PCT WORLD INTELLECTUAL PROPER
International Bur

INTERNATIONAL APPLICATION PUBLISHED UNDER

WO 9606393A1

(51) International Patent Classification 6 : G06F 9/44, 13/10

A1

(11) International Publication Number: **WO 96/06393**

(43) International Publication Date: 29 February 1996 (29.02.96)

(21) International Application Number: PCT/US95/10763

(22) International Filing Date: 24 August 1995 (24.08.95)

(30) Priority Data:
08/294,974 24 August 1994 (24.08.94) US

(71) Applicant: ARCADA SOFTWARE, INC. [US/US]; Suite 1101, 37 Skyline Drive, Lake Mary, FL 32746 (US).

(72) Inventor: ROSSI, Robert, P.; 1551 Monica Joy Circle, Longwood, FL 32779 (US).

(74) Agents: FLIESLER, Martin, C. et al.; Fliesler, Dubb, Meyer and Lovejoy, Suite 400, Four Embarcadero Center, San Francisco, CA 94111-4156 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published

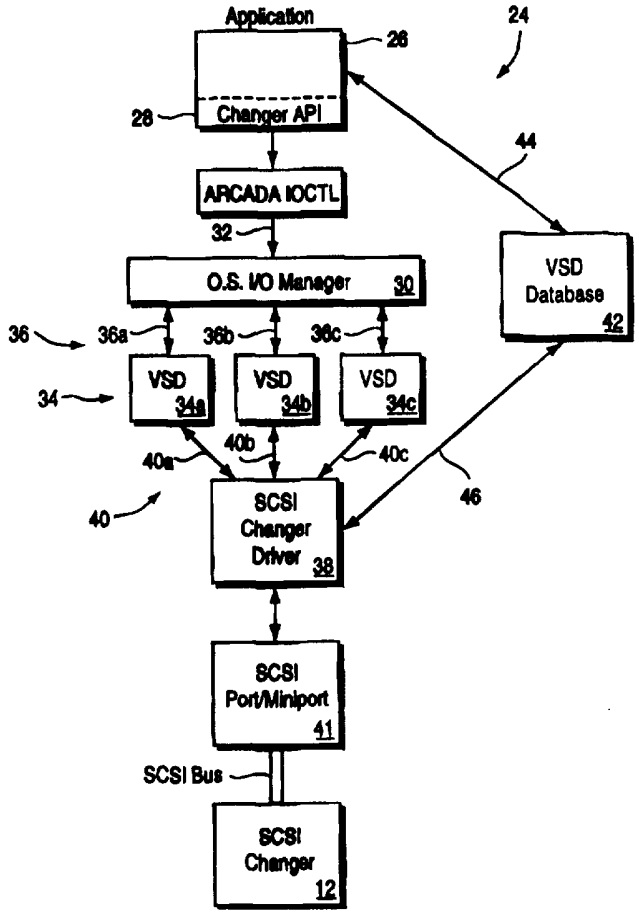
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: APPLICATION PROGRAM INTERFACE (API) FOR A MEDIUM CHANGER

(57) Abstract

An application program interface (API) which interfaces one or more applications programs with one or more media changer devices having a plurality of addressable physical elements, including a plurality of API calls for obtaining information about parameters of any of the devices, and for moving a medium from one of the elements to another, in which one of the calls has a data structure storing, among other information, the features parameters of a given medium changer device and another of the calls has a data structure storing, among other information, the status of the elements of the devices.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

APPLICATION PROGRAM INTERFACE (API) FOR A MEDIUM CHANGER

Inventor(s): Robert P. Rossi

FIELD OF THE INVENTION

This invention relates generally to an applications program interface for interfacing an applications program with a medium changer and, more particularly, to an applications program interface for interfacing any applications program that may control any type of tape medium changer.

BACKGROUND OF THE INVENTION

An applications program or software is, generally, a program that is written to cause a device such as a computer or any of its computer peripherals to perform a specific task. For example, an applications program may cause the computer to perform data management or word processing functions. The applications program also may be written to cause the computer to control and/or obtain status information about a peripheral device such as a medium changer which can move any one or more data storage media into and out of a drive for reading or writing data on the medium.

A medium changer is a device that mechanizes the movement of media to and from a media drive such as a disk or tape drive and between other locations within the range of the medium changer device. Each of the medium changer devices that may be coupled to a computer or be a component of a computer network may have its own characteristics and capabilities; however, they all have one or more instances of at least three common physical elements. These are (1) a storage element which is a location within the medium changer device to hold a unit of media while it is not in some

other type of element, (2) a drive or data transfer element which is a location of the drive that is capable of reading or writing the medium, and (3) a transport element which is a location that a given medium occupies while it is being moved from one element address to another within or about the medium changer device. Optionally, the medium changer device has one or more instances of (4) an import/export element which is a location for inserting and removing media from the medium changer device (which may be referred to as a portal). To the computer, each of these four elements is viewed as a unique addressable element having its own element address.

Standards known as small computer system interface (SCSI) specifications have been published by which manufacturers of products such as disk drives, tape drives, printers and medium changers can design their products. The published SCSI standards include command sets for these products. Most medium changer devices are designed to support the set of commands currently known as SCSI-2 to provide a way to get information about the available elements and a method to move the media from one element to another, and may have some of their own unique additional commands to which they respond. For a more detailed explanation of medium changer devices and the SCSI-2 commands, reference should be made to the standards set forth in Information Technology - SCSI-2 X3T9.2, Project 375R, Revision 10k, dated April 28, 1993, and, in particular, to Chapter 17 relating to medium changer devices 3, which is incorporated by reference herein in its entirety.

