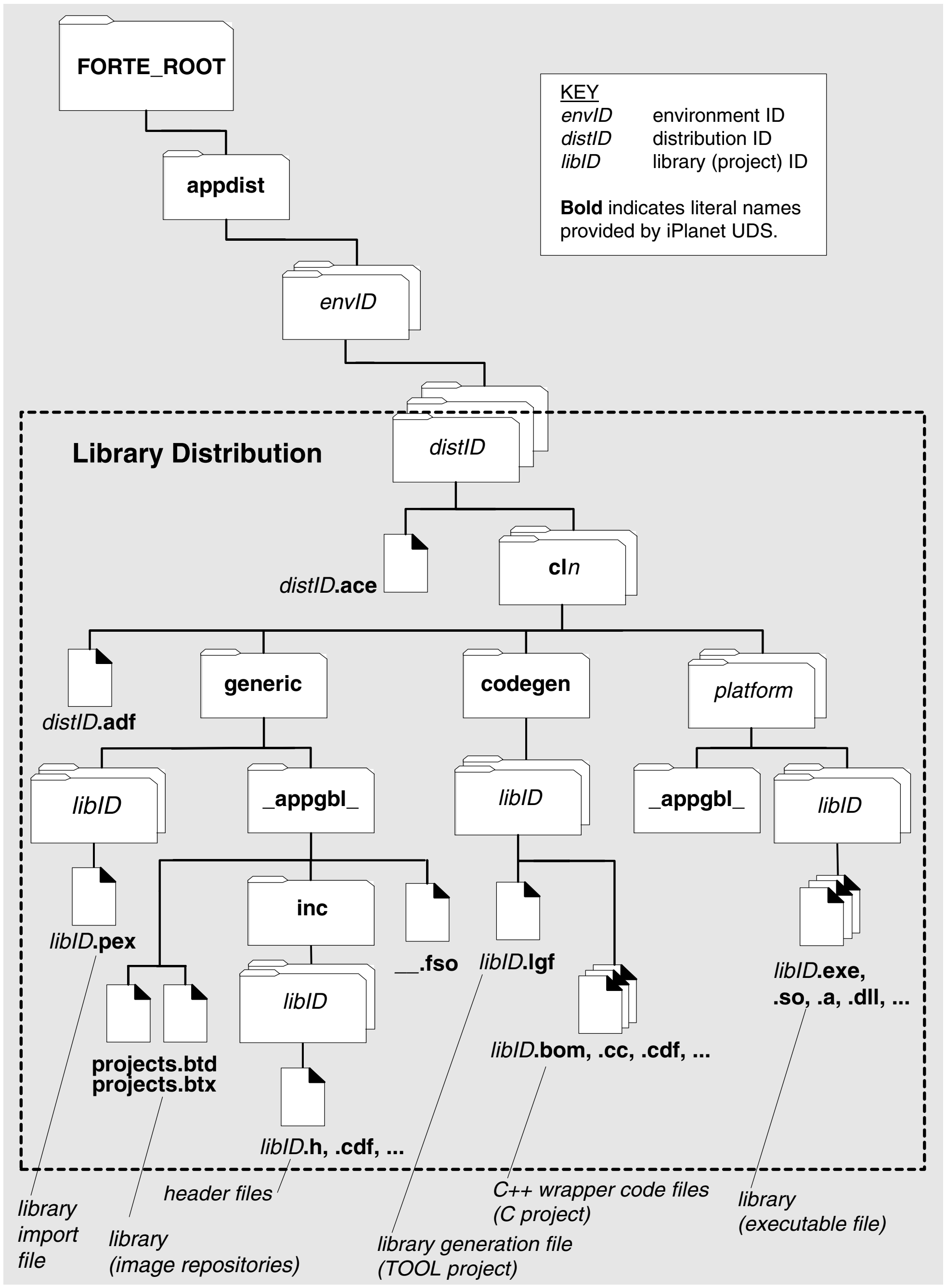
You create a library distribution from your library configuration, using the Make Distribution command.

About Library Distributions

A library distribution is a collection of files outside of the development repository that represent a library configuration (that is, a set of related shared libraries) intended for deployment. Once you create a library distribution, you can load the distribution into a target environment, and install it using iPlanet UDS system management tools.

"Understanding Application Distributions" on page 716 describes the directory structure iPlanet UDS uses for an application distribution. The following figure shows the directories and files that iPlanet UDS adds to the distribution directory structure when you create a library distribution.

Figure 14-24 Library Distribution Directory Structure



The following table describes these files.

Name	Purpose
—.ace	File which maps iPlanet UDS library configuration component names to unique identifier names.
—.adf	Application distribution file. Contains information about the library (partitioning) configuration—what libraries are assigned to what nodes—for use in the deployment process. (The .adf extension is added to the library configuration's unique identifier name.)
generic	Directory that contains all portable files. The partition directories below the generic directory contain all the standard partitions.
codegen	Directory that contains all source files used for generating C++ compiled partitions.
platform1…	Directories that contain non-portable compiled partitions for each supported platform.
appgbl	Directory in which you or iPlanet UDS can place files that would be installed along with any library or set of libraries assigned to a node.
projects.btd, projects.btx	Standard image repository to be used on platforms for which no compiled libraries have been made.
—.fso	File that maps method names to ids, which might be used for some external interfaces or compiled partitions.
—.h, —.cdf, …	Header files
—.pex	Export file used to import the library into development repositories in a target development environment.
—.lgf	Library generation file. Contains the source code used to make compiled libraries for a single project. (The .lgf extension is added to the library's unique identifier name.)
—.bom, —.cc, —.cdf, …	C++ wrapper files used as source code for making compiled libraries for a single C project.
—.so, —.exe, —.a, —.dll…	Compiled library file. (The name is platform dependent and based on library's unique identifier name.)

Using the Make Distribution Command

In the Partition Workshop, the File > Make Distribution command lets you make a distribution for a library configuration the same way you make an application. The only difference between using the Make Distribution command to create a library distribution is that you have the option of specifying that the source code be included in the library. See "Using the Make Distribution Command" on page 722 for complete information about the Make Distribution command.

➤ **To make a library distribution**

1. Choose the File > Make Distribution command.
2. Select the appropriate distribution options on the Make Distribution dialog. The Include Source toggle lets you specify that the source code be included in the distribution.

When the Make Distribution completes, a message in the status line indicates that the distribution was complete.

If your configuration contains only standard libraries, or if your automatic library compilation completed successfully, you now have a library distribution, ready to install.

If your automatic compilation failed or if you did not select automatic compilation for your compiled libraries, you must compile the individual libraries before you can deploy the library distribution. The following section provides information about compiling libraries.

Compiling Libraries

If your library configuration contains one or more compiled libraries and you did not select automatic compilation when you gave the Make Distribution command (or not all the libraries compiled successfully), you need to compile the libraries yourself using the `fcompile` command.

.lgf files When you specify that a library is "compiled," the Make Distribution command creates a library generation file (.lgf), which you must use with the `fcompile` command to generate the compiled library.

Compile supplier projects first! If your library configuration contains multiple libraries, you must be sure to compile the libraries in the correct order. If any of the projects in the configuration are suppliers for other projects in the configuration, you must compile each of the supplier projects before compiling the main project.

Note that the automatic compilation feature of the Make Distribution command ensures that the libraries are compiled in the correct order. You need to be concerned about compilation order only when using the `fcompile` command.

Platform-specific compilation You need to compile a library on the platform on which it will run. To do so, you move the .lgf file to the appropriate node, compile it, and then move the executable library file (and import file if appropriate) to the library subdirectory for the specific platform in the library distribution directory.

If your library configuration contains more than one library, you need to move and compile the libraries as a group, not one at a time.

➤ **To compile multiple libraries**

1. Move all the .lgf files to the nodes where they need to be compiled.
2. Compile all the libraries.
3. Move all executable library files and import files back to the appropriate subdirectories in the library distribution.

If the environment includes a mixture of platforms, you may need to compile a library on more than one platform. For example, let's say you are compiling a library for an environment that includes three servers: one server is an IBM RS/6000, one is a Sun SPARCStation running UNIX, and a third is a VAXStation running VMS. In this case, you would compile three separate executable library files using the same .lgf file.

Before running `fcompile`, check to see if you need to set environment variables prior to running `fcompile`. These environment variables are described under "Environment Variables and Path" on page 727. You also need the C++ compiler. See *iPlanet UDS System Installation Guide* for the currently supported compiler version.

Using the fcompile Command for Libraries

The `fcompile` command generates code, compiles, and links a compiled library from an .lgf file. See "Using the fcompile Command for Partitions" on page 729 for the complete syntax of the command.

Where to run fcompile When you give the `fcompile` command for a given library, you must do so on machine with the same architecture as the node to which the library is assigned. For example, if you are building a compiled library for an RS6000, you must run `fcompile` on an RS6000.

Do not use a model node You cannot use a model node to create a distribution. Therefore, you should not run the `fcompile` command on a machine that is defined as a model node within the current environment.

➤ **To run fcompile for a compiled library**

1. If necessary, move the .lgf file to a machine that has the correct architecture.
2. Set the necessary environment variables as described in "Environment Variables and Path" on page 727.
3. Run the `fcompile` command as described above.

   ```
   fcompile -c aucio1.lgf -d ./rs6000
   ```

   This produces an executable library file in the format used by the specific platform. It may also produce a library import file.
4. Move the executable library file (and import file if appropriate) to the library subdirectory for the specific platform in the library distribution directory.

iPlanet UDS Example Applications

iPlanet UDS provides a number of example applications that illustrate how to use TOOL and the iPlanet UDS classes. This appendix provides instructions on how to install the examples, a brief overview of the applications to help you locate relevant examples, and a section describing each example in detail. Typically, you run an example application, then examine it in the various iPlanet UDS Workshops to see how it is implemented. You can modify the examples if you wish.

How to Install iPlanet UDS Example Applications

You can access the iPlanet UDS example applications only if they have been installed into your central repository or into a private local repository during installation of iPlanet UDS, or if you have imported them into your repository.

The examples are located in subdirectories under the `FORTE_ROOT/install/examples` directory. The example applications are stored as .pex files. If they are not already installed in your repository, import them by including the `tstapps.fsc` script in Fscript. The `tstapps.fsc` script is located in the `FORTE_ROOT/install/examples/install` directory. Bring up Fscript in standalone mode and issue the following command:

```
fscript> UsePortable
fscript> SetPath %{FORTE_ROOT}/install/examples/install
fscript> Include tstapps.fsc
```

This will import most of the example applications and quit Fscript. Note that certain highly specialized examples are not automatically imported by `tstapps.fsc`.

To run an application, select it in the Repository Workshop's plan browser and then click on the Run button.

If you want to remove all the examples from your workspace, you can do so by including the remprj.fsc script in Fscript. Bring up Fscript in standalone mode and issue the following commands:

```
fscript> UsePortable
fscript> SetPath %{FORTE_ROOT}/install/examples/install
fscript> Include remprj.fsc
```

This will exclude all the example applications and quit Fscript.

Overview of iPlanet UDS Example Applications

This section provides an overview of the iPlanet UDS example applications, organized by general topic. The following tables list the example applications under the particular part of the iPlanet UDS system they demonstrate.

