

Manual: FDO91 Manual

Chapter 1: Introduction describes the basics of form development, the FDO91 language and its structure, and the contents of this manual.

Last updated: February 1998

# CHAPTER 1

## Introduction

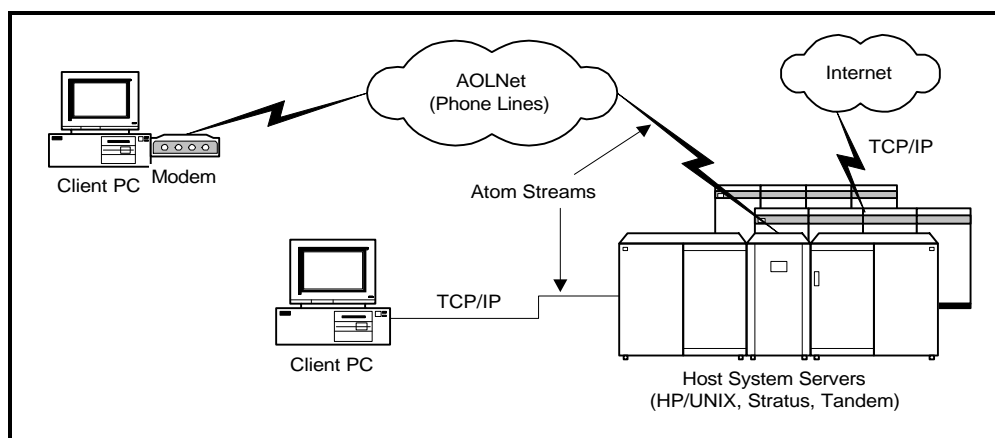
---

FDO91 is a programming language designed to control all client computer functionality for the Windows™ and for Macintosh® platforms of the online service. You can enhance existing client functionality by using this language.

FDO91 evolved from the FDO88 language, which is the original online service forms display language developed for the Apple® II platform. The acronym FDO stands for "Form Display Operation."

FDO91 atoms are the commands that form the basis of the FDO91 language. An atom stream is created when a number of atoms are put together in a sequence that performs a specific function. Atom streams can be stored and sent from the host system servers or from the local client database. Atom streams sent from a host system server or from the client are assembled into data packets.

Figure 1.1 shows the host system server to client PC connection types, which are a phone line network and a TCP/IP network that carry the atom streams:



*Figure 1.1 – Atom Streams in the Host to Client PC Connections*

## Form Development

The major use of the FDO91 language is for the development of forms, or windows, for the online service. This section presents an atom stream that builds a simple form and information about how a form is developed.

### The Form

An atom stream, which consists of a series of FDO91 atoms, determines the content and design of a form. Figure 1.2 shows a sample form (dialog box) as an online member sees it, followed by the FDO91 atom stream that produced the form, and a brief description of each atom's function:


	
Atom Stream	Atom Function
atom\$uni_start_stream	Starts the atom stream.
atom\$man_start_object <ind_group, "Exit Gateway">	Creates a window.
atom\$mat_orientation <vff>	Makes the window vertically oriented.
atom\$mat_position <5>	Centers the window on the screen.
atom\$man_start_object <ornament, "Are you sure you want to sign off?">	Creates a text field with specified text.
atom\$mat_size 14x 02x	Sets the size of the text field.
atom\$mat_font_id <1>	Sets the text field's font ID.
atom\$mat_font_size <14>	Sets the text field's font size.
atom\$mat_font_style <1>	Sets the text field's font style.
atom\$man_start_sibling <org_group>	Creates an organizational group.
atom\$mat_orientation <hef>	Makes the group horizontally oriented.
atom\$man_start_object <trigger, "Yes">	Creates a button with a "Yes" label.
atom\$mat_bool_default <yes>	Sets the previous object as the default.
atom\$man_start_sibling <trigger, "No">	Creates a button with a "No" label.
atom\$man_start_sibling <trigger, "Exit Application">	Creates a button with an "Exit Application" label.
atom\$man_end_object	Terminates the organizational group.
atom\$man_update_display	Sends the objects to the screen.
atom\$uni_wait_off	Turns off the wait cursor.
atom\$uni_end_stream	Specifies the end of the atom stream.