Applications program interfaces (API) are available for interfacing an applications program with a given medium changer device. There are two typical

types of conceptual interface boundaries that are utilized for medium changer device APIs. One conceptual interface boundary is a low level I/O Control Code (IOCTL) kernel API. The other is an abstract high level API which hides the IOCTL API. An IOCTL is a mechanism for user-mode (application level code) to communicate with kernel-mode (device driver level code).

An applications program using the low level IOCTL kernel API as its interface boundary between the application and kernel level is going to, by definition, be forced to handle most non-standard medium changer device issues in the application. This is because it is not easy to expand the low level IOCTL kernel API, or add more APIs without extensive work if such a non-standard device needs to be supported but cannot fit within the existing definitions of the IOCTLs. For example, adding a new IOCTL would typically require every device driver to be altered to handle and return gracefully from an IOCTL request, even though only one device may really have any true use for it.

And, handling device specific problems in the application has other disadvantages. The application should only see the changer device as an "object", not as a specific physical product, e.g. one having a particular model number, and made by a specific company, e.g., company XYZ. With the low level IOCTL kernel API, as the code in the application grows so as to handle new devices and their specific problems, the application becomes more prone to bugs and more difficult to maintain.

The more abstract, high level API which hides the IOCTL API has its own problems. A basic problem with this approach is that much is typically hidden below

the API to truly satisfy its definition as an API. When devices that need to be supported are far from standardized, such as SCSI-2 medium changers, setting the conceptual interface at a high abstract level is a risky choice at best. Depending on the definition, there may be no way to have a given device supported by that API and, just as importantly, there may be no way to know that until it is too late, so that the API and/or the application code would have to be hacked to make the non-standardized device work.

For example, assume an API call MountMedia (MEDIA_ID). This API's definition would be that when called the media with media code MEDIA_ID must be located in the changer, and put or mounted into any available drive. However, suppose the media with MEDIA_ID is found in the portal, having been manually placed there by the user. There then may have to be multiple commands issued to the device to satisfy this API call. For example, one command would cause the portal to be closed, another command would cause all the drive elements to be scanned to find an available drive, and yet another command would cause the media to be moved into the drive. And, while responding to these commands, checks would have to be performed to be sure the device has not errored, or is busy servicing another initiator. This example assumed that the particular changer had media code ("bar code") scanning ability to get the media ID, but if it did not then multiple problems arise. Thus, there simply is too much occurring underneath the call MountMedia (MEDIA_ID) to safely set the conceptual interface boundary at the high, abstract level, particularly when trying to support changers which, more often than not, are different from one to the next.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide an API which is flexible and expandable so as to interface with any number of different types of medium changer devices.

It is another object of the present invention to provide an API which interfaces with many different applications programs such that the API allows each such applications program to tailor itself to the characteristics and parameters of any type of medium changer device without the programmer requiring special knowledge of what kind of physical medium changer device is being used. That is, from the applications program viewpoint, it's accessing a changer object via a handle (described below), and performs functions based on what the features parameter and status parameter say the changer object is capable of.

It is yet another object of the present invention to provide an API for a medium changer device that is device centric, meaning the definition of the API, as it pertains to a medium changer object, directly reflects the functions and capabilities of a physical medium changer device. Furthermore, the API is such that it is layered between the high level (abstract) and low level (IOCTL) interface layers described above.

It is still another object of the present invention to provide an API that allows for all functionality defined by the SCSI-2 specifications for medium changer devices, as well as other functions which are not defined by such SCSI-2 specifications, such as media or bar code scanning, and control of the portal.

These and other objects of the present invention are obtained with an applications program interface for interfacing one or more applications programs with one

or more medium changer devices, in which the API has a plurality of API calls for obtaining information about the status parameters of the elements of a given medium changer device and the features parameters of any such device, and for moving the medium supported by the device from one of the elements to another, and wherein one of the calls has a data structure including a plurality of data structure members in which one of the members can store information about the status parameters of the elements of the device, and another one of the calls has a data structure including a plurality of data structure members in which one of the members can store information indicating one or more features that the device supports.

In another aspect, the present invention is a method for interfacing any of a plurality of applications programs with any of a plurality of medium changer devices supporting a set of SCSI commands, using an API, comprising the API receiving from the applications program a pointer to a data structure storing device parameters from which the applications program can configure itself to utilize a given medium changer device, getting information about the vendor of the device, the specific device itself and the features supported by the device, and building an inventory in the database of the information.

In yet another aspect the present invention constitutes a multilayered software architecture for supporting one or more different medium changer devices, including (a) an applications program; (b) an applications program interface layered beneath the applications program; (c) an I/O manager of an operating system layered beneath the applications program interface; (d) a plurality of different vendor specific drivers layered beneath the I/O manager; (e)

a SCSI changer driver layered beneath the plurality of vendor specific drivers for driving the one or more medium changer drivers; and (f) a database used by the applications program to specify supported changers and by the SCSI changer driver to load the correct VSDs.

Consequently, the API of the present invention, among other advantages described throughout this specification, has the advantage of supporting many different medium changer device vendors and status parameters and products, as well as any particular features (up to 32 in the specific embodiment described below) that a particular changer device may have. This allows the applications program to tailor itself to any given medium changer device, thereby making use of the flexibility and expandability of the API.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a perspective view of one example of a medium changer device having instances of the basic four types of elements defined by the SCSI-2 specifications.