The margin note for each of the following tables shows the name of the subdirectory under FORTE_ROOT/install/examples where you can find the .pex files for the examples. For the complete description of an individual application, see "Application Descriptions" on page 746, which lists the applications in alphabetical order.

General-Purpose Examples

	Example	**Description**
tool/	Auction	Illustrates prominent features of an iPlanet UDS distributed application.
	AuctionServerProject	Acts as server to the Auction project.
frame/	Banking	Acts as a simple distributed application.
tool/	ImageProject	Provides images for the Auction project.
	ImageTester	Retrieves an image using the ImageProject service.

Display Library Examples

	Example	Description
display/	AutoTester	Shows how to use the Capture and Playback classes to automate GUI testing.
	ClipboardSample	Illustrates the use of the Image and TextField classes.
	DynamicList	Shows how to move the contents of an array field to a drop list, fillin field, and scroll list.
	FileBrowser	Illustrates drag and drop, and refreshing multiple windows.
	InheritedWindow	Shows how to subclass your own UserWindow classes.
	MultiList	Shows how to move selected items in a scroll list to an array field.
	NestedWindow	Illustrates multiple tasks, nested windows, event handlers, and input validation.
	PencilPlay	Illustrates the use of the `PictureButton`, `Rectangle`, `Ellipse`, `Line`, and `Point` classes.
	PrintSample	Shows how to use the printing classes.
	SimpleOutline	Illustrates the use of the `OutlineField` class.
	TabFolders	Illustrates the use of the `TabFolder` class and Popup Menus.
	TreeList	Shows how to coordinate data displayed in TreeView and List View fields.

Framework Library Examples

	Example	Description
frame/	AdaptableAuction	Shows how to dynamically load an implementation of an interface.
frame/	Banking1-2	Illustrates how to use convertors to run old and new clients and servers together.
frame/	DVSubClass	Illustrates how to create subclasses of certain iPlanet UDS data type classes.

	Example	Description
frame/	FileUtil	Illustrates the use of the file I/O classes with a command line utility.
extcon/	InboundExternalConnection	Illustrates how to use the ExternalConnection class to listen for a connection.
frame/	NomadicOrderClient	Illustrates how to design an application that uses nomadic clients.
frame/	OLMBanking	Demonstrates how to use the Object Location Manager to manage named objects.
extcon/	OutboundExternalConnection	Illustrates how to use the `ExternalConnection` class to initiate a connection.
frame/	Timeit	Illustrates the use of the `Timer` class.
frame/	TimeItV1-4	Illustrates runtime compatibility issues.
frame/	Utility	Processes commands and handles basic I/O.

AppletSupport Library Examples

	Example	Description
frame/	AppletBanking	Illustrates launching of applets from a main application.
frame/	LauncherGUI	Provides a console interface from which the user can control the execution of applets and iPlanet UDS clients.

GenericDBMS Library Examples

	Example	Description
database/	DynamicDataAccess	Shows how to build dynamic SQL queries.
	DynamicSQL	Illustrates the use of the GenericDBMS classes with a command line utility.
	QueryMgr	Acts as a server for DynamicDataAccess.
	WinDB	Illustrates the use of GenericDBMS classes in a window-based application.

International Examples

	Example	Description
internat/	InternatBank	Uses catalog files to translate windows and error messages.

SystemMonitor Examples

	Example	Description
sysmon/	AgentAccess	Logs agent instrument data to a file.
	AgentBanking	Defines an agent and lets a user interact with the agent.

External Systems Examples—C

	Example	Description
extsys/c/	AllCType	Maps TOOL C data types to variables in C functions. All C data types are covered.
	DMathTm	Shows how to integrate TOOL code with C functions in a distributed application.
	MathTime	Shows how to integrate TOOL code with C functions.
	XRefTime	Shows how to free external resources based on TOOL memory reclamation.

External Systems Examples—C++

	Example	Description
extsys/cpp/server	CPPBanking	Demonstrates how to write a C++ application that uses a generated C++ API to access the services of an iPlanet UDS application.

External Systems Examples—DDE

	Example	Description
extsys/dde/	DDEClient	Illustrates the use of the DDE Conversation class; iPlanet UDS is the client.
	DDEServer	Lets an iPlanet UDS application act as Microsoft Windows DDE server application.

External Systems Examples—ExternalConnections

	Example	Description
extcon	InboundExternalConnection	Illustrates how to use the `ExternalConnection` class to listen for a connection.
extcon	OutboundExternalConnection	Illustrates how to use the `ExternalConnection` class to initiate a connection.

External Systems Examples—OLE and ActiveX

	Example	Description
extsys/ole/client	ActiveXDemo	Demonstrates how to include an ActiveX control in your iPlanet UDS application.
extsys/ole/server	OLEBankingEV	Demonstrates how to write a Visual Basic client that interacts with an iPlanet UDS environment-visible service object.
	OLEBankingUV	Demonstrates how to write a Visual Basic client that interacts with an iPlanet UDS user-visible service object.
extsys/ole/client	OLESample	Illustrates the use of OLEField, `Olegen`, and iPlanet UDS's OLE Automation.

Application Descriptions

This section lists the example applications in alphabetical order. Each example has five sections describing it.

The Description section defines the purpose of the example, what problem it solves, and what TOOL features and iPlanet UDS classes it illustrates.

The Pex Files section gives you the subdirectory and file names of the exported projects. The examples are in subdirectories under the `FORTE_ROOT/install/examples` directory. You can import example applications individually if you wish. When multiple .pex files are listed, there are supplier projects in addition to the main project. You will need to import all the files listed to run the application. Import them in the order given so that dependencies will be satisfied.

The Mode section indicates whether the application can be run in either standalone or distributed mode, or whether it must be run in distributed mode.

The Special Requirements section identifies whether you need a database connection, an external file, or any other special setup.

Finally, the To Use section tells you how to step through the application's functions.

See the *iPlanet UDS System Management Guide* if you need directions to set up an iPlanet UDS server. See *Accessing Databases* if you need information on how to make a connection to a database. The database examples run against either Sybase or Oracle.

ActiveXDemo

Description ActiveXDemo shows how to use ActiveXField widgets to display and interact with ActiveX controls. To use this example, you need the following files:

- actxsamp.pex, which contains a project that uses ActiveX fields to interact with the ActiveX control
- FourDir ActiveX control file, which is one of the following, depending on the platform:

Platform	File name for FourDir ActiveX control file
Windows 3.1	fdir16.ocx
Windows 95	fdir32.ocx
Windows NT	fdir32.ocx
Windows NT on Alpha	fdirant.ocx

Pex File extsys\ole\client\actxsamp.pex.

Mode Standalone.

Special Requirements Windows 3.1, Windows 95, or Windows NT.

➤ To use ActiveXDemo

1. Copy the .ocx file for the FourDir ActiveX control to another location on your system. If you wish, you can copy `olegen.exe` from `FORTE_ROOT\install\bin` directory to the same location.

2. Register the FourDir ActiveX control in the Windows registry. The following table shows how to register the ActiveX control for each platform:

Platform	Command to Register the Control
Windows 3.1	regsvr fdir16.ocx (must be run in the Windows Run Command dialog)
Windows 95	regsvr32 fdir32.ocx
Windows NT	regsvr32 fdir32.ocx
Windows NT on Alpha	regsvr32 fdirant.ocx

The `regsvr.exe` and `regsvr32.exe` are distributed with iPlanet UDS in the `FORTE_ROOT\install\bin` directory.

For example, in Windows 95 or Windows NT, use the following command on the command prompt in the directory that contains the .ocx file for the control:

```
regsvr32 fdir32.ocx
```

3. Use the `Olegen` utility to generate a pex file based on the .ocx file using a command like the following:

```
olegen -it fdir32.ocx -of fdir32.pex -ai
```

This command generates a .pex file called `fdir32.pex`.

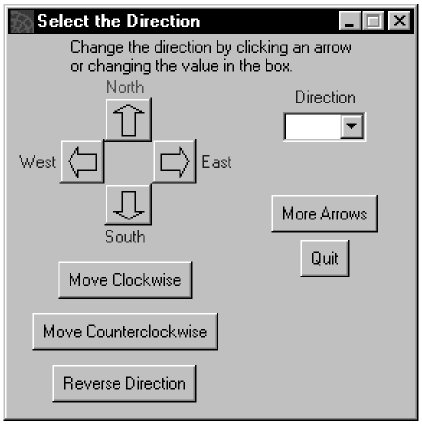
4. Import the generated .pex file into your repository.

5. Import the `actxsamp.pex` file into your repository.

6. Test run the ActiveXDemo application.

The window in the ActiveXDemo application is shown in the following figure:

Figure A-1 ActiveXDemo Window



Change the direction by clicking an arrow or changing the value in the box.

The four arrows in this window belong to the FourDir ActiveX control in an ActiveX field. The direction indicated by the selected arrow is reflected in the Direction droplist, and vice-versa. Another ActiveX field is defined in the lower right hand corner, but invisible.

The functions provided by the buttons are described below:

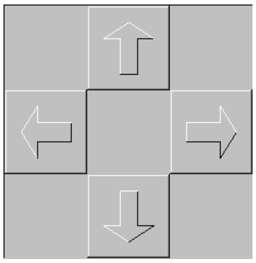
Button	**Description**
More Arrows	Sets the state of the invisible ActiveX field to update, then creates a new instance of the fdir class (the ActiveX interface class for the FourDir ActiveX control) and inserts it into the ActiveX field. When you click one of the arrows in this control, the selected arrow moves clockwise until it returns to the arrow you selected.
Move Clockwise	Makes the selected arrow the next one in a clockwise direction.
Move Counterclockwise	Makes the selected arrow the next one in a clockwise direction.
Quit	Shuts down this application.

Button	Description
Remove Arrows	Removes the FourDir ActiveX control added by the More Arrows button. When the More Arrows button is clicked, this button replaces it.
Reverse Direction	Makes the selected arrow the one pointing in the opposite direction from the arrow currently selected.

FourDir ActiveX Control

The FourDir ActiveX control is a sample ActiveX control provided by iPlanet UDS to demonstrate how you can use ActiveX controls in your iPlanet UDS applications.

Figure A-2 The FourDir ActiveX Control



The FourDir ActiveX control is provided as an .ocx file, as described in "ActiveXDemo" on page 747. This section also describes how to install the FourDir ActiveX control.

The following table outlines the methods, properties, and events defined for the FourDir ActiveX control:

Element	Name	Description
Property	Value	String. Sets the initial direction for the selected arrow. Valid values are N (north), S (south), E (east), or W (west). By default, the initial value is N.

Element	Name	Description
Method	MoveClockwise	No arguments and no return value. Changes the selected arrow to the next arrow in the clockwise direction.
	MoveCounterClockwise	No arguments and no return value. Changes the selected arrow to the next arrow in the counterclockwise direction.
	MoveOpposite	No arguments and no return value. Changes the selected arrow to the arrow in the opposite direction.
Event	Click	Posted when someone clicks an arrow in the control.

AdaptableAuction

Description AdaptableAuction shows how to use methods on the Partition and Library classes to dynamically load implementations of an interface.

A simple auction application calculates the taxes on the items sold. A tax calculation interface class provides the signature for a method which will calculate the tax on a sale item. Two implementations of this interface are provided. The implementations must be configured as libraries, in order to be dynamically loaded. The AdaptableAuction application reads from a file to determine at runtime which of the two implementations to load. You can copy over this file while the application is running and it will load the other implementation, which calculates the tax differently.