Figure 1.2 – FDO91 Form Example

A typical atom stream for a form consists of:

- A command that creates a window (form)
- One or more commands that set attributes of the form
- Several commands that create objects (such as buttons, list boxes, view fields) or actions on a form
- One or more commands that set attributes for the objects on a form

All forms have a form ID called a global ID (GID). Identifiers 32-0199 and 40-2000 are examples of GIDs for forms. Typically, GIDs that begin with 32 provide the basic core of a form (canned-form functionality) and often they are client-based forms. GIDs that begin with 40 through 43 typically provide more online content for information providers. Note that forms produced with the **VP Designer** tool begin with GIDs of 43.

All streams have a unique stream identifier called a stream ID (SID) so that a number of concurrent form activities can be tracked and processed between the client and host. Stream IDs can be generated by the host or client.

## Objects, IDs, and Their Context

Objects are an integral part of form design. Within the atom streams, the objects are defined in a hierarchical order (tree structure) that begins with a window (root) object. Window objects can have object groups that contain subordinate (children) objects, which together provide the form features and content. Figure 1.3 shows some common types of form display objects such as ornamental pictures and text, list boxes, buttons, and input fields. For more specific information about object names, see Chapter 3, "Attribute Manager Protocol," which defines the available form display objects.

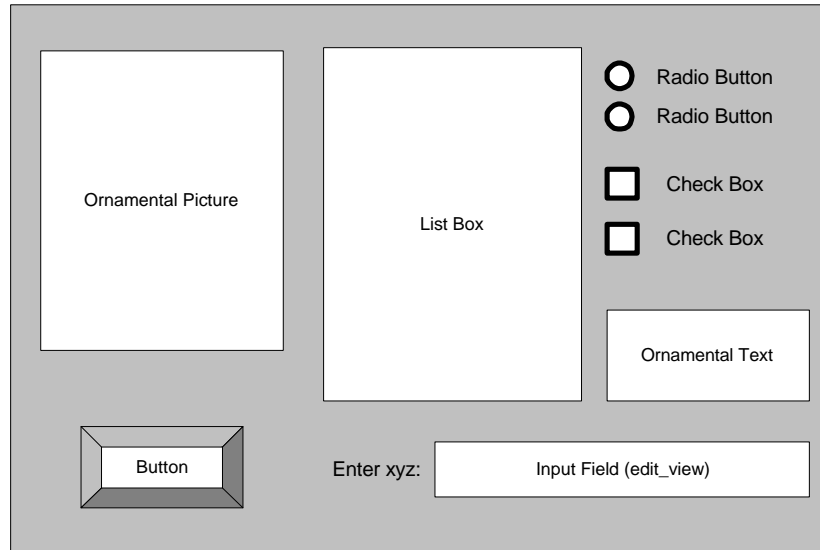


Figure 1.3 – Objects on Forms

When you modify objects on a form, you must establish an operating focus (context) in your atom streams to a specific object so that the form compiler tool can synchronize its operation to the correct object on your form. In your code stream, set the context to the specific object, and then end that context when the defined operations on the object are complete (before you code the next object in the sequence). All objects on a form can be identified with a unique relative ID. In the following example, there are two button (trigger) objects in the form being defined as relative ID 1 and 2:

```
atom$man_start_object <trigger>
⇨ atom$mat_relative_tag <1>
atom$man_end_object
atom$man_start_object <trigger>
⇨ atom$mat_relative_tag <2>
atom$man_end_object
```

In the following example, the context is set to relative ID 14 to add mail folder items to this object. Then the context is changed to relative ID 13 to make object 13 nonselectable:

```
atom$uni_start_stream
⇨ atom$man_set_context_relative <14>
atom$man_build_savemail_menu
atom$man_end_context
⇨ atom$man_set_context_relative <13>
atom$mat_bool_disabled <yes>
atom$man_end_context
atom$man_update_display
atom$uni_end_stream
```

Art (graphic) and binary record objects are also identified with a GID. Identifiers 1-0-01234 and 20-0-00335 are examples of GIDs for art and binary

records. Typically, GIDs that begin with 1 are art-based objects, and GIDs that begin with 20 are records stored in a database. The following example assigns an art GID of 1-0-02201 to a button (trigger) object:

```
atom$mat_start_object <trigger>
atom$mat_relative_tag <1>
↔ atom$mat_art_id <1-0-02201>
atom$mat_precise_x <370>
atom$mat_precise_y <105>
atom$mat_art_animation_rate <250>
atom$mat_art_animation_seq <1 5 0 1>
atom$mat_repeat_animation <yes>
.
.
.
atom$mat_end_object
```

Objects on a form can be placed in one of two layout forms: precise positioning and relative grouping.