FIGURE 2 illustrates the multilayered architecture of a software system in which the API of the present invention is layered.

FIGURES 3A and 3B illustrate data structures of the API of the present invention useable in a computer system.

FIGURE 4 is a flow diagram of the overall procedure of the API of the present invention.

FIGURE 5 is a flow diagram illustrating the procedure in which the API of the present invention initializes a medium changer device.

FIGURE 6 is a flow diagram showing the procedure in which the API of the present invention performs medium changer device functions.

FIGURE 7 is a flow diagram illustrating the procedure in which the API of the present invention performs de-initialization of the medium changer device.

DETAILED DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a representative example of a medium changer device 10 that has a set of four classes or types of discrete physical elements defined by SCSI-2 specifications. In this particular example, the medium changer device 10 is a tape medium changer 12, although the principles of the API of the present invention apply to other media and media changer devices such as disk and CD-ROM. Instances of four elements are illustrated as (1) one or more storage elements 14 storing a plurality of tape cassette media 16, (2) one or more drive elements 18 for reading and writing one of the loadable tape media 16, (3) a transport element 20 which is the location of a given tape medium 16 while the medium is being moved to another element such as the drive element 18, and (4) an import/export element or portal 22 showing the location for inserting and removing media 16 from the tape medium changer 12. Each of these elements is an addressable element having, for example, a unique 16-bit address.

Fig. 2 illustrates the multi-layered software architecture 24 of the present invention. As shown, at the highest layer is a given applications program 26 which has been written to, among many other things, cause the medium changer 12 to perform one or more functions. A medium changer API 28 is layered directly beneath the applications program 26 and interfaces the applications program 26 with a medium changer 12. This API 28 is neither a high level, highly abstract API

such as MountMedia(MEDIA_ID), nor is it defined at the IOCTL level, such that the API may require new application-level code be written for each new medium changer 12 (only one changer 12 shown in Fig. 2) added to a computer or a computer network system (not shown).

The API 28 communicates with the next layer of the architecture 24 which is an I/O manager 30 of a given operating system via a line 32 transferring API IOCTLs. The API 28 can interface with any type of operating system; however, for each O.S., there will typically be unique methods of obtaining a "handle" to a given medium changer device 10, as well as a way to associate which drives belong within the scope of that desired medium changer device 10. A "handle" is a variable that identifies an object; an indirect reference to an operating system resource.

For example, under the O.S. known as Windows NT written and sold by Microsoft Corporation, Redmond, Washington, the CreateFile () API would be called to obtain a handle to a given medium changer device, while the registry would be used to obtain SCSI bus information to aid in making the associations as to what drives belong in the scope of the medium changer, and which drive element maps to which physical device.

The I/O manager 30 interfaces with and selects any of a plurality of vendor specific drivers 34 (VSDs) over respective lines 36. In the particular example shown, there are three VSDs 34a, 34b and 34c each of which communicates with the I/O manager 30 over a respective line 36a, 36b and 36c. Vendor specific processing occurs at the layer of each VSD 34. When an API is executed, the I/O manager 30 uses the changer handle to route the request to a proper VSD 34 and process it.

The next layer of the software architecture 24 is a conventional SCSI-2 compliant changer driver 38 which interfaces with the VSDs 34 over lines 40 shown respectively as 40a, 40b and 40c. The changer driver 38 is the layer that communicates with a SCSI port/miniport driver(s) 41 which interacts with the SCSI bus host adapter to drive the medium changer 12.

SCSI changer driver 38 will see what medium changer or changers 12 are being supported by looking at a VSD Load Table Database 42. The VSD Database 42 is used as the communication hub between the applications program and the SCSI changer driver 38. It contains Vendor and Product ID and the associated VSD name for each device 12 the applications program 26 wishes to support, and for which there exists a VSD for the SCSI changer drive 38 to load. Under the above-mentioned Windows NT operating system, the registry would be used for the VSD Database 42.

As already mentioned, the applications program uses the VSD Database 42 as a method to tell the SCSI changer driver 38 what changers it wants support for. This Database 42, described more fully below, is created by the applications program at installation time on a computer system (not shown), and devices may be added or deleted as desired at a later time. Depending on the O.S., changes to the VSD Database 42, might not be recognized until the drivers are re-loaded, or the operating system re-booted.

Since the SCSI-2 specification may be open for individual interpretation, and it only requires one functional changer command, i.e., Move Medium, most all changer manufacturers are producing a different device from the perspective of the SCSI command interface and the data pages that are returned from the device. Thus, the VSDs 34 are important in the architecture of

this invention. A given VSD 34 should make any digressions from the SCSI-2 specification transparent at the API level, and the VSD 34 should take advantage of any vendor unique commands that make compliance to the rules of the API easier.

The API 28 of the present invention has ten calls 1-10 for any applications program 26 utilizing the medium changer 12. These are defined generally as follows and thereafter in full detail:

API 28 Definitions:

1. **Changer_Get_Element_Status**

   Returns the current status of a given element 14, 18, 20 and 22 in a given medium changer 12.

2. **Changer_Get_Parameters**

   Returns the capabilities of a given medium changer 12.

3. **Changer_Get_Status**

   Returns the status of a given medium changer 12.

4. **Changer_Initialize_Element_Status**

   Resets the status of given elements 14, 18, 20 and 22 in a given medium changer 12, and initializes the changer with the status of those elements.

5. **Changer_Manual_Access**

   Locks/unlocks a given medium changer 12, preventing/allowing the user manual access to media 16.

6. **Changer_Move_Medium**

   Moves media 16 in a given medium changer 12.

