NSTableDataSource

(informal protocol)

Category Of: NSObject

Declared In: AppKit/NSTableView.h

Category Description

The NSTableDataSource category declares the methods that an NSTableView uses to access the contents of its data source object. It determines how many rows to display by sending a **numberOfRowsInTableView:** message, and accesses individual values with the **tableView:objectValueForTableColumn:row:** and **tableView:setObjectValue:forTableColumn:row:** methods. A data source must implement the first two methods to work with an NSTableView, but if it doesn't implement the third the NSTableView simply provides read-only access to its contents.

The NSTableView treats objects provided by its data source as values to be displayed in NSCell objects. If these objects aren't of common value classes—such as NSString, NSNumber, and so on—you'll need to create a custom NSFormatter to display them. See the NSFormatter class specification for more information.

Suppose that an NSTableView's column identifiers are set up as NSStrings containing the names of attributes for the column, such as "Last Name", "City", and so on, and that the data source stores its records as an NSMutableArray, called **records**, of NSMutableDictionary objects using those names as keys. Here's a small example, given as an ASCII property list:

```
(
  { "Last Name" = Anderson;
    "First Name" = James;
    Abode = apartment;
    City = "San Francisco";
  },
  { "Last Name" = Beresford;
    "First Name" = Keith;
    Abode = apartment;
    City = "Redwood City";
  }
)
```

With such a record structure, this implementation of **tableView:objectValueForTableColumn:row:** suffices to retrieve values for the NSTableView:

```
- (id)tableView:(NSTableView *)aTableView
    objectValueForTableColumn:(NSTableColumn *)aTableColumn
    row:(int)rowIndex
{
    id theRecord, theValue;

    NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
    theRecord = [records objectAtIndex:rowIndex];
    theValue = [theRecord objectForKey:[aTableColumn identifier]];
    return theValue;
}
```

Here's the corresponding method for setting values:

```
- (void)tableView:(NSTableView *)aTableView
    setObjectValue:anObject
    forTableColumn:(NSTableColumn *)aTableColumn
    row:(int)rowIndex
{
    id theRecord;

    NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
    theRecord = [records objectAtIndex:rowIndex];
    [theRecord setObject:anObject forKey:[aTableColumn identifier]];
    return;
}
```

Finally, **numberOfRowsInTableView:** simply returns the count of the NSArray:

```
- (int)numberOfRowsInTableView:(NSTableView *)aTableView
{
    return [records count];
}
```

In each case, the NSTableView that sends the message is provided as *aTableView.* A data source object that manages several sets of data can choose the appropriate set based on which NSTableView sends the message.

Instance Methods

numberOfRowsInTableView:

– (int)**numberOfRowsInTableView:**(NSTableView *)*aTableView*

Returns the number of records managed for *aTableView* by the data source object. An NSTableView uses this method to determine how many rows it should create and display.

tableView:objectValueForTableColumn:row:

– (id)**tableView:**(NSTableView *)*aTableView*
objectValueForTableColumn:(NSTableColumn *)*aTableColumn*
row:(int)*rowIndex*

Returns an attribute value for the record in *aTableView* at *rowIndex*. *aTableColumn* contains the identifier for the attribute, which you get by using NSTableColumn's **identifier** method. For example, if *aTableColumn* stands for the city that an employee lives in and *rowIndex* specifies the record for an employee who lives in Portland, this method returns an object with a string value of "Portland". See the category description for an example.

tableView:setObjectValue:forTableColumn:row:

– (void)**tableView:**(NSTableView *)*aTableView*
setObjectValue:(id)*anObject*
forTableColumn:(NSTableColumn *)*aTableColumn*
row:(int)*rowIndex*

Sets an attribute value for the record in *aTableView* at *rowIndex*. *anObject* is the new value, and *aTableColumn* contains the identifier for the attribute, which you get by using NSTableColumn's **identifier** method. See the category description for an example.