Pex Files frame/aa.pex

Mode Standalone or Distributed.

Special Requirements You will need to configure the two implementations as libraries and distribute them. You will need to copy a file while the application is running to see the dynamic loading behavior.

➤ **To use Adaptable Auction**

1. The two files `dynload1.dat` and `dynload2.dat` provide the library and class information used by the application at runtime. The application expects a file named `dynload.dat` to be in `FORTE_ROOT/install/examples/frame`. Copy `dynload1.dat` to `FORTE_ROOT/install/examples/frame/dynload.dat`.

2. Import the aa.pex file, which includes the interface project, two implementations of that interface, and the AdaptableAuction client and server projects.

3. Configure the two implementation projects, AAImplementations and AAImp2, as libraries. To do this, open the project workshop for each project. Select File > Configure as | Library... Then select File > Make Distribution... and perform a full make in the current environment. Once they have been distributed as libraries, AAImplementations and AAImp2 can be deleted from your workspace.

4. Run AdaptableAuction. Click the Calculate Tax for Sale Items button. This will write information to the log. The data used to load the library and find the class is written to the log. The log also shows information on each of three sale items, including the tax amount. In each of the three sale items, the tax is seven percent of the value of the bid amount. You can click the Calculate Tax for Sale Items button several times, to confirm that the same output is written to the log each time.

5. Do not exit the application. Copy the file `dynload2.dat` to `FORTE_ROOT/install/examples/frame/dynload.dat`. Click the Calculate Tax for Sale Items button again. Note that a different library and class have been loaded this time. Also note, the second bidder's tax amount is now 0. The second implementation checked the bidder type. Since the second bidder was a nonprofit organization, he was not charged any taxes.

AgentAccess

Description AgentAccess shows how to retrieve information from one or more system-management agents and log this data into a file.

Pex Files sysmon/agentasv.pex, sysmon/agentacc.pex.

Mode Standalone or Distributed.

Special Requirements You need to use Escript or the Environment Console to set up agent instruments to be logged by this application at regular intervals.

➤ To use AgentAccess

1. Select instruments that will be logged by this application, and set them to be logged using the Environment Console or Escript. Set instruments for logging by first locating the agent that owns the instruments, then by setting each instrument's `isLogged` property to `TRUE`. For example, to log instrument data from the DistObjectMgr agent, you can use a series of Escript commands, like the following:

```
escript> ShowAgent
escript> FindSubAgent <node_name>
escript> showag
escript> findsub Forte_Executor
escript> showag
escript> findsub <partition_identifier>
escript> showag
escript> findsub DistObjectMgr
escript> showag
escript> SetInstrumentLogging MethodsReceived TRUE
escript> SetInstrumentLogging MethodsSent TRUE
```

   This series of commands assumes that you have just started Escript, or are at the Environment Agent before you issue these commands. The `ShowAgent` command, abbreviated as `showag`, displays information about the current agent, including a list of instruments and subagents. The `FindSubAgent` command, abbreviated as `findsub`, moves to a subagent of the current agent, based on the name of the subagent.

   To use this series of commands, you need to substitute the name of a node that has a running standard partition for <node_name>. You need to substitute the hexadecimal active partition identifier for <partition_identifier>. You can see the list of available active partitions in the listing provided by the previous `showag` command. For example, if the active partition name is Forte_Executor_0x14d, you can substitute 0x14d for <partition_identifer>.

   For information about setting instruments for logging using Escript or the Environment Console, see *iPlanet UDS System Management Guide*.

2. Set the LogTimer for the Active Partition agent where the ActivePartition or one of its subagents contains the instrument. Starting from the ending point in the above example, you can use a series of Escript commands like the following to set a LogTimer to tick every 30 seconds:

```
escript> FindPar
escript> UpdateInstrument LogTimer "TRUE 30000"
```

   In this series of commands, the `FindParent` command (abbreviated as `findpar`) moves to the parent agent of the DistObjectMgr agent, which is the Active Partition agent. The LogTimer instrument has a property that determines whether it is ticking or not; the `TRUE` value enables the LogTimer. The value 30000 means that the LogTimer will tick every 30 seconds (30000 milliseconds).

   For more information about setting the LogTimer for an Active Partition agent in Escript, see *iPlanet UDS System Management Guide*. For instructions for setting the LogTimer in the Environment Console, see *iPlanet UDS System Management Guide*.

3. When you start AgentAccess, you should define the Log File name as a file that does not currently exist. The default log file name is agent.log, and the file is always stored in FORTE_ROOT/log.
4. Select the Start button to start logging instrument data to your log file.

AgentBanking

Description AgentBanking shows how to implement an agent for the service object of a simple banking application. This application also provides a simple administrator's window that lets the user manage the AgentBanking application using instruments and commands provided by the service object's agent.

Pex Files sysmon/agentbsv.pex, sysmon/agentb.pex.

Mode Distributed only.

Special Requirements You need to install this application to be able to see that the agent is working. This application is designed to run with several clients in a distributed environment. While it will work with a single client, some features do not make sense with only one client running.

➤ To use AgentBanking

1. Partition the AgentBanking application with the AgentBanking project as the main project, using either the Partition command in the Partition Workshop or the `Partition` command in Fscript.

2. Make a distribution of the application and auto-install the application in your environment, using either the Make Distribution command in the Partition Workshop or the `MakeAppDistrib` command (with the arguments 1 "" "" 1) in Fscript.

3. Start the application using either the application icon, or a command like the following:

   ```
   ftexec -fi ct:$FORTE_ROOT/userapp/agentban/cl0/agentb0
   ```

   For more information about starting client partitions, see *iPlanet UDS System Management Guide*.

4. In the Banking: Welcome! window, enter any numeric value for the user ID. (Note that each time you log in, you should enter a different user ID.) Select either the User or Administrator buttons, then select the Start button.

   If you login as a user, then you can select an account to work with, then add or subtract amounts of money from the selected account.

   If you login as an administrator, you can perform actions that affect the running banking application by invoking commands and reading and updating instruments on the BankServiceAgent agent for the service object of the banking application.

5. You can also monitor the BankServiceAgent agent by using the Environment Console or Escript at the same time as this application and looking at the instrument values of the BankServiceAgent. This agent is a subagent of the Active Partition agent for the server partition of the banking application. For information about using the Environment Console or Escript, see *iPlanet UDS System Management Guide*.

AllCType

Description AllCType shows how to map TOOL C data types to variables in C functions. The "mapping methods" in this example are the methods defined in a TOOL C project which enable TOOL methods to access C functions. The methods and functions in this example perform extremely simple operations. Their purpose is to show how to define input, output, and input output parameters, and return values in the mapping methods, and how to call those methods from TOOL, and how to de-reference the parameters in the C functions. Also see the MathTime and DMathTm examples for a simple, practical example of how to use mapping methods. DMathTm is the distributed version of MathTime.

Pex Files extsys/c/allctype.pex.

Mode Distributed only.

Special Requirements Access to a C++ Compiler, creation of a working directory, autocompile must be turned on.

➤ **To use AllCType**

1. Create a working directory where you have read and write permission. Copy the following three files from `FORTE_ROOT/install/examples/extsys/c` to your working directory: `allctype.pex`, `allctype.fsc`, `allctype.c`. Set the environment variable FORTE_EP_WRKDIR to your working directory.

2. Compile the file `allctype.c` into an object file called `allctype.o`. Under the directory `FORTE_ROOT/tmp`, create the directory '`examples`', if it isn't there already. Copy the file `allctype.o` to the `FORTE_ROOT/tmp/examples` directory.

3. Before completing this step, make sure autocompile is available on your system. If autocompile is not set up, ask your System Administrator to set it up for you. Now run Fscript and enter the following commands:

```
UsePortable
SetPath %{FORTE_EP_WRKDIR}
Include allctype.fsc
```

4. The `allctype.fsc` script will import, distribute, compile, install, and run the AllCType example. You may want to examine this script to understand the steps involved in linking TOOL code with external C routines.

AppletBanking

Description This application starts two applets that perform its functions: Banking and BankRecords.

Pex Files In the `FORTE_ROOT/install/examples/frame` directory: apltbank.pex, banksvc.pex, banking.pex, bankrec.pex

Mode Distributed.

Special Requirements Banking and BankRecords must be installed in your environment as applets. The `apltbank.fsc` file contains the Fscript commands to import, make distributions, and autoinstall Banking and BankRecords as applets.

The AppletSupport library is a supplier to the AppletBanking project. If this library is not already in your repository, you need to import the `apltsupp.pex` file from the `FORTE_ROOT/userapp/appletsu/cl0` directory before you can import the apltbank.pex file.

You can run the AppletBanking example either as a test run from within the iPlanet UDS Workshops, or you can deploy the application.

➤ **To use AppletBanking**

1. Start the AppletBanking application.
2. Click a button to perform a function:

Button	Description
Manage Accounts	Starts the Banking application in a separate window.
View Account Data	Starts the BankRecords application in a separate window.
Quit	Stops the AppletBanking, and also stops Banking and BankRecords, if they have been started by AppletBanking.

When a button is clicked, the application invokes the `RunApplet` method on the LaunchService service object to start the appropriate applet.

For more information about how this application works, see the *iPlanet UDS Programming Guide* and the Framework Library online Help.

Auction

Description Auction illustrates prominent features and capabilities of an iPlanet UDS distributed application: GUI independence, distributed processing, event handling, multitasking, and image handling. The application allows a number of bidders located at their respective computers to bid on a set of paintings being offered by an auctioneer located at some other computer. The Art Auction application provides a list of paintings available for bidding and notifies interested bidders when a price changes.

Pex Files frame/utility.pex, tool/imageprj.pex, tool/aucserv.pex, tool/auction.pex.

Mode Standalone or Distributed.

Special Requirements The image files used by this application must be located in FORTE_ROOT/install/examples/images.

➤ **To use Auction**

1. Start up the auction by clicking the Be Auctioneer option in the radio list, then clicking the Start Auction button.
2. Assume the role of a bidder by clicking the Be Bidder option in the radio list. You should click on a painting in the array, then click the View Painting button.

   From the painting window, you can double-click on the image to see it enlarged. You can also click the Bid button to set a bid.
3. Another bidder can view available paintings being offered and then join the bidding process.

   Both bidders become involved in bidding on the same painting. In the standalone use of this application, you can simulate a second bidder on the same screen by opening a second bidding window.

AutoTester

Description AutoTester enables you to create test suites of iPlanet UDS GUI applications. It shows how to use the Capture and Playback class in the Display Library. You may find it covers all your automated testing needs. You may want to modify it for special testing purposes, or just use it as a reference when creating your own test utility.

Pex Files display/autotest.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use AutoTester**

1. Refer to the testing chapter in *iPlanet UDS Programming Guide* for a complete description of how to run AutoTester on the PencilPlay sample program.

Banking

Description Banking is a simple distributed application that demonstrates a client partition that starts a server partition.

Pex Files frame/banksvr.pex, frame/banking.pex.

Mode Distributed.

Special Requirements None.