7. **Changer_Portal_Operation**

Opens or closes a given portal 22 in a given medium changer 12, enabling/disabling the user from inserting/removing media via the given portal.

8. **Changer_Position_Transport**

Positions a given transport 20 to a given element address.

9. **Changer_Reserve_Release**

Reserves/Releases given elements 14, 18, 20 and 22 in a given medium changer 12 for the current initiator.

10. **Changer_Scan_Media_Code**

Scans the media code on media 16 at given element addresses.

A more detailed description of each call 1-10 will be given with reference to call parameters and data structures which are members or fields. There are various types of parameters which have symbols in the detailed description, as follows:

Symbol	Type
b	boolean
c	character
s	short
dw	double word
h	handle (OS specific)
p	pointer

For example, if a given parameter has the symbol pdw, that is a pointer to a double word.

Each detailed definition of a call lists information about parameters that are input by the applications program 26 to the API 28, followed by a description of the data structure member associated with that call. For example, for the call Changer_Get Element_Status there is a listing:

IN DWORD dwElementAddress;

which is a double word identifying the address of an element 14, 18, 20 or 22 where status is to be reported.

An example of a data structure member for that call Changer_Get_Element_Status is listed as:

dwSourceAddress

which is the source address a medium 16 at dwElementAddress (another parameter) was moved from.

Fig. 3A and 3B illustrate, as examples, two respective data structures whose members are used in connection with two calls. These are the calls Changer Get_Element_Status and Changer_Get_Parameters. As shown in Fig. 3A, and with reference to the detailed definition of this call given below, this data structure has several members or fields which include a source address (SA), primary media code (PMC), alternate media code (AMC), and element status (ES). The field ES is, for example, a 32-bit field in which a "1" in a particular bit position represents a defined condition as specified in the detailed listing below. For example, a "1" in the bit position identified in the detailed listing ELEMENT_ACCESS means that the transport element(s) 20 is currently allowed access to an element at the parameter dwElementAddress (which is the address of the element whose status is to be reported).

As shown in Fig. 3B, the data structure member has several members or fields which include a vendor ID field (VID), a product ID field (PID)... and a features field FTRs. The field VID contains information about the specific vendor of a given tape medium changer 12 as returned by a SCSI-2 command and the field PID contains information about the specific medium changer 12 as returned by a SCSI-2 command. The field FTR is a 32-bit field in which a "1" in a given bit position indicates that the corresponding feature is supported. For example, a "1" in the bit position Move_Drive_To Drive means that the medium changer 12 requires only one changer move medium command to move media between drive elements 18.

Thus, these 32-bit features parameters and status parameters provide for an API 28 that is flexible and expandable by being able to store up to 32 features and status information, respectively.

The detailed definitions of the API calls 1-10, therefore, are as follows:

(1) Changer_Get_Element_Status

DWORD ChangerGetElementStatus(hDevice, dwElement Address, pdwBufferSize, peiElementInformation)

```
IN HANDLE                hDevice;
IN DWORD                 dwElementAddress;
IN PDWORD                pdwBufferSize;
IN PELEMENT_INFORMATION  peiElementInformation;
```

Parameter	Description
hDevice	Handle of the medium changer device on which to obtain an element's status.
dwElementAddress	Address of the element whose status will be reported.
pdwBufferSize	Pointer to a dword that should have the size, in bytes, of the buffer specified by **peiElementInformation**. If the buffer is too small, the dword receives the required size.
peiElementInformation	Pointer to a structure containing the status information of the element at **dwElementAddress**. The following describes the structure members:

Member	Description
dwSourceAddress	The source address the media at **dwElementAddress** was last moved from. Valid only if media is present at **dwElementAddress**, and the **SOURCE_ADDRESS VALID** status is set.
cPrimaryMediaCode[n]	Null-terminated string containing the media code of the media at **dwElementAddress**, accessible via a **ChangerMoveMedium** command without the

	invert bit set. Valid only if the device has media code scanning capabilities, and media is present at **dwElementAddress**.
cAlternateMediaCode[n]	Null-terminated string containing the media code of the media at **dwElementAddress**, accessible via a **ChangerMoveMedium** command with the invert bit set (the other side of the media). Valid only if the device has media code scanning capabilities, inverting media is supported, and media is present at **dwElementAddress**.
dwElementStatus	Contains supported information about the element at **dwElementAddress**. A value of 1 in the proper bit-field represents a given condition. The following describes the element status:

Value (1 in proper bit field)	Description
EXPORT_PORTAL_ACCESS	The changer currently allows exporting media out of the scope of the medium changer device via the portal at **dwElementAddress**. Valid only if **dwElementAddress** is a portal element address.
ELEMENT_ACCESS	The transport element(s) are currently allowed access to the element at **dwElementAddress**. Valid only if **dwElementAddress** is a portal element address, storage element address, or drive element address.
IMPORT_PORTAL_ACCESS	The changer currently allows importing media into the scope of the medium changer device via the portal at **dwElementAddress**. Valid only if **dwElementAddress** is a portal element address.
INVERTING_TRANSPORT	The element at **dwElementAddress** allows media to be inverted with a **ChangerMoveMedium** command. Valid only if **dwElementAddress** is a transport element address.
LAST_PORTAL_ACCESS	Indicates the media at **dwElementAddress** was placed there by an operator. Otherwise, the media was placed there by a transport element. Valid only if **dwElementAddress** is a portal element address, and the **MEDIA_PRESENT** status is set.
MEDIA_INVERTED	The media at **dwElementAddress** has been inverted by a **ChangerMoveMedium** command since the media was last at **dwElementAddress**. Valid only if the **MEDIA_ INVERTED_VALID** status is set.