Precise positioning requires exact pixel placement on both the x and y axes. The following example defines the placement of all objects on the form as precise positioned:

```
atom$man_start_object <ind_group, "Roll Over Beethoven">
atom$mat_bool_background_pic <yes>
atom$mat_art_id <1-0-2196>
↔ atom$mat_bool_precise <yes>
atom$mat_precise_width <500>
atom$mat_precise_height <333>
atom$man_start_object <ornament, "Music Forums">
atom$mat_precise_x <9>
atom$mat_precise_y <10>
atom$mat_precise_width <204>
atom$mat_precise_height <18>
.
.
.
atom$man_end_object
```

Relative grouping requires horizontal and vertical justification (hff and vff format) for object placement. The following example evenly spaces the placement of trigger objects along the horizontal axis (hee):

```
atom$man_start_object <ind_group>
atom$man_start_object <org_group>
↔ atom$mat_orientation <hee>
atom$man_start_object <trigger>
atom$mat_title <"Button 1">
atom$man_end_object
atom$man_start_object <trigger>
atom$mat_title <"Button 2">
atom$man_end_object
atom$man_start_object <trigger>
```

```
atom$mat_title <"Button 3">
atom$man_end_object
```

You can combine the two groupings within a form; however, you cannot combine them within the context of the group. Once an item in the object tree is made precise, all objects subordinate to it must be precise.

## Form Compilers

There are two form compiler tools that you can use to create and edit atom streams for form development: **VP Designer** and **form\_edit**.

**VP Designer** is a Graphical User Interface (GUI) tool that lets you create, modify, and browse forms while still online. **form\_edit** is a Stratus-based tool that lets you create and modify forms offline.

After you create the form with either tool, you compile it and then display it on a client platform. The tools also let you decompile an existing form so you can view both its object tree structure and its raw data structure, which is the atom stream.

When you use either tool, forms are constructed in a general tree structure, which begins with a root (the window), which splits into branches (groups) that hold the leaves (the objects the member sees on the form).

When you initially execute **form\_edit**, the main screen appears with a window object already created. You can add objects to the object tree structure by making menu selections.

When you use **form\_edit** to create or edit a tree structure, you view the form or object definitions in Pre-stream, In-stream, and Post-stream entry fields. Each object in the tree must be relative to the three entry fields. The Pre-stream is used for the typical initial code to prepare or define global IDs before the form is transferred to the client. In-stream code is where the actual form construction takes place. Post-stream is where the maintenance, upkeep, and cleanup tasks are performed.

For more information about the use of **VP Designer**, see Chapter 4 of the America Online manual *Remote Managed Gateways*.

For more information about the use of **form\_edit**, see the America Online manual *Building an Online Service*.

## FDO91 Features

The main design features of FDO91 include:

- Support of multiple, concurrent atom streams
- Use of a single, centralized command structure (atom\$) that drives all client functionality
- Use of abstract object classes to allow for platform independence
- Use of a true object tree model for complex and flexible object relationships
- Use of consistent operations across object classes
- Ability for action criteria to include custom behavior that can be programmer-defined
- Support of numeric, string, and raw data variables
- Support of branching and looping
- Ability for atoms to return values and to substitute a return value as a command parameter
- Ability to create, delete, change, and query objects at any time
- Ability to perform data extraction

## FDO91 Structure

FDO91 atoms are sent back and forth to enable interaction between the client and the host. Atoms are grouped into protocols, which are sets of commands that perform related logical functions. The atoms and protocols are described in the following sections.

### Atoms

Atoms follow a format that consists of the string "atom\$" followed by the atom name and its arguments. Atoms can also return values to which other atoms



can respond. Although most atoms have arguments and return values, some do not.