➤ **To use Banking**

1. Test run the application within the iPlanet UDS Workshops, or make distribution and install the application, as described in *A Guide to the iPlanet UDS Workshops*.
2. In the Select Account window, choose an account number and Click the Display Info button.
3. In the Transactions window, enter a dollar amount in the Transaction Amount field, then click the Deposit or Withdraw buttons.
4. To exit this application, click the Done button in the Transactions window, then click the Quit button in the Select Account window.

Banking1-2

Description Banking1-2 example illustrates iPlanet UDS's interoperability features. It creates an old server, a new server, an old client and a new client, and allows you to run any combination of client and server.

Pex Files frame/banking1.pex, frame/banking2.pex, frame/banksvc1.pex, frame/banksvc2.pex, interop.fsc.

Mode Distributed only.

Special Requirements None.

➤ **To use Banking1-2**

1. Run fscript and include `interop.fsc`.

   ```
   fscript -i interop.fsc
   ```

2. Start a new server, and run both old and new clients with it.

```
-- Use the following commands to start servers and clients:
-- To start a new server:
ftexec -fi bt:$FORTE_ROOT/userapp/bankserv/cl0/bankse1 -ftsvr 0
-- To start an old server:
ftexec -fi bt:$FORTE_ROOT/userapp/bankser0/cl0/bankse1 -ftsvr 0
-- To start a new client:
ftexec -fi bt:$FORTE_ROOT/userapp/banking/cl0/bankin0
-- To start an old client:
ftexec -fi bt:$FORTE_ROOT/userapp/banking0/cl0/bankin0
```

3. Start an old server, and run both old and new clients with it.

ClipboardSample

Description ClipboardSample uses the Image and TextField classes, and the clipboard methods on WindowSystem. It allows you to move images and text to and from a clipboard.

Pex Files display/clipbrd.pex.

Mode Standalone or Distributed.

Special Requirements PC client.

➤ **To use ClipboardSample**

1. You must load a clipboard viewer, and a paint program.
2. You can move the image and text from the ClipboardSample screen to the clipboard, using the Move Image and Move Text buttons.
3. The image and text can be brought into the paint program, modified, placed on the clipboard and then moved back into the ClipboardSample program using the Copy From Clipboard button.

CPPBanking

Description CPPBanking shows how to create an iPlanet UDS service object for which you can generate a C++ API. This example also shows how to generate a C++ API and how to write a C++ client that uses the API.

- cppbank.pex contains the TOOL project CPPBanking, which contains a simple class and starting method that references the BankServer service object in the BankServices project.
- cppbancl.cpp is the C++ client application that uses the generated C++ API to access an iPlanet UDS client partition.

BankServer is a simple bank account service object. That lets clients query and update bank accounts.

Pex Files frame/banksvc.pex, extsys/cpp/server/cppbank.pex.

Mode Distributed only.

Special Requirements Access to a C++ compiler, set up autocompile, set up your environment and c++ compiler and linker, as described in *Integrating with External Systems*.

➤ **To use CPPBanking**

1. Import the `banksvc.pex` (BankServices project) and `cppbank.pex` (CPPBanking project) files into your repository.
2. Partition the CPPBanking project.
3. Open the properties dialog for the client partition for the CPPBanking application.
4. Mark the Generate C++ API toggle and click the OK button.
5. Make a distribution for this application using autocompile and autoinstall.
6. Compile the `cppbancl.cpp` file using the compiler and linking options described in *Integrating with External Systems*.

DDEClient

Description DDEClient uses the DDEConversation class, which lets an iPlanet UDS application access a Microsoft Windows Dynamic Data Exchange (DDE) server application on a PC/Windows platform. It allows you to establish a connection with Excel and move data to an Excel spreadsheet.

Pex Files extsys/dde/ddecli.pex.

Mode Standalone or Distributed.

Special Requirements PC client running Excel, access to `extsys/dde/ddecli.xls`.

➤ **To use DDEClient**

1. Before trying to run this application, check the location of your Excel executable.

   If it is not in `C:\EXCEL`, edit the Display method in the DDEClientWindow class to point to the right directory. Enter the full path name to your Excel spreadsheet and click on the Connect button. If Excel is not running, it will be started and Already Running will be checked.
2. Place the windows so that both the DDEclient and Excel are visible. You can retrieve data from a particular cell of the spreadsheet by specifying the cell name and clicking the Get button. Similarly, you can place data by entering the Cell Value and clicking the Set button.

3. You can also change data in the Excel spreadsheet, click on the HotLink and WarmLink buttons, and note the status line at the bottom of the application. A hot link changes the data in the client display, while a warm link only notifies you of a change.

DDEServer

Description DDEServer uses the DDEServer and DDEClient classes, which let an iPlanet UDS application act as a Microsoft Windows Dynamic Data Exchange (DDE) server application on a PC/Windows platform. It is a simple utility for servicing a DDE client application.

Pex Files extsys/dde/ddeserv.pex.

Mode Standalone or Distributed.

Special Requirements PC client running Excel, access to `extsys/dde/ddeserv1.xls and ddeserv2.xls`.

➤ **To use DDEServer**

1. Start the application, then bring up Excel and open the example Excel files: `ddeserv1.xls` and `ddeserv2.xls`.
2. Select the StartTimer button. You should see the changing numbers in the iPlanet UDS server reflected in your example spreadsheets.
3. You can also use the appropriate menu items in Excel to retrieve data from the DDEserver application and place in the Excel spreadsheet, or to export data from the spreadsheet and place it in the DDEserver application.

DMathTm

Description DMathTm is the distributed version of MathTime. Both DMathTm and MathTime are examples of a TOOL C project, along with a TOOL project that calls the TOOL C project. They are both useful for seeing how to integrate TOOL code with C functions. DMathTm shows how to use a service object to restrict access to the C project. This is a realistic approach to accessing C functions in a distributed environment, since pointers cannot be passed across partitions. The example program allctype is a reference for how to define and call TOOL C methods with parameters of all C data types at assorted levels of indirection.

Pex Files extsys/c/dmathtm.pex.

Mode Distributed only.

Special Requirements Access to standard C Runtime Libraries and a C++ Compiler, creation of a working directory, autocompile must be turned on.

➤ **To use DMathTm**

1. Create a working directory where you have read and write permission. Copy the following three files from `FORTE_ROOT/install/examples/extsys/c` to your working directory: `dmathtm.pex`, `dmathtm.fsc`, `dmathtm.c`. Set the environment variable FORTE_EP_WRKDIR to your working directory.

2. The file `dmathtm.pex` contains the C project DistMathAndTimeProject and the TOOL project TestDistMathAndTimeProject. They assume you have access to the standard C runtime libraries. Make sure you know where these are located and what they are called on your system.

3. Edit your copy of `dmathtm.pex` so that its ExternalSharedLibs extended property points to the standard C shared library. Search the file for the string '`/usr/shlib/libc`'. Change this string to the correct path and library name for your system.

4. Compile the file dmathtm.c into an object file called dmathtm.o. Under the directory `FORTE_ROOT/tmp`, create the directory '`examples`', if it isn't there already. Copy the file `dmathtm.o` to the `FORTE_ROOT/tmp/examples` directory.

5. Before completing this step, make sure autocompile is available on your system. If autocompile is not set up, ask your System Administrator to set it up for you. Now run Fscript and enter the following commands:

```
fscript> UsePortable
fscript> SetPath %{FORTE_EP_WRKDIR}
fscript> Include dmathtm.fsc
```

The `dmathtm.fsc` script will import, distribute, compile, install, and run the DMathTm example. You may want to examine this script to understand the steps involved in linking TOOL code with external C routines.

DVSubClass

Description DVSubClass illustrates how to create subclasses of certain iPlanet UDS data type classes. It shows how you can provide additional input validation and string interpretation for a subclassed data type, while still benefitting from all the methods, attributes, and events defined on the superclass. There are two data type subclasses in this example: `GoodDateTime`, a subclass of `DateTimeNullable`, limits the acceptable range of dates. `StockPrice`, a subclass of `DecimalNullable`, allows stock prices to be entered as decimal values or as fractions.

Pex Files frame/dvsubcl.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use DVSubClass**

1. When you start the application, the window shows a default date and two stock fields.
2. Enter a date that is before 1984 and press Return when you see the error message.
3. Enter a date that is after 2007 and press Return at the error message again.
4. Enter a date between 1984 and 2007.
5. In the first stock field, enter a decimal value. You will see it displayed as an integer and a fraction. The second stock field will display twice the value of the decimal you entered.
6. In the first stock field, enter a value in the form of integer <space> fraction, where fraction is 1/4, 1/2, or 3/4. Again the second stock field will display twice that value.
7. Enter an invalid stock value such as 33 1/3. You will see an error message. Note that the value in the second stock field is two times the value of the decimal data, not the integer and fraction display.

DynamicDataAccess

Description DynamicDataAccess lets you construct ad hoc SQL queries. The data fields used to display the data and the text graphics used for labels are created at runtime to match the columns in the dynamic query.

Pex Files database/querymgr.pex, database/dda.pex.

Mode Distributed only.

Special Requirements Database connection. The files `artist.dat` and `painting.dat` must be located in `FORTE_ROOT/install/examples/database`.

➤ **To use DynamicDataAccess**

1. Before running DynamicDataAccess, open the Project Workshop for the QueryManager project.
2. Open the DBResourceMgr service object properties dialog. Make sure the Database Manager value is correct for your database connection.
3. Run DynamicDataAccess. You will be prompted for a User Name, Password, and Database. You must provide all three values.

   This application will create its own tables and data in the database you selected. The data is read in from the files `FORTE_ROOT/install/examples/database/artist.dat` and `painting.dat`.
4. Choose Select from the radio list to select from existing artist and painting values. Choose Insert to add your own data. Follow the prompts in the Insert and Select Windows to construct SQL statements and view the results.

DynamicList

Description DynamicList uses the Array, PushButton, DropList, FillInField and ScrollList classes. It allows you to move the contents of an array field to a drop list, fillin field, and scroll list.

Pex Files display/dynlist.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ To use DynamicList

1. Insert a new first row in the array field at the top of the screen using the Ins button (and then edit the text).
2. You can delete the last row in the array using the Del button.
3. The primary function of this utility, however, is to move the current contents of the array field to the various list fields using the Move List button, automatically resizing these fields to accommodate entries in the array field.

DynamicSQL

Description DynamicSQL is a command line utility illustrating the use of the GenericDBMS Library classes. It lets you perform standard SQL database access commands.

Pex Files frame/utility.pex, database/dynsql.pex.

Mode Distributed only.

Special Requirements Database connection, table creation (if necessary).

➤ To use DynamicSQL

1. You need an environment that has a node with a resource manager. Before running DynamicSQL, open the DynamicSQL Project Workshop. Open the AnyDBMgr service object property sheet. Make sure the Database Manager value is correct for your database connection. Save you changes and exit iPlanet UDS.
2. DynamicSQL does not create any tables. You must either create them ahead of time, or use an existing test table. If you need to create a test table, the files `maketst.syb` and `maketst.ora` are provided in `FORTE_ROOT/install/examples/database`. If you will be using maketst.syb, edit the first line to use an existing database. For example, create a database called testapps, then edit maketst.syb to start with:

   ```
   use testapps
   ```

   Use the standard mechanism for redirecting the maketst file to load the data into your database.