MEDIA_INVERTED_VALID	Indicates the **MEDIA_INVERTED** status is valid.
MEDIA_PRESENT	Media is currently present at **dwElementAddress**. Valid only if supported.
SOURCE_ADDRESS_VALID	Indicates that **dwSourceAddress** is valid.
Returns	If the function is successful, the return value is COMPLETE_SUCCESS. Otherwise it is an error code.

(2) Changer_Get_Parameters

DWORD ChangerGetParameters(hDevice, pdwBufferSize, pciChangerInformation)

```
IN HANDLE                hDevice;
IN PDWORD                pdwBufferSize;
IN PCHANGER_INFORMATION  pciChangerInformation;
```

Parameter	Description
hDevice	Handle of the medium changer device from which to return device capabilities.
pdwBufferSize	Pointer to a dword that should have the size, in bytes, of the buffer specified by **pciChangerInformation**. If the buffer is too small, the dword receives the required size.
pciChangerInformation	Pointer to an information structure from which the application can configure itself to utilize the medium changer driver to its fullest capabilities. The following describes the structure members:

Member	Description
cVendorID[]	Null-terminated string containing the vendor information as returned by a SCSI-2 inquiry command.
cProductID[]	Null-terminated string containing the product information as returned by a SCSI-2 inquire command.
dwNumTransportElements	Number of transport elements in the medium changer device.

dwFirstTransportAddress	Element address of the first transport element in the medium changer device.
dwNumDriveElements	Number of drive elements in the medium changer device.
dwFirstDriveAddress	Element address of the first drive element in the medium changer device.
dwNumStorageElements	Number of storage elements in the medium changer device.
dwFirstStorageAddress	Element address of the first storage element in the medium changer device.
dwNumPortalElements	Number of import/export elements in the medium changer device.
dwFirstPortalAddress	Element address of the first import/export element in the medium changer device.
dwFeatures	Changer features. A value of 1 in the proper bit field indicates a supported feature. The following pages describe the features:

Value (1 in proper bit field)	Description
EJECT_MEDIA_BEFORE_INITIALIZE	The device requires that in a drive element be ejected (unloaded) before calling **ChangerInitializeElementStatus** to initialize that drive element.
EJECT_MEDIA_BEFORE_MOVE	The device requires that media in a given drive element be ejected (unloaded) before calling **ChangerMoveMedium** with that drive element or the source element address.

INITIALIZE_BY_ELEMENT	Supports initializing a single given element via **ChangerInitializeElementStatus**.
INITIALIZE_SCANS_MEDIA_CODE	**ChangerInitializeElementStatus** also automatically scans the code of the media at the specified element(s).
MOVE_DRIVE_TO_DRIVE	The device supports moving media between drive elements: You can specify both the source and destination element address to be that of drive elements in a **ChangerMoveMedium** command.
MOVE_STORAGE_TO_STORAGE	The device supports moving media between storage elements: You can specify both the source and destination element address to be that of storage elements in a **ChangerMoveMedium** command.
MOVE_TRANPORT_TO_TRANSPORT	The device supports moving media between transport elements: You can specify both the source and destination element address to be that of transport elements in a **ChangerMoveMedium** command.
PORTAL_OPERATION_REQUIRED	The portal(s) must be opened/closed each time media is to be moved into/out of the scope of the medium changer device.
PORTALS_PRESENT	The medium changer device has import/export portal(s).
PORTAL_ELEMENTS_AS_STORAGE	The portal element(s) can be used to (temporarily) store media. A portal must be closed to be used as a (temporary) storage location.

POSITION_TRANSPORT	The medium changer device supports positioning the transport elements.
PREVENT_MANUAL_ACCESS	Supports locking the device, preventing the user from manual operation of panel controls and media.
REPORT_MEDIA_PRESCENCE	The medium changer device has the ability to report the prescence of media at all of its element addresses.
RESERVE_RELEASE_DEVICE	Supports reserving/releasing all elements in the medium changer device at once.
RESERVE_RELEASE_ELEMENTS	Supports reserving/releasing specified elements in the medium changer device.
SCAN_MEDIA_CODE_ALL_ELEMENTS	Supports scanning the media code of all the elements at once via **ChangerScanMediaCode**.
SCAN_MEDIA_CODE_BY_ELEMENT	Supports scanning the media code of a single given element via **ChangerScanMediaCode**.
TRANSPORT_ELEMENT_ILLEGAL	It is illegal to specify a transport element address as a source or destination element address in a **ChangerMoveMedium** command. The command expects the source and destination element addresses to be element addresses of one of the other valid element types.
TRANSPORT_ELEMENT_REQUIRED	It is required for every **ChangerMoveMedium** command, either the source or destination element

address must be the element address of the transport element.

Returns If the function is successful, the return value is COMPLETE_SUCCESS. Otherwise it is an error code.

(3) Changer_Get_Status

DWORD ChangerGetStatus(hDevice)

IN HANDLE hDevice;

Parameter	Description
hDevice	Handle of the medium changer device on which to get the status.
Returns	If the device is ready to accept a command without returning an error, the return value is COMPLETE_____SUCCESS. Otherwise, it is an error code.