An example of the syntax of an atom is as follows:

```
atom$var_number_set <register, data>
```

where

atom\$var\_number\_set is the atom command.

<register> is a required first argument.

<data> is a required second argument.

For more information about the conventions used for command syntax, see "Conventions in This Manual" on page 1-15.

## Data Types and Lengths of Arguments

An argument of an atom is specified in this manual as numeric, data (raw or byte), string, or boolean. Numeric values are decimal whole numbers, which are 0 or greater. Data (raw or byte) values are hexadecimal numbers, which are identified with an x following each byte value (for example, 0C5E would be coded 0Cx 5Ex). String values are a series of alphanumeric characters bracketed with quotes, for example, "Today's stock quotes." Boolean values are a two-state notation and are distinguished with a yes or a no value.

Argument data lengths are one byte (8 bits) or greater. One to four bytes are typical argument lengths. The following data length notations are used in this manual:

byte — 8 bits

word — 16 bits (2 bytes)

dword — 32 bits (4 bytes)

## Protocols

Each FDO91 protocol provides specific functionality for the online service. For example, the Chat protocol provides the chat room functionality of the online service. Each protocol has a unique 2-byte protocol ID that is used to

reference the specific atom handler (for America Online for Macintosh) or client tool (for America Online for Windows).

The following table lists the FDO91 protocols, in alphabetical order, with the protocol ID, atom prefix, description, and the chapter in which the protocol is defined:

*Table 1.1 – FDO91 Protocols*

<b>FDO91 Protocol</b>	<b>Proto ID</b>	<b>Atom Prefix</b>	<b>Description</b>	<b>Chap No.</b>
Action	2	atom\$act	Associates actions with objects	2
ActiveX	54	atom\$activex	Implements forms with ActiveX™ controls	37
Address	39	atom\$ad	Maintains address list	9
ARTdoc	31	atom\$doc	Opens and plays data streams from the ARTdoc database	38
Attribute Manager	16	atom\$mat	Sets attributes on forms	3
Async	13	atom\$async	Miscellaneous functions	10
Asynchronous Data	66	atom\$adp	Handles large chunks of data transfers	42
Buffer	4	atom\$buf	Handles data packets	4
Chart	23	atom\$chart	Provides charting functions	12
Chat	11	atom\$chat	Provides chat functions	11
Code Manager	10	atom\$cm	Handles code management	13
Comm. Control Language	34	atom\$ccl	Handles CCL	14
Conditional	15	atom\$if	Handles conditional operations	5
Data Manager	3	atom\$de	Handles data extraction	15
Database Manager	5	atom\$idb	Maintains online database	16
Device Independent Connectivity Engine	57	atom\$dice	Handles client computer connectivity	34
Display Manager	1	atom\$man	Handles forms display	6
External API	26	atom\$exapi	Handles external applications	17
File Manager	8	atom\$fm	Handles file management	18
File Transfer	7	atom\$xfer	Handles file transfer	19

Table 1.1 – FDO91 Protocols (Continued)

<b>FDO91 Protocol</b>	<b>Proto ID</b>	<b>Atom Prefix</b>	<b>Description</b>	<b>Chap No.</b>
Gallery	56	atom\$gallery	Handles thumbnail images and their display	36
Host Forms Server	51	atom\$hfs	Handles host forms	20
Image Manager	22	atom\$image	Sets attributes for graphic displays	21
Image Transfer	21	atom\$imgxfer	Handles image display	22
List Manager	9	atom\$lm	Handles list management	23
Message Interchange	17	atom\$mip	Handles message data transport	24
Multimedia organizer	24	atom\$morg	Maintains personal filing cabinet	25
Multimedia Interface	20	atom\$mmi	Handles multimedia files and display	26
Pictalk	29	atom\$pictalk	Handles slideshow data streams	39
P3	35	atom\$p3	Handles P3 communication	28
Progressive Rendering	27	atom\$dod	Handles progressive rendering	27
Radio	28	atom\$radio	Captures and plays radio program streams	40
Rich	25	atom\$rich	Handles enhanced text attributes	29
Shorthand Manager	14	atom\$sm	Replaces lengthy atom streams	30
Spell	61	atom\$spell	Invokes the spell dialog box	41
Tool Manager	42	atom\$mt	Manages client tools	31
Universal	0	atom\$uni	Controls atom streams	7
Variable	12	atom\$var	Associates variables with objects	8
Video	53	atom\$vid	Handles video images and display	35
Visual Rainman	47	atom\$vrn	Manages Rainman forms	32
WWW	48	atom\$www	Handles web browser forms	33