3. Run DynamicSQL from fscript. You will need to know the name of the database you will use, and a userid and password. Start fscript and enter the following commands:

```
Open
FindPlan DynamicSQL
Run
```

4. At the prompts enter your database name, then your userid and password. You should then see a SQL> prompt. If you used the maketst file, enter:

   ```
   SQL> select * from alltypes;
   ```

5. You should see the data displayed. Then enter:

```
SQL> select * from alltypes where intcol = :a;
Enter values for the following placeholders:
a:> 1
Reexecute? (Y/N)y
Enter values for the following placeholders:
a:> 5
Reexecute? (Y/N)y
```

6. To end your session, type:

   ```
   SQL> exit
   ```

7. To end your fscript session, type:

```
Commit
Exit
```

FileBrowser

Description FileBrowser illustrates drag and drop functionality and refreshing multiple windows. As its name implies, it is a file browser similar to the Windows File Manager.

Pex Files display/fileb.pex.

Mode Standalone or Distributed.

Special Requirements Create a couple of temporary directories with junk files in them.

➤ **To use FileBrowser**

1. Browse any drive and directory by entering it in the Volume field at the top left of the window and clicking the Tab key.
2. You can expand directories using the triangular expansion tool.
3. To delete a file or directory (be careful, this is the real thing!), select an item in the browser window and click on the trash can.
4. To open a second browser window double-click on the “Volume” label. With a second window open you can copy between windows by dragging an entity from one browser window and dropping it in the other.

FileUtil

Description FileUtil is a command line utility that uses the File I/O classes. It lets you perform standard command line file operations.

Pex Files frame/utility.pex, frame/fileutil.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use FileUtil**

1. After starting up the application, make sure that the iPlanet UDS Workshop Interpreter Console window is visible. It is this application’s only window.

2. At the prompt (forte>) enter a question mark ('?') to get a list of FileUtil commands.
3. Try using the available commands. To exit the application, type 'quit'.

ImageTester

Description ImageTester retrieves an image using the ImageProject service. It is normally used to start up the ImageProject service in conjunction with a demonstration of the Auction application. It can also be used to set up a reference partition in conjunction with Auction.

Pex Files frame/utility.pex, tool/imageprj.pex, tool/imagetst.pex.

Mode Standalone or Distributed.

Special Requirements The image files used by this application must be located in `FORTE_ROOT/install/examples/images`.

➤ **To use ImageTester**

1. Enter a bitmap graphic file name in the Name field and click on the GetImage button.

   Graphic files, for example mona.fso, can be found in `FORTE_ROOT/install/examples/images`. The .fso suffix is automatically appended. Enter the filename without the suffix. For example, enter 'mona'.

InboundExternalConnection

Description InboundExternalConnection illustrates how to use the ExternalConnection class to listen for a connection. The iPlanet UDS program waits for a new connection, then starts a task to handle each new connection. The C program extcon will initiate the connection this example is waiting for. Once the connection is established, data is read and written. For the read, the iPlanet UDS program checks for an end of string marker to make sure all the data is received.

Pex Files inbound.pex.

Mode Distributed.

Special Requirements C compiler; C portion of this example will run on NT and Unix platforms; it will not run on Windows or VMS.

➤ To use InboundExternalConnection

1. Decide which platform you want to run the C program on, and which platform you want to run the iPlanet UDS program on. Compile the C program `extcon.c` into the executable extcon on the desired platform.

   On most Unix systems, simply use the following command:

   ```
   cc extcon.c -o extcon
   ```

   This will work on the following platforms:

   - AlphaOSF
   - RS6000
   - Solaris
   - Data General

   On Sequent, use the following command:

   ```
   cc extcon.c -o extcon -lsocket -linet -lnsl
   ```

   On HP, use the following command:

   ```
   cc +Z extcon.c -o extcon
   ```

   On NT, if you use Visual C to compile extcon.c, make sure to include `wsock32.lib` with your standard Object/Library modules. Also, make sure the application is defined as a console application, not a windows application.

2. Both the iPlanet UDS program and the C program will use a default port number for the listener, unless you supply it as an environment variable. The default port number is 6867. If you need to use another port number, set the environment variable FORTE_EP_REG_PORT_1 to the desired port number in both the environment where you will run the iPlanet UDS program and the environment where you will run the C program.

3. If you want to establish an external connection between the iPlanet UDS program and the C program running on the same Unix machine, you do not need to set an environment variable for the node name. If you want to connect between different machines, or if you want to make the connection on the same NT machine, you will need to set an environment variable. Set the environment variable FORTE_EP_NODENAME_1 in the environment where you are running the C program. Set it to the name of the machine where the iPlanet UDS program is running.

4. Use the file `inbound.scr` to supply the necessary commands to fscript. `Inbound.scr` will import the pex file `inbound.pex`, find the project, run it, and remove the project after the run is complete. Inbound.pex must be in the same directory as `inbound.scr`. Use fscript's `-i` flag to input `inbound.scr` to fscript:

   ```
   fscript -i inbound.scr
   ```

   Wait for fscript to import the project, load it, partition the service object, and return the client partition. The iPlanet UDS program is now waiting to accept an inbound connection.

5. On the machine where you compiled extcon, run it with the m command line option:

   ```
   extcon m
   ```

   When you use the m option, extcon attempts to make a connection.

6. Observe the output of both processes. On the iPlanet UDS side, you should see the following results:

   ```
   Waiting to connect on port 6867
   Waiting to connect on port 6867
   Inbound Connection: server read got back 34 bytes
   Lab<EOW1>50<EOW2>Stable<EOW3><EOS>
   Inbound Connection: server wrote 35 bytes
   Inbound Connection: server read got back 46 bytes
   Storage Shed<EOW1>90<EOW2>Emergency<EOW3><EOS>
   Inbound Connection: server wrote 35 bytes
   Inbound Connection: server read got back 38 bytes
   Vat<EOW1>200<EOW2>Red Alert<EOW3><EOS>
   Inbound Connection: server wrote 35 bytes
   Inbound Connection: RemoteAccessException caught in
   ProcessConnection
   Connection closed. All done.
   ```

   From the C program, you should see the following lines:

   ```
   Attempting to initiate connection on port 6867.
   Attempting to initiate connection on the current machine.
   Thank you for the information.
   Thank you for the information.
   Thank you for the information.
   ```

InheritedWindow

Description InheritedWindow shows how to create subclasses of your own UserWindow classes. In this example, the parent window has decorative widgets at the top, and three buttons at the bottom. It is a generic data entry window. Two windows are subclassed from this parent window. One is for entering information about art, the other about artists. They both use their own event handlers to validate data, and call the parent window's event handler to perform exit processing.

Pex Files display/inherwin.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use InheritedWindow**

1. The first window lets you call up the Art Data Entry window or the Artist Data Entry window. They are spawned as tasks, so you can have them both open at once. They both have field and cross-field validation.
2. In the Artist window, try entering various countries. When you click OK, you will be warned that only certain names are compatible with that country. In the Art window, try entering various types of art.
3. Try entering 'Performance' as the type and '1910' as the year, then click the OK button.

   In both windows, the Reset button will restore the original data, and the Cancel button will let you exit even if the data is invalid. The OK button exits if the data is valid, or keeps you in the grid if it is not.

InternatBank

Description InternatBank shows how to use several of the iPlanet UDS international features. It can initially be brought up in several languages, and the user can choose to switch languages at runtime. It uses message catalogs to translate windows and error and informational messages.

Pex Files internat/internat.pex.

Mode Standalone or Distributed.

Special Requirements Copy files from `FORTE_ROOT/install/examples/internat` subdirectories to corresponding directories under `FORTE_ROOT/workmsg`.

➤ **To use InternatBank**

1. If you are running this application on a machine other than a PC, be sure your FORTE_LOCALE environment variable is set correctly.
2. Below the examples directory, you will find a subdirectory called internat. Below it are the directories for each of the languages supported by the InternatBank example: en_us, fr_fr, and de_de. In each of these directories is a file called internat.cat. This is the compiled message catalog for each language. Examine the text file that is compatible with the machine you are on. The compiled message file, internat.cat, is portable across all platforms.
3. Before running InternatBank, you will need to copy the internat.cat files from the language subdirectories under internat to the appropriate subdirectories under `FORTE_ROOT/workmsg`.

   For this application, you should create three subdirectories under workmsg: en_us, fr_fr, de_de.

   Copy the appropriate internat.cat files from the example program language subdirectories to the workmsg subdirectories.
4. Choose the language you want the application to start in. At the first window, select the language you wish to use.

   You will see the screen change to that language immediately.
5. When you enter a numeric id, and click OK, the next screen will appear in the chosen language. Numeric, date time, and money fields should display values with appropriate formatting. You can move the Account Information window aside, since it is started as a separate task, and change the language on the first window. When you click OK, the Account Information window will come up in the newly requested language.
6. To see the application start in French, exit iPlanet UDS, set the language/territory component of your FORTE_LOCALE environment variable to fr_fr. If you are using a PC, use the Control Panel to change the FORTE_LOCALE. Restart iPlanet UDS and run InternatBank. The first screen will be in French. You can still change to other languages at runtime.

LauncherGUI

Description LauncherGUI is used to run both iPlanet UDS applets and standards clients. It provides a console interface from which the user can control the execution of applets and the installation of iPlanet UDS clients.

Pex Files FORTE_ROOT/userapp/appletsu/cl0/apltsupp.pex, frame/launcher.pex.

Mode Distributed only.

Special Requirements Before loading the launcher.pex file, you must load the AppletSupport Library. The location of this .pex file is described above.

➤ **To use LauncherGUI**

1. The LauncherGUI is provided as an image with the iPlanet UDS product set and is called the iPlanet UDS *Launcher Application*. See "Using the iPlanet UDS Launcher Application" on page 122 for a full description of how to use the LauncherGUI example.

MathTime

Description MathTime is an example of a TOOL C project, along with a TOOL project that calls the TOOL C project. It is useful for seeing how to integrate TOOL code with C functions. The example program DMathTm is the distributed version of MathTime. The example program AllCType is a reference for how to define and call TOOL C methods with parameters of all C data types at assorted levels of indirection.

Pex Files extsys/c/mathtime.pex.

Mode Distributed only.

Special Requirements Access to standard C Runtime Libraries and a C++ Compiler, creation of a working directory, autocompile must be turned on.

➤ To use MathTime

1. Create a working directory where you have read and write permission. Copy the following three files from `FORTE_ROOT/install/examples/extsys/c` to your working directory: `mathtime.pex`, `mathtime.fsc`, `mathtime.c`. Set the environment variable FORTE_EP_WRKDIR to your working directory.

2. The file mathtime.pex contains the C project MathAndTimeProject and the TOOL project TestMathAndTimeProject. They assume you have access to the standard C runtime libraries. Make sure you know where these are located and what they are called on your system.

3. Edit your copy of mathtime.pex so that its ExternalSharedLibs extended property points to the standard C shared library. Search the file for the string '`/usr/shlib/libc`'. Change this string to the correct path and library name for your system.

4. Compile the file mathtime.c into an object file called mathtime.o. Under the directory `FORTE_ROOT/tmp`, create the directory '`examples`', if it isn't there already. Copy the file `mathtime.o` to the `FORTE_ROOT/tmp/examples` directory.