(4) Changer_Initialize_Element_Status

DWORD ChangerInitializeElementStatus(hDevice, dwElementAddress, dwSpecialInitialize)

IN HANDLE	hDevice
IN DWORD	dwElementAddress;
IN DWORD	dwSpecialInitialize;

Parameter	Description
hDevice	Handle of the medium changer device to initialize/reset element status.
dwElementAddress	Address of the element to be initialized. Valid only if supported. Ignored if the **INITIALIZE___ALL_ELEMENTS** bit is set in **dwSpecialInitialize**.
dwSpecialInitialize	If non-zero, indicates a special function of the initialize process. The following describes the special initialize.

Value (1 in proper bit field)	Description
INITIALIZE_ALL_ELEMENTS	Initialize all elements in the medium changer device.
Returns	If the function is successful, the return is COMPLETE_SUCCESS. Otherwise it is an error code based on the first error encountered.

(5) Changer_Manual_Access

DWORD ChangerManualAccess (hDevice, bPreventAllow)

IN HANDLE	hDevice;
IN BOOL	bPreventAllow;

Parameter	Description
hDevice	Handle of the medium changer device on which to prevent/allow manual operation of panel control(s) and media.
bPreventAllow	If TRUE, prevents the user from manual operation of panel control(s) (eject, move, etc...) and media. If FALSE, allows the user manual operation of panel control(s) and media.
Returns	If the function is successful, the return value is COMPLETE_SUCCESS. Otherwise it is an error code.

(6) Changer_Move_Medium

DWORD ChangerMoveMedium(hDevice, dwTransportAddress, dwSourceAddress, dwDestinationAddress, dwSpecialMove)

```
IN HANDLE       hDevice;
IN DWORD        dwTransportAddress;
IN DWORD        dwSourceAddress;
IN DWORD        dwDestinationAddress;
IN DWORD        dwSpecialMove;
```

Parameter	Description
hDevice	Handle of the medium changer device on which to perform a move medium command. All portal elements must be closed before a move can be performed.
dwTransportAddress	Element address of the transport to perform the move.
dwSourceAddress	Source element address.
dwDestinationAddress	Destination element address.
dwSpecialMove	If non-zero, indicates a special function of the move process. The following describes the special moves:

Value (1 in proper bit field)	Description
INVERT_MEDIA	Invert (flip) the media prior to moving to the destination element address.

Returns If the function is successful, the return value is COMPLETE__SUCCESS. Otherwise it is an error code.

(7) Changer_Portal_Operation

DWORD ChangerPortalOperation(hDevice, dwPortalAddress, dwTransportAddress, bOpenClose)

IN HANDLE	hDevice;
IN DWORD	dwPortalAddress;
IN DWORD	dwTransportAddress;
IN BOOL	bOpenClose;

Parameter	Description
hDevice	Handle of the medium changer device on which to open/close the portal at **dwPortalAddress**.
dwPortalAddress	Element address of the portal element to open/close.
dwTransportAddress	Element address of the transport to be associated with the portal operation. If there is only one transport element in the device, then the element address of that transport should be used.
bOpenClose	Indicates open or close operation: TRUE open the portal, FALSE close the portal.

Returns If the function is successful, the return value is COMPLETE_SUCCESS. Otherwise it is an error code.

* Manual access to the device must be enabled before calling this API.

(8) Changer_Position_Transport

DWORD ChangerPositionTransport(hDevice, dwTransportAddress, dwDestinationAddress, dwSpecialPosition)

```
IN HANDLE       hDevice;
IN DWORD        dwTransportAddress;
IN DWORD        dwDestinationAddress;
IN DWORD        dwSpecialPosition;
```

Parameter	Description
hDevice	Handle of the medium changer device on which to position a given transport element.
dwTransportAddress	Element address of the transport to position.
dwDestinationAddress	Destination element address. Must be the address of a drive, storage, or portal element only.
dwSpecialPosition	If non-zero, indicates a special function of the position process. The following describes the special positions:

Value (1 in proper bit field)	Description
INVERT_TRANSPORT	Invert (flip) the transport element while positioning to the destination element address.

Returns If the function is successful, the return value is COMPLETE__SUCCESS. Otherwise it is an error code.

(9) Changer_Reserve_Release

DWORD ChangerReserveRelease(hDevice, sID, dwStartingAddress, dwRange, dwOperation)

IN HANDLE	hDevice;
IN SHORT	sID;
IN DWORD	dwStartingAddress
IN DWORD	dwRange;
IN DWORD	dwOperation;

Parameter	Description
hDevice	Handle of the medium changer device on which to Reserve/Release elements.
sID	The ID to be associated with the element(s) to be Reserved, or the ID of the element(s) to be Released. For a Reserve operation, the ID is an arbitrary value in the range 0-255. Ignored if the Reserve/Release operation is of **RESERVE/RELEASE_DEVICE** type.
dwStartingAddress	The starting element of a specific element-type to be Reserved. Ignored unless **RESERVE_ELEMENTS** operation. Valid only if supported.
dwRange	Number of elements from **dwStartingElementAddress** to be reserved. Ignored unless **RESERVE____ELEMENTS** operation. Non-zero values are valid only if reserving specified elements is supported.
dwOperation	Type of Reserve/Release operation to perform.

Value (One of the following)	Description

RESERVE_ELEMENTS	Reserve specified elements of a specific element-type.
RELEASE_ELEMENTS	Release specified elements of a specific element-type.
RESERVE_DEVICE	Reserve all elements in the medium changer device.
RELEASE_DEVICE	Release all elements in the medium changer device.

Returns If the function is successful, the return value is COMPLETE__SUCCESS. Otherwise it is an error code.