## About This Manual

This manual describes the FDO91 protocols and provides detailed definitions and examples of the atoms within each protocol. The manual is organized into two volumes:

- Volume 1 contains the general purpose protocols responsible for overall stream management functions, such as starting and stopping atom streams, setting action criteria, and specifying object attributes.
- Volume 2 contains a wide variety of miscellaneous protocols that perform specialized functions, such as handling P3 communications, maintaining address lists, and managing web browser forms.

## Who Should Use This Manual

This manual is intended as a reference guide for programmers who want to develop additional functionality for the online service client application. This manual is also intended for online service staff who create and maintain areas, forms, and pathways for the online service.

## Security Issues

This manual contains confidential material and is intended for use only by America Online employees. As the owner of this manual, it is your responsibility to take the necessary precautions to ensure that it does not leave your possession.

## Chapter Overview

This manual is divided into 2 volumes, 42 chapters, and 2 appendixes, as follows:

### Volume 1

Chapter 1, "Introduction," describes the basics of form development, the FDO91 language and its structure, and the contents of this manual.

Chapter 2, "Action Protocol," defines the action protocol atoms and provides information for associating actions with FDO91 objects.

Chapter 3, "Attribute Manager Protocol," defines the attribute manager protocol atoms and provides information for specifying form attributes.

Chapter 4, "Buffer Protocol," defines the buffer protocol atoms and provides information for handling data packets.

Chapter 5, "Conditional Protocol," defines the conditional protocol atoms and provides information for handling conditional operations.

Chapter 6, "Display Manager Protocol," defines the display manager protocol atoms and provides information for handling forms display.

Chapter 7, "Universal Protocol," defines the universal protocol atoms and provides information for controlling atom streams.

Chapter 8, "Variable Protocol," defines the variable protocol atoms and provides information for associating variables with objects.

Chapter 9, "Address Protocol," defines the address protocol atoms and provides information for maintaining address lists.

Chapter 10, "Async Protocol," defines the async protocol atoms and provides information on miscellaneous FDO91 functions.

## **Volume 2**

Chapter 11, "Chart Protocol," defines chart protocol atoms and provides information for building charts with numeric data from the online service.

Chapter 12, "Chat Protocol," defines chat protocol atoms and provides information for chat functionality.

Chapter 13, "Code Manager Protocol," defines code manager protocol atoms and provides information for handling code management.

Chapter 14, "Communications Control Language (CCL) Protocol," defines CCL protocol atoms and provides information for the CCL.

Chapter 15, "Data Manager Protocol," defines data manager protocol atoms and provides information for handling data extraction.

Chapter 16, "Database Manager Protocol," defines database manager protocol atoms and provides information for maintaining the online database.

Chapter 17, "External API Protocol," defines external API protocol atoms and provides information for external third-party applications.

Chapter 18, "File Manager Protocol," defines file manager protocol atoms and provides information for handling file management.

Chapter 19, "File Transfer Protocol," defines file transfer protocol atoms and provides information for handling file transfer.

Chapter 20, "Host Forms Server Protocol," defines host forms server protocol atoms and provides host forms information.

Chapter 21, "Image Manager Protocol," defines image manager protocol atoms and provides information for handling hotspots and image display.

Chapter 22, "Image Transfer Protocol," defines image transfer protocol atoms and provides information for handling image display.

Chapter 23, "List Manager Protocol," defines list manager protocol atoms and provides information for managing lists such as lists of e-mail messages.

Chapter 24, "Message Interchange Protocol," defines message interchange protocol atoms and provides information for handling message data transport.

Chapter 25, "MORG Protocol," defines the MORG (multimedia organizer) protocol atoms and provides information for maintaining personal filing cabinets.