5. Before completing this step, make sure autocompile is available on your system. If autocompile is not set up, ask your System Administrator to set it up for you. Now run Fscript and enter the following commands:

```
fscript> UsePortable
fscript> SetPath %{FORTE_EP_WRKDIR}
fscript> Include mathtime.fsc
```

6. The mathtime.fsc script will import, distribute, compile, install, and run the MathTime example. You may want to examine this script to understand the steps involved in linking TOOL code with external C routines.

MultiList

Description MultiList illustrates multiple selection capability. It allows you to move selected items in a scroll list to an array field.

Pex Files display/mlist.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use MultiList**

1. Move the contents of the array field at the top of the window to the scroll list using the Move to List button.
2. After selecting more than one item in the scroll list, move the contents to the array field by clicking the Move to Array button.
3. Check that the IsSelected ToggleFields in the array are properly set.

NestedWindow

Description NestedWindow illustrates multiple tasks, nested windows, event handlers, and input validation. A launch window allows you to start two windows. Each of these windows nests another window. Field and cross-field input validation techniques are demonstrated.

Pex Files display/nestwin.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use NestedWindow**

1. Click on the Purchase Art button. When the window comes up, move it aside.
2. On the first window, click on the Sell Art button. Notice that the same window is nested in both. They should come up with different initial values in the Type of Art field. This data was passed to the nested window's event handler as a parameter.

3. Experiment with entering different values in the Type of Art field and the Year field. There is input validation on the Type of Art field. There is cross-field validation between the Type of Art field and the Year field. Enter 'Performance' in the Type of Art field, and '1900' in the Year field, then click the OK button.

NomadicOrderClient

Description This application starts a standalone client. In standalone mode, you can enter orders and search for entered orders on a local database based on the customer ID. You can also search the server database for customer orders and reconcile the local database with the server database. When you search for entered orders in the server database, the client partition connects to the name server and the service object. The client stays connected until you close the Find Order window. When you reconcile with the server (upload new orders and download a new copy of the server database to the client), the client connects to the service object, then disconnects when it is done.

Pex Files FORTE_ROOT/install/examples/frame/nomad.pex

Mode Distributed

Special Requirements See the information about testing nomadic applications in *iPlanet UDS Programming Guide*. You need to deploy this application to actually connect to and disconnect from the environment.

➤ **To use NomadicOrderClient**

1. Start the application using the `-fnomad` flag, for example:

   ```
   ftexec -fi c:\forte\userapp\nomadico\cl0\nomadi0 -fnomad
   ```

2. Enter one or more orders, remembering the customer IDs.

3. Perform the following actions, which cause the client to connect to the environment, then disconnect:

   a. Reconcile the client with the server.

   b. Search for orders on the server.

OLEBankEV

Description OLEBankEV shows how to create an environment-visible service object that provides wrapper methods to the methods on other iPlanet UDS service objects. This example then provides a Visual Basic client that shows how an OLE client application could interact with the environment-visible service object to use the services provided by the BankServices.BankServer service object.

- olebanev.pex contains the OLEBankEV project, which defines a class named BankServiceOLEInterface, which defines wrapper methods that in turn invoke methods on the BankServices.BankServer service object. This project also defines an environment-visible service object called BankServerOLE.
- All the other files are part of the Visual Basic OLE client.

Pex Files frame/banksvc.pex, extsys/ole/server/olebanev.pex.

Mode Distributed only.

Special Requirements This example runs only on a Windows NT server node. You need to have Microsoft Visual Basic installed on the Windows NT server node and a C++ compiler. You should have autocompile set up.

The Visual Basic clients were written to use Visual Basic Version 4.0. If you are using later versions of Visual Basic, you might need to upgrade the provided Visual Basic components, adjust the following instructions.

➤ **To use OLEBankEV**

1. Import the .pex files, listed above.
2. Configure the OLEBankEV project as a server application.
3. Mark the BankServerOLE service object as an OLE server, as described in *Integrating with External Systems*.
4. Remove the server partition from all nodes that are not running Windows NT.
5. Make a distribution using autocompile and autoinstall.
6. Start the iPlanet UDS server partition, as described in *Integrating with External Systems*.
7. Using Visual Basic, open the `OLEBankEV.vbp` file.
8. Make sure that the OLEBankEV project can find the `BankEV.frm` file by using the Visual Basic File > Add File command.

9. In Visual Basic, use the TOOL > References command to tell the OLE client application the location of the iPlanet UDS service object .tlb file, which is in the `FORTE_ROOT\userapp\olebanke\cl0\` directory.

10. Run the Visual Basic OLE client example. Valid account numbers are 1000, 2000, and 3000.

OLEBankUV

Description OLEBankUV shows how to create a user-visible service object that provides wrapper methods to the methods on other iPlanet UDS service objects. This example then provides a Visual Basic client that shows how an OLE client application could interact with the user-visible service object to use the services provided by the BankServices.BankServer service object.

- olebanuv.pex contains the OLEBankUV project, which defines a class named BankServiceOLEInterface, which defines wrapper methods that in turn invoke methods on the BankServices.BankServer service object. This project also defines a user-visible service object called BankServerOLE.
- All the other files are part of the Visual Basic OLE client.

Pex Files frame/banksvc.pex, extsys/ole/server/olebanuv.pex.

Mode Distributed only.

Special Requirements This example runs only on a node running Windows 95 and Windows NT. You need to have Microsoft Visual Basic installed on the Windows NT or WIndows 95 node and a C++ compiler. You should have autocompile set up.

The Visual Basic clients were written to use Visual Basic Version 4.0. If you are using later versions of Visual Basic, you might need to upgrade the provided Visual Basic components, adjust the following instructions.

➤ **To use OLEBankUV**

1. Import the .pex files, listed above.
2. Configure the OLEBankUV project as a client application, with the BankServerOLE service object in the client partition.
3. Mark the BankServerOLE service object as an OLE server, as described in *Integrating with External Systems*.

4. Remove the client partition containing the BankServerOLE service object from all nodes that are not running Windows 95 or Windows NT.
5. Make a distribution using autocompile and autoinstall.
6. Start the iPlanet UDS client partition, as described in *Integrating with External Systems*.
7. Using Visual Basic, open the OLEBankUV.vbp file.
8. Make sure that the OLEBankUV project can find the `BankUV.frm` file by using the Visual Basic File > Add File command.
9. In Visual Basic, use the TOOL > References command to tell the OLE client application the location of the iPlanet UDS service object .tlb file, which is in the `FORTE_ROOT\userapp\olebanku\cl0\` directory
10. Run the Visual Basic OLE client example. Valid account numbers are 1000, 2000, and 3000.

OLESample

Description OLESample uses OLEField, `Olegen`, and iPlanet UDS's implementation of OLE Automation. It uses a Microsoft Chart application (part of Microsoft Graph5.0). The chart is embedded in an OLEField. `Olegen` has been run to create a Graph project. OLE Automation methods are used to access and manipulate objects in the chart.

Pex Files extsys/ole/msgraph.pex, extsys/ole/olesam.pex.

Mode Standalone or Distributed.

Special Requirements MSWindows NT environment, MSGraph5.0.

➤ **To use OLESample**

1. The OLESample .pex files are not imported automatically by the tstapps.fsc script, so you must first import them in the order given above. `msgraph.pex` was generated by invoking `Olegen`. The following command line was used:

```
-- If you run this, use paths appropriate for your environment.
olegen -it c:\windows\msapps\msgraph5\gren50.olb
-of c:\examples\extsys\ole\msgraph.pex
```

You can generate your own msgraph.pex, to see olegen in operation, or you can use the `msgraph.pex` provided in the examples directory.

2. Start the application. Click on the New Graph button. The embedded OLE field will activate a generic Microsoft Graph Chart application.
3. Click on the iPlanet UDS window to deactivate the field.
4. Double-click in the OLE chart field to activate it. Choose Insert and Titles from the Microsoft Graph Chart menu. Choose Chart Title and click the OK button. Change the title if you wish.
5. Click in the iPlanet UDS window to deactivate Microsoft Graph Chart.
6. Click the Rotate Chart button as many times as you like.
7. Click the Change Title button and provide a title of your choice.
8. When you exit the example, the graph with your changes will be saved in the file `olesam.out` in `FORTE_ROOT/tmp`.
9. Start the application again. This time the Load Saved Graph button will be activated. Click it. The chart it loads will reflect the changes you just made after creating a new chart. You can change the title and rotate the graph again. These changes will be saved when you exit the application.

OLMBanking

Description OLMBanking shows how to use methods of the Framework `ObjectLocationmgr` class to register named objects with the Name Service, then bind to a particular named object and invoke methods on it. This application uses the values of the `-country` command-line argument to determine which named object to bind to.

This application lets the user select either the US or Canada, then select a bank branch. Then, the user can deposit or withdraw money from the bank accounts at the selected branch.

Pex Files frame/olmbank.pex.

Mode Distributed only.

Special Requirements None

➤ **To use OLMBanking**

1. On the command line, specify the country whose branches are to be available, using the `-country` command-line argument. Valid values are `US` (the default) or `Canada`.

   If you test run this example in the Window Workshops, then you cannot specify the `-country` command-line argument, and you will always see the US bank branches.

2. Select the branch, then the account at that branch.

3. Deposit and withdraw money from the account.

OutboundExternalConnection

Description OutboundExternalConnection illustrates how to use the `ExternalConnection` class to initiate a connection. The C program extcon waits for a new connection. OutboundExternalConnection will initiate the connection extcon is waiting for. Once the connection is established, data is read and written. For the read, the iPlanet UDS program makes sure the anticipated number of bytes have been received. For the write, the iPlanet UDS program uses the `UseData` method on MemoryStream to improve efficiency.

Pex Files outbound.pex.

Mode Distributed.

Special Requirements C compiler; C portion of this example will run on NT and Unix platforms; it will not run on Windows or VMS.

➤ **To use OutboundExternalConnection**

1. Decide which platform you want to run the C program on, and which platform you want to run the iPlanet UDS program on. Compile the C program extcon.c into the executable extcon on the desired platform.

   On most Unix systems, simply use the following command:

   ```
   cc extcon.c -o extcon
   ```

   This will work on the following platforms:

   - AlphaOSF
   - RS6000

- ❍ Solaris
- ❍ Data General

On Sequent, use the following command:

```
cc extcon.c -o extcon -lsocket -linet -lnsl
```

On HP, use the following command:

```
cc +Z extcon.c -o extcon
```

On NT, if you use Visual C to compile extcon.c, make sure to include `wsock32.lib` with your standard Object/Library modules. Also, make sure the application is defined as a console application, not a windows application.

2. Both the iPlanet UDS program and the C program will use a default port number for the listener, unless you supply it as an environment variable. The default port number is 6868. If you need to use another port number, set the environment variable FORTE_EP_REG_PORT_2 to the desired port number in both the environment where you will run the iPlanet UDS program and the environment where you will run the C program.

3. If you want to establish an external connection between the iPlanet UDS program and the C program running on the same machine, you do not need to set an environment variable for the node name. If you want to connect between different machines, you will need to set an environment variable. Set the environment variable FORTE_EP_NODENAME_2 in the environment where you are running the iPlanet UDS program. Set it to the name of the machine where the C program is running.