(10) Changer_Scan_Media_Code

DWORD ChangerScanMediaCode(hDevice, dwElementAddress, dwTransportAddress, dwSpecialScan)

```
IN HANDLE       hDevice;
IN DWORD        dwElementAddress;
IN DWORD        dwTransportAddress;
IN DWORD        dwSpecialScan;
```

Parameter	Description
hDevice	Handle of the medium changer device on which to scan media codes.
dwElementAddress	Address of the element to be scanned. Valid only if supported. Ignored if the SCAN_ALL_ELEMENTS bit is set in dwSpecialScan.
dwTransportAddress	Element address of the transport to be associated with the scan operation. If there is only one transport element in the device, then the element address of that transport should be used.
dwSpecialScan	If non-zero, indicates a special function of the scan process. The following describes the special scan:

Value (1 in proper bit field)	Description
SCAN_ALL_ELEMENTS	Scan media at all elements in the medium changer device.

Returns If the function is successful, the return value is COMPLETE_SUCCESS. Otherwise it is an error code based on the first error encountered.

Fig. 4 is a flow diagram used to explain one manner in which the API 28 of the present invention may be used on a computer system (not shown) together with the execution of components of the software shown in Fig. 2. On start-up, the API 28 will perform medium changer initializations (block 52). If the initializations are successful (block 54), then the API 28 will cause performance of the requested changer functions (block 56) as will be further described. The primary functions that may be performed are for the medium changer 12 to move a medium 16, position the transport, do a portal operation, scan the media and/or get the status of a given element 16. Once the changer functions are successfully completed, the de-initialization of the changer 12 occurs (block 58) and the program is done (block 60). If the changer initializations are not successful (block 54) then the program is done (block 60).

Fig. 5 is a flow diagram showing in more detail the initializations that are performed by the API 28 of the present invention with reference to a number of the calls (1)-(10). The first step is to obtain a handle for the changer (block 62). Next is the call of Changer_Get_Status which returns the status of medium changer 12 (block 64). If there is no error (block 66), the next call is Changer_Get_Parameters (block 68), which returns the capabilities of medium changer 12. If there is no error (block 70), the next call is Changer_Reserve_Release (block 72), to reserve elements 14, 18, 20 and 22 in the medium changer 12 for the initiator being accessed by one of the applications program 26 that may be on a network and which may be initiating this request. Next, if there is no error (block 74), call Changer_Manual__Access (block 76) is executed, which locks up the changer 12 and prevents

manual access to the media 16. Then, if there is no error (block 78), all the elements of medium changer 12 are initialized by calling Changer_Initialize_Element Status (block 80). If there is no error (Block 82), the application may then enter a loop (Blocks 84, 86, 87, 88) obtaining information about each element by calling Changer_Get_Element_Status (Block 84). If media is present at a particular element, and the device supports scanning the media code ("bar code"), Change_Scan_Media_Code may then be called (Block 84) to get the media code. The information returned from these calls may be used at this time to build (Block 87) or if persistent, verify the Internal Inventory. If the status of all the elements has been obtained (block 88), then the initializations have been successfully performed and this program is done (block 90).

As shown on the right side of the flow diagram of Fig. 5, a series 92 of error recovery procedures will occur in connection with each call. For example, if there is an error (block 66) during or on the completion of the call Changer_Get_Status (block 64), then an error handler is invoked (block 94). If the error is recoverable (block 96), then the call (block 64), is executed again. If the error is not recoverable (block 96), then initialization has failed and this program is done (block 98). As can be seen in Fig. 5, a similar error handler procedure for each call can be executed.

If all the initializations have been performed as illustrated in Fig. 5, then the calls shown in Fig. 6 for performing the requested changer functions (block 56) are executed. First, the call Changer_Get_Status (block 100) is executed. If there is no error (block 102), then any one of five functions being requested is

executed (block 104). These functions can be carried out by the calls Changer_Move_Medium, which moves media 16 in the medium changer 12, Changer_Position_Transport which positions the element 20 to a given element address, Changer_Portal_Operation which opens or closes portal element 22 to enable or disable a user from inserting or removing media 16 into the scope of the changer, Changer_Scan_Medium_Code to scan media code at a given element address, or Changer_Get_Element_Status which returns the status of a given element of medium changer 12. If there is no error (block 106), then the Internal Inventory (described more fully below) is updated, if necessary (Block 107), and the changer function has been completed (block 108). Again, a simple error handle procedure 92 similar to that as was shown in Fig. 5 is used as shown for each call (block 100) and (block 104).

Fig. 7 illustrates the flow for performing changer de-initialization (block 58). First, the call Changer Manual_Access (block 110) is executed to allow manual access to the medium changer 12. If there is no error (block 112) then the call Changer__Reserve__Release (block 114) is executed to release any lock on the changer so it may be then used by other initiators. If there is no error (block 116) then de-initialization is done (block 118). Again, as shown to the right of Fig. 7, a simple error handle 92 procedure is invoked at each call.

With respect to the Internal Inventory (Block 107 of Fig. 6) and the VSD Load Table Database 42, there is a difference between the two. The Internal Inventory is an internal database, stored in the computer system (not shown), unique to each changer that the applications program 26 has the *option* of generating. It will contain information about the elements in the

changer 12 such as (i) are they full of media, (ii) if so, what is the media code of that media, (iii) is this one or two sided media, (iv) where was the media last moved from, etc... This Internal Inventory is then updated when required after performing certain changer functions such as Changer_Move_Medium. This Internal Inventory also has the option of being persistent, or being generated every time at initialization of the programs application. The persistent Internal Inventory is especially useful in very large changers 12, where numerous media and drive elements exist, and initialization of every element would take a long period of time. Should for some reasons the applications program shut down, it can be re-started, and the persistent Internal Inventory can be used to continue operations. The applications program would also have the option of calling certain changer API's, and comparing the results to its Internal Inventory to detect problems, discrepancies, or possible user tampering inside the changer 12 while the applications program 26 was down.