Chapter 26, "Multimedia Interface (MMI) Protocol," defines the multimedia interface protocol atoms and provides information for handling and playing multimedia files.

Chapter 27, "P3 Protocol," defines P3 protocol atoms and provides information for handling P3 communication.

Chapter 28, "Progressive Rendering Protocol," defines progressive rendering protocol atoms and provides information for rendering forms progressively.

Chapter 29, "Rich Protocol," defines rich protocol atoms and provides information for rendering text with hypertext markup language (HTML) tags.

Chapter 30, "Shorthand Protocol," defines shorthand protocol atoms and provides information for replacing lengthy atom streams.

Chapter 31, "Tool Manager Protocol," is not included in this release. It will define tool manager protocol atoms and provide information for tool management.

Chapter 32, "Visual Rainman Protocol," is not included in this release. It will define visual Rainman protocol atoms and provide information on the online publishing tools used to maintain areas of the online service.

Chapter 33, "WWW Protocol," defines World Wide Web protocol atoms and provides information for handling Web browser forms.

Chapter 34, "Device Independent Connectivity (DICE) Protocol," defines atoms that initiate and manage all client computer connections.

Chapter 35, "Video (VID) Protocol," defines the atoms that manage and transmit video and still-camera images between members.

Chapter 36, "Gallery Protocol," defines the atoms that manage thumbnail images and arrange them in gallery views.

Chapter 37, "ActiveX Protocol," defines the atoms that manage ActiveX™ controls for 32-bit Windows applications.

Chapter 38, "ARTdoc Protocol," defines the atoms that open and play ARTdoc database record streams.

Chapter 39, "Pictalk Protocol," defines the atoms that open and play multimedia files or slideshow data streams.

Chapter 40, "Radio Protocol," defines the atoms that capture and play radio program streams from live broadcasts and stored files.

Chapter 41, "Spell Protocol," defines the atoms that bring up the spell preferences dialog box.

Chapter 42, "Asynchronous Data Protocol," defines the atoms that manage the asynchronous transfer of large blocks of data between the host and client PCs.

Appendix A, "Examples," provides examples of MIP and List Manager atom streams.

Appendix B, "Internationalization (i18n) Atoms," defines i18n atoms and provides information for localizing data sent from the host system to international clients of the online service.

## Conventions in This Manual

Atom commands in this manual appear in the following format:

```
atom$prot_command <arguments>
```

Note that when you are using **VP Designer**, you do not code the prefix (**atom\$**) with each atom entry.

The following conventions are used throughout this manual in defining atom command syntax:

<arguments>	Angle brackets enclose all arguments whether the argument is required or optional.
<arg1, arg2[, arg3]>	When you code atoms with multiple arguments using <b>VP Designer</b> , commas separate each argument. Note that chapters with a header date before April 1997 do not show this convention, because they were written for <b>form_edit</b> . An argument shown in square brackets indicates that the argument is optional.
<arg1> <arg2> [<arg3>]	When you code atoms with multiple arguments using <b>form_edit</b> , space delimiters and/or angle brackets separate each argument without commas. Note that chapters with a header date before April 1997 still use this convention with angle brackets around each argument. An argument shown in square brackets indicates an optional argument.
(<arg1>   <arg2>)	Parentheses are used around arguments that have an association, and a vertical bar indicates an either/or choice between arguments.
<arg1> <arg2> ...	Trailing periods indicate additional arguments can be specified.

In examples throughout this manual, arguments appear enclosed in angle brackets (<>) as required for **VP Designer**. However, you do not have to use angle brackets around argument values when you are creating or editing an atom stream using **form\_edit**.



An atom's argument can be defined to have a set of numeric and associated label values. For example, `atom$mat_trigger_style` has a list of 10 defined argument values from which you select trigger styles. The values are defined 0 through 9 with associated value labels. For example, either the value 8 or the value label `text_on_picture` can be used to apply text on the art of a trigger object. Note that defined value labels in the text of this manual appear in a monospace (Courier) font as illustrated in this paragraph. This convention does not apply to chapters with a header date before July 1997.

## Providing Feedback

If you have comments, corrections or questions related to this manual, please send e-mail to screen name DevelopHlp. Please include the name of this manual when providing your feedback.