4. On the machine where you compiled extcon, run it with the w command line option, so that it will wait for a connection:

   ```
   extcon w
   ```

   Extcon will time out after three minutes. If you need more than three minutes to start the iPlanet UDS part of this example, edit extcon.c. Increase the value of DEFAULT_REG_UPTIME and recompile extcon.c

5. Use the file `outbound.scr` to supply the necessary commands to fscript. `Outbound.scr` will import the pex file `outbound.pex`, find the project, run it, and remove the project after the run is complete. Outbound.pex must be in the same directory as `outbound.scr`. Use fscript's `-i` flag to input `outbound.scr` to fscript:

   ```
   fscript -i outbound.scr
   ```

6. Observe the output of both processes. On the iPlanet UDS side, you should see the following results:

```
Attempting to make a connection on port 6868.
Attempting to make a connection on canis.
OutboundConnection: server read got back 14 bytes
Data received.
Outbound connection: close done
```

From the C program, you should see the following lines:

```
Waiting to connect on port 6868.
QString = Tire<EOW1>65psi<EOW2>Inflated<EOW3>
```

PencilPlay

Description PencilPlay uses the PictureButton, Rectangle, Ellipse, Line and Point classes. It is a simple draw program that lets you dynamically create graphics.

Pex Files display/pencil.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use PencilPlay**

1. Use any of the standard palette list tools and the commands in the two menus. In addition you can move a selected item by positioning the cursor over it and pressing the mouse button.

2. To copy a selected item choose the Copy command, then select the Paste command. Drag the original item to a new location and a copy will appear when you release the mouse button.

PrintSample

Description PrintSample shows how to use the printing classes. There are six subsections to the application; each one shows how you might solve different printing problems.

The first option, SimplePrint, uses no TOOL code. It relies on the Print and PrintSetup menu commands to print the current UserWindow.

The second option, SimpleClone, clones the current UserWindow and makes some changes to the cloned window's data. It also shows how to force changes to the DefaultPrintOptions object.

The third option, Expand&Tile, prints the current UserWindow, expanding all the fields so that their hidden data as well as their displayed data will be printed. When all the fields are expanded, page tiling will occur.

The fourth option, Report, uses a template window with header and footer information to print the data from the current window. The data from a large array is printed on a multi-page report. This part of the example shows how to create page and line breaks, while using DrawMultiLineText.

The fifth option, EmptyPage, uses the EmptyPage class as the WorkingPage.

The sixth option, ExpandText, prints a TextData object that contains long lines. DrawMultiLineText is used with different height and width policies on the PrintDocument.

Pex Files display/printsam.pex.

Mode Standalone or Distributed.

Special Requirements Printer connection. The files `artist.dat` and `painting.dat` must be located in `FORTE_ROOT/install/examples/database`. The file `hr.dat` must be located in `FORTE_ROOT/install/examples/display`.

➤ **To use PrintSample**

1. From the main window, choose the buttons that sound of interest. Each subsection allows you to print and to set up print options. Some have more refined options.
2. Report lets you print the array without the report template, using standard array field expansion and vertical tiling, for comparison. It also offers two styles for the report: one where it is measured in mils, another where it is measured in columns.

3. The ExpandText option lets you see the use of DrawMultiLineText with and without tiling and with various line spacing options. You can choose between natural and explicit height and width policies to observe horizontal and vertical tiling, as well as multi-page printing without tiling. You can also choose various line spacing options.

SimpleOutline

Description SimpleOutline uses the OutlineField class. It is a simple utility for building a document structured as an outline.

Pex Files display/soutline.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use SimpleOutline**

1. Add a few initial entries by clicking on the AddChild button several times.
2. You can then add items before or after an item, delete an item, or add child items by selecting an item and clicking on the appropriate button.

TabFolders

Description TabFolders illustrates how to use the TabFolder class, and how to use Popup Menus. The example allows you to open two windows. One window illustrates basic TabFolder and Popup functionality. The TabFolder widget was constructed in the Window Workshop. The second window illustrates how to create a TabFolder dynamically and how to show Popup Menus dynamically.

Pex Files display/tfolder.pex.

Mode Standalone or Distributed.

Special Requirements None.

➤ **To use TabFolders**

1. The first window presents two pushbutton fields: Basic TabFolder and Dynamic TabFolder. Click Basic TabFolder.
2. A TabFolder with three tab pages appears on the Basic TabFolder window. The second tab page is the top page. Bring other tab pages to the front by clicking on them.
3. On the Bring To Front (Popup Demo) pushbutton, use the key combination required by your client operating system to activate the Popup Menu. You will see a submenu with three menu items: Tab One, Tab Two, and Tab Three. Click any of these menu items, and that tab page will become the top tab page in the tabfolder.
4. Back in the first window, click the Dynamic TabFolder pushbutton. Bring each tab page to the front by clicking on it. Turn the Show Tree Tab Page toggle off. The Tree Tab Page will disappear from the TabFolder. Turn the Show Tree Tab Page toggle back on. The Tree Tab Page will reappear in the TabFolder.
5. The Popup Demo 1 and Popup Demo 2 pushbuttons each have Popup submenus associated with them. The Bring To Front (Dynamic Popup Demo) pushbutton will present a different popup menu depending on whether the Tree Tab Page is visible or not visible. Try it under both conditions.

TimeIt

Description TimeIt illustrates the use of the Timer class. It provides a simple on screen timer clock.

Pex Files frame/timeit.pex.

Mode Standalone or Distributed.

Special Requirements Client and server running on different machines.

➤ **To use TimeIt**

1. Click on the Start button to start the timer, and on the Stop button to stop the timer.

TimeItV1-4

Description The files `timeitv1.pex`, `timeitv2.pex`, `timeitv3.pex`, and `timeitv4.pex` are different versions of the TimeIt application, used to illustrate run-time compatibility issues. `timeitv2.pex` is compatible with `timeitv1.pex`, while `timeitv3.pex` and `timeitv4.pex` are not.

Pex Files frame/timeitv1.pex, frame/timeitv2.pex, frame/timeitv3.pex, frame/timeitv4.pex.

Mode Distributed only.

Special Requirements None.

➤ **To use TimeItV1-4**

1. You will build distributions from the different versions of TimeIt supplied, and install partitions from different distributions on the client and the server. For information on runtime compatibility, see the *iPlanet UDS Programming Guide*.

➤ **To use TimeItV1**

1. Import `timeitv1.pex` into your workspace.
2. Partition it so that the client partition (Timeit_cl0_Client_<node>) id is on the client machine and the server partition (Timeit_cl0_Part1_<node>) is on the server machine.
3. Make a distribution.
4. Install the distribution on both the client and server machine.
5. Run the server partition on the server machine and the client partition on the client machine.

   You'll see the usual TimeIt example application. Pressing the Start button makes the time display every second and beep at the minute. Now exit the client partition and kill the server partition.

➤ **To use TimeItV2**

1. Import `timeitv2.pex` into the same workspace.
2. This time partition it with both partitions on the server machine and install it only on the server machine.

3. Run the server and client partitions on the server machine. There is one new feature: you can optionally beep at the quarter minute as well as the minute.

   The following changes were made to TimeItV1:

   - The class Clock has a new attribute BeepOnQuarterMinute. It has a higher serial number than any previously existing attributes had.
   - Clock has a new event, DoBeep.
   - Clock has a new overloading of the method StartClock.
   - ClockWindow has changed. It includes a new attribute ToBeep generated by the new button.

   None of the existing IDs have changed, and all the rules for adding things have been obeyed. The partitions from TimeItV1 and TimeItV2 should be compatible. To test this, run the client partition from the client machine, which is still the version from TimeItV1. Note that it works correctly. It cannot access the new feature of beeping on the quarter minute, but the existing features work correctly.

➤ To use TimeItV3

1. Import `timeitv3.pex` into the same workspace.
2. Again, partition it so that both partitions run on the server machine, and install it only on the server.
3. There is one important difference between TimeItV1 and TimeItV3 that will cause an incompatibility: instead of adding a new overloading of `StartClock`, TimeItV3 replaces the old `StartClock`, which had no parameters, with a new one, which has a boolean input parameter. Since the overloading of the method which the older client uses no longer exists, TimeItV3 is incompatible with TimeItV1.
4. You can see this by running the new server partition on the server machine and the old client partition on the client machine. When you click the Start button on the client, which calls the method, you get the error:

   ```
   SYSTEM ERROR: No actual parameter for argument 1 of StartClock
   ```

   This happens because the client partition doesn't pass a parameter, but the server expects one.

➤ **To use TimeItV4**

1. Import `timeitv4.pex` into the same workspace.
2. Again, partition it so that both partitions run on the server machine, and install it only on the server.
3. There is one important difference between TimeItV1 and TimeItV4 that will cause an incompatibility: TimeItV4 adds a parameter to the event `SecondElapsed` which is used to communicate between the client and the server. This makes the new server partition incompatible with the old client partition.

To see this, run the new server partition and the old client partition. Click the Start button. The first time `SecondElapsed` is sent to the client partition, the result is:

```
SYSTEM ERROR: After processing event <xxx> the stack is
incorrectly set.
```

iPlanet UDS's interpreter's stack is incorrect because the event was passed an unexpected number of parameters.

TreeList

Description The TreeList example shows how to coordinate the display of data in TreeView and ListView fields. The TreeView field displays a hierarchy of biological classifications: orders, families, and genera. The ListView field displays detailed information on species within a genus.

Pex Files display/tvlv.pex.

Mode Standalone or Distributed.

Special Requirements none.

➤ **To use TreeListExample**

1. Click on the controls in the TreeView field to expand and collapse the outline. Not all the nodes have children, but by opening them all you will see order, family, and genus names for some Costa Rican birds.
2. When you click a node at the genus level, you will see the species in that genus displayed in the ListView field.

WinDB

Description WinDB uses the GenericDBMS Library classes. It lets you perform standard SQL database access commands. It also illustrates how to send and retrieve Binary Data (BLOBs) from a database, and how to read and write serialized data to a file.

Pex Files database/windb.pex.

Mode Distributed only.

Special Requirements Database connection. The files `artist.dat` and `painting.dat` must be located in `FORTE_ROOT/install/examples/database`.

➤ **To use WinDB**

1. You need an environment that has a node with a resource manager.

   Before running WinDB, open the WinDB Project Workshop, and open the MySession service object. Provide the correct values for your database in the Database Manager, Database Name, User Name, and User Password fields, then click OK. WinDB creates its own tables in the database you specified. The data is provided from files in the `FORTE_ROOT/install/examples/database` directory, called `artist.dat` and `painting.dat`.

2. Start the application. In the radio list, Table is selected by default. Click on the Make Database button. The following painter names are valid:

```
Leonardo da Vinci
Henri Rousseau
Edgar Degas
Jaspar Johns
Pablo Picasso
```

3. Enter a valid painter name and click the Select button. (Note that additional painting data is available for Edgar Degas and Leonardo da Vinci only.)

4. Enter a valid first letter of a painter name followed by% (such as E%), and click the Select button.

5. Enter an invalid painter name and click the Select button.

6. Click on the Drop Database button.

7. Now select Blob from the radio list, click on the Make Database button, and make the same selections as you did for Table. Click the Drop Database button when you're done.

8. Now select File from the radio list, and click on the Make Database button. This will actually create a file to which objects are written. This time, when you select artists, you must type their entire name. Again, click the Drop Database button when you're done.

XRefTime

Description XRefTime is an example of a TOOL C project, along with a TOOL project that calls the TOOL C project. It is useful for seeing how to use the ExternalRef class to free memory associated with iPlanet UDS objects after those objects have been reclaimed by memory management. The example program MathTime shows how to write C projects. The example program DMathTm is the distributed version of MathTime. The example program AllCType is a reference for how to define and call TOOL C methods with parameters of all C data types at assorted levels of indirection.

Pex Files extsys/c/xreftime.pex.

Mode Distributed only.

Special Requirements Access to standard C Runtime Libraries and a C++ Compiler, creation of a working directory, autocompile must be turned on.

➤ **To use XRefTime**

1. Create a working directory where you have read and write permission. Copy the following three files from `FORTE_ROOT/install/examples/extsys/c` to your working directory: `xreftime.pex`, `xreftime.fsc`, `xreftime.c`. Set the environment variable FORTE_EP_WRKDIR to your working directory.

2. The file xreftime.pex contains the C project XRefTimeProject and the TOOL project TestXRefTimeProject. They assume you have access to the standard C runtime libraries. Make sure you know where these are located and what they are called on your system.

3. Edit your copy of `xreftime.pex` so that its ExternalSharedLibs extended property points to the standard C shared library. Search the file for the string '`/usr/shlib/libc`'. Change this string to the correct path and library name for your system.

4. Compile the file mathtime.c into an object file called xreftime.o. Under the directory `FORTE_ROOT/tmp`, create the directory ‘`examples`’, if it isn’t there already. Copy the file xreftime.o to the `FORTE_ROOT/tmp/examples` directory.

5. Before completing this step, make sure autocompile is available on your system. If autocompile is not set up, ask your System Administrator to set it up for you. Now run Fscript and enter the following commands:

```
UsePortable
SetPath %{FORTE_EP_WRKDIR}
Include xreftime.fsc
```

6. The `xreftime.fsc` script will import, distribute, compile, install, and run the XRefTime example. You may want to examine this script to understand the steps involved in linking TOOL code with external C routines.

Memory and Logger Flags

This appendix contains a detailed description of how to use the memory (`-fm`) and logger (`-fl`) flags.

-Fl Flag (iPlanet UDS Logger)

The `-fl` flog allows you to specify logger flags to be used for the command. The logger flags set the file or files used by the LogMgr object for logging messages, and specify the types of messages logged in each file. See LogMgr class in Framework Library online Help for information on how to produce the actual messages.

The `-fl` flag overrides the setting of the FORTE_LOGGER_SETUP environment variable.

`-fl` *file_name*(*file_filter*)[*file_name*(*file_filter*)...]

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes.

The following sections provide information specifying the file name and file filters.

File Name

The log file name is any valid file name where you want to log messages. The special file names "`%stdout`" and "`%stderr`" log the messages to standard output or standard error, respectively.

You can specify several files for logging messages. Multiple logging files are useful, for example, in an application where you want to display general tracing on standard output (`%stdout`), but want detailed tracing logged to a file for later review.

On Windows only, you can use the name "`%stdwin`" to create a simple, scrollable output window for textual output. "`%stdwin`" is particularly useful to specify an alternative file for the output from Fscript or the iPlanet UDS Workshops.

File Filter

Each file name is associated with a *file filter*.

message_type[:*service_type*[:*group_number*[:*level_number*]]]

A description of each file filter option follows.

Message Type Option

The most general filter is message type. The value of message type differentiates messages such as errors, debugging information, or performance data. The message types appear in the following table. Each type is paired with a runtime LogMgr constant that corresponds to the message type when used with the more complex versions of the Put and PutLine methods:

Type	Meaning	Put or PutLine Constant
err	Error Messages	SP_MT_ERROR
sec	Security messages	SP_MT_SECURITY
aud	Audit messages	SP_MT_AUDIT
prf	Performance information	SP_MT_PERFORMANCE
cfg	Configuration modification	SP_MT_CONFIGURATION
trc	Debugging Information	SP_MT_DEBUG
*	All of the above	Any of the above

By using the message type categories, you can print different types of messages to different files. For example, you may want to print trace messages on standard output, error messages on standard output and an error log file, and performance information in a performance log file. The specification for this setup might be the following:

```
%stdout(trc:user err:user) err.log(err:user) perf.log(prf:user)
```

Service Type Option

Within message types there are service types. Service types are the large subdivisions you make within your program and typically map to projects. The service type parameter is optional. If used, the service type value must be between "user1" and "user10". Typically, a service is a large portion of your application, such as inventory control, accounts receivable, or employee administration.

The LogMgr constant that corresponds to the service types "user1" through "user10" is SP_ST_USER1 through SP_ST_USER10. You can use these constants with the advanced version of the `Put` or `PutLine` methods or with the `Test` method. For convenience, you can use the name "user" or the asterisk symbol (*) to specify all user service types. Previous examples used the specification "user" without a trailing digit to indicate all user services.

For example, if you want all tracing to go to standard output, but tracing from service types "user1" and "user3" to be logged in a special file as well, you would use the following specification:

```
%stdout(trc:user) trc1_3.log(trc:user1 trc:user3)
```

Group Number Option

Within a service type there are group numbers. Group numbers are smaller subdivisions you make within a particular service and typically map to a group of related facilities. The optional group number provides further filtering within the service. A group number is between 1 and 63 inclusive.

For example, within a particular service (say, "user3") you may have subdivided the modules into groups (for example, "transactions in progress", "queued work lists", and "problem reports"). Each module is large enough to warrant a group number within the service. "Transactions in progress" may be group number 2, whereas "problem reports" may be the group number 4. The following specification puts performance information from group number 2 into one file and trace information from group number 4 into another file:

```
xactprog.prf(prf:user3:2) probrep.trc(trc:user3:4)
```

The group number you specify in a `Put` method may be a constant that you defined to be equivalent to the numeric literal that you specified in FORTE_LOGGER_SETUP. For example, even though the literal 2 indicates the “transaction in progress” group, your specification to print the related performance information may be the following:

```
task.Part.LogMgr.PutLine(SP_MT_PERFORMANCE,
       SP_ST_USER3,TRANSACT_IN_PROGRESS,1,perfTextData);
```

This code assumes the value of the TOOL constant TRANSACT_IN_PROGRESS is 2.

You can also specify a range of group numbers using the syntax *group#-group#*. In the previous example, if you want trace information from groups 2 through 4 to go to a specific file, you would use the following statement:

```
some_trc.log(trc:user3:2-4)
```

Level Number Option

Within a group there are level numbers that you use to specify particularly detailed levels of information. The greater the level number value, the more detailed the information. The optional level number indicates the detail level of the information printed. Level numbers must be from 1 to 255 inclusive.

As with group numbers, the level number is determined by the application. Typically, developers use level numbers to filter out trace messages. Using the current example, the specification%stdout(trc:user3:2:1) indicates that all level 1 trace data from the “transaction in progress” (group 2) module of the “user3” service should be printed to standard output. Levels greater than 1 do not print. Thus, the following fragment prints only one line:

```
log: LogMgr = task.Part.LogMgr;
-- Printed (level <= 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS, 1,
           ’Browsing account # ’);
   log.PutLine(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             1,acc.Number);
-- Not printed (level > 1)
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             2,acc.Owner);
   log.Put(SP_MT_DEBUG, SP_MT_USER3, TRANSACT_IN_PROGRESS,
             2,acc.LastChangeDate);
```

-Fm Flag (Memory Manager)

The `-fm` flag allows you to control the space used by the iPlanet UDS memory manager.

If you do not set the memory flags, iPlanet UDS uses defaults appropriate for the operating system.

Note that you can change the memory configurations for a running application using the Environment Console and instruments defined on the OperatingSystem agent. See *iPlanet UDS System Management Guide* for information.

`-fm`(*memory_option* {: | =} *number* [, *memory_option* {: | =} *number*])

To make this flag portable across the platforms supported by iPlanet UDS, do not include any spaces in this argument, and do not enclose any part of the argument in single quotes.

For UNIX and VMS, any arguments that contain parentheses must be enclosed by double quotes, as shown in the following example:

```
"-fm(n:4000,x:8000)"
```

In UNIX, if you include spaces in this argument, you need to enclose the values, including the parentheses, in single quotes. You do not need to use single quotes for any other platform. The following table describes the memory options. For options that refer to "pages," a page is 1024 bytes of memory.

Memory Option	Description
c	Specifies when the memory pool should be contracted. The value represents the percentage utilization of the active pages that will trigger a memory pool contraction. Range is 0 to 100. The default value is 80. This option is valid only for Windows 95.

Memory Option	Description
d	Sets the level of debugging information that is provided. The value is interpreted as a bit-mask of enabled options. The default is 0. The options are:
	1—Verify memory before every collect. This checks that all of the memory manager's data structures are correct, that all pages containing user objects are correct, and that all pointers point to something legal.
	2—Verify memory after every collect.
	4—Verify memory before every allocation.
	8—Zero-Fill free memory.
	16—Pattern-Fill free memory.
e	Specifies when the memory pool should be expanded. The value represents the percentage utilization of the active pages that will trigger a memory pool expansion. Range is 0 to 100. The default value is 80.
g	Sets the percentage by which the memory pool is expanded. The default is 10 percent.
i	Incremental unit in pages for memory expansion or contracting. Range is 64 to 1,048,576 (or 4,096 on Windows 3.1). Default is 256.
n	Minimum number of pages managed by the memory manager. The value specifies the absolute minimum number of pages that will be allocated to the memory heap. Range is 1024 to 4194304 (32384 on WIndows 3.1). Must be less than the x memory option. The default value is 1024. See "Setting Maximum and Minimum Size of the Memory Heap" on page 801 for information about how n and x interact.
r	Sets the minimum number of free pages needed to perform a shutdown. Range is 64 to 1,024. The default is 64.
s	Sets the percentage by which the memory pool is contracted. The default is 5 percent.
	This option is valid only for Windows 3.1.
u	Target average memory use. The value specifies the target percentage utilization of the memory heap, calculated as the proportion of allocated pages that are active. Specify this as a percent of currently allocated memory. Legal range is 25 to 95. The default is 85.

Memory Option	Description
x	Maximum number of pages managed by the memory manager. The value specifies the absolute maximum number of pages that can be allocated to the memory heap. Range is 1024 to 4194304 (32384 on Windows 3.1). Must be greater than the n memory option.The default value is 8192. See "Setting Maximum and Minimum Size of the Memory Heap" for information about how n and x interact.

Setting Maximum and Minimum Size of the Memory Heap

To specify the maximum and minimum sizes of the iPlanet UDS memory heap, use the n and x memory options as described in the previous table.

For most operating systems, iPlanet UDS follows these rules to determine the actual maximum and minimum sizes, based on the values specified:

When you specify only the value of n:

- If n is less than 1024, n is set to 1024.
- If n is smaller than the default value of x (8192), then x is 8192.
- If n is larger than the default value of x (8192), then x is also set to n. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify only the value of x:

- If x is larger than the default value of n (1024), then n is 1024.
- If x is smaller than the default value of n (1024), then n is also set to x. The values of the maximum and minimum memory heap sizes in this case are equal.

When you specify both the n and x values:

- x is set to the larger value specified, whether by x or n. The value of n is always the value specified.

Index