Another advantage to building and maintaining an Internal Inventory (persistent or not) is increased performance and reliability. Since the changer APIs require communication with the changer, it is more time consuming than looking up the desired information in the Internal Inventory. Also, most of the information returned from the changer APIs is obtained from the changer's firmware, which is equally prone to bugs as software.

More specifically, the Internal Inventory, whether it is a temporary or persistent database, contains information about the media, relative to the elements in a changer. The applications program can continually access this database to verify user's requests are

valid, or to report back information to the user. It can be used to determine error or inconsistencies with what the APIs are reporting back as well. An advantage to a persistent inventory is the increased ability to recover from power-failures, one of storage management applications' biggest nightmares. In such a situation a changer device's firmware may be reset, such that it cannot provide reliable information regarding its elements relative to the state it was in prior to the power-outage. But if a persistent Internal Inventory (database) is used, the applications program could reference that, and most likely figure out the state it was in when it was last shut-down, and continue with operations, reporting any necessary errors/warnings to the user. Not only does this persistent Internal Inventory help with regards to the changer, but to the devices IN the changer. If there is media in the drives, and there is a power outage, then when the applications program starts back up, it may be in an error state relative to those drives because it was not expecting media to be in them at start-up. The Internal Inventory could then be used to tell the applications program what storage elements those tapes came from, and decide how to recover, whether to continue drive operations, or move them back to their original storage element.

If the Internal Inventory method is chosen, as indicated above a unique Internal Inventory Database is required for each changer the applications program 26 wishes to concurrently access through the changer APIs.

Examples of the VSD Load Table Database 42 and the Internal Inventory database are given below.

VSD Load Table Database example:

Vendor	Product	VSD Name	Comment
Exabyte	EXB-120	EXB120.VSD	Exabyte 120 changer
•			
•			
•			
Archive	Python	DIAMDBAK.VSD	Conner Diamondback changer.

Internal Inventory Database example for a given changer:
(May contain more or less fields of information, at the discretion of the application developer)

Element Type	Element Address	Media Present	Pri. Media Code	Alt. Media Code	Media Two Sided?	Current Media Side	Address Media Last Moved From
Drive	116	Y	"842C..."	na	N	Front	0
Drive	117	N	na	na	na	na	na
Drive	118	Y	"942D..."	na	N	Front	85
Drive	119	Y	"345C..."	na	N	Front	32
Storage	0	N	na	na	na	na	na
•	•						
•	•						
•	•						
Storage	115	y	"539D..."	na	N	Front	116
Portal	120	N	na	na	na	na	na

Transport	121	N	na	na	na	na	na

na = not available

The API 28 allows for not just one, but two different approaches in its use. One is an interactive approach where the APIs are called every time information is needed about an element or prior to performing an operation. This approach relies on the device's ability to report the information based on the devices firmware, which is the origin of the information that is reported back through the APIs.

The second approach is to initially call the information APIs (i.e. ChangerGetElementStatus) to build the Internal Inventory, and then use that inventory whenever possible to obtain information. The changer APIs then only to need to be called for functional requests. The information in the Internal Inventory can also be used to check against an informational API, or to recover from an error state.

In summary, the conceptual definition of the changer API of the present invention is different than that of the low level IOCTL kernel API and abstract high level API described above. The changer API of the present invention is defined such that it creates a very device-centric view of a changer object so that the applications program can tailor itself to use the different physical changers 12 in a more organized fashion without actually knowing what the make and model of changer 12 it is using. The API 28 is a layer over the IOCTL API to hide the exact driver/device protocols, making the API 28 more like what applications programmers are used to interfacing with.

Properly implemented there will be far less

coding, testing, and bugs in an application using the API 28 of the present invention. If a given device 12 has some kind of new feature, it can be added to the 32 bit features word (Fig. 3B), without breaking the syntax of the API 28. Then, minimal lines of code would typically be needed to recognize and act on that feature. And other previously developed device drivers 34 do not need to be altered because they do not return that feature, so the applications program 26 will never enter a code path assuming that feature for those devices 12.

The task of making the API 28 conform to its definition takes place under the API before making the IOCTL call, or in the drivers. The application no longer needs to be altered every time a device 12 that needs special treatment to meet the definition of the changer API 28 is to be supported. In the rare instance when additional code is required in the applications program 26, it is clearly defined and organized, and once put in place, will work for every device 12 that requires the support of that new code. Aiding in accomplishing this is the design of the VSD/SCSI Changer Driver architecture of Fig. 2 lends itself to better organization, separation, and ease of adding new device support at the driver level than traditional class or filter driver architectures. Only VSDs 34 should have to be added/altered, not the SCSI changer driver 38.

None of this precludes an applications programmer from wrapping a higher level API around the portions of the API 28 of the present invention to help abstract the changer operations. But if properly defined and implemented, this higher level API wrapper would not have the risks discussed above associated with actually defining the device interface at that higher, more

abstract level. This illustrates the flexibility of changer APIs 28.

Furthermore, because of the design of the API 28, the applications program 26 views a medium changer 10 as an object with a handle, which lends itself to be easily utilized using object-oriented program methodologies. For example, a C++ class "wrapper" can be placed around the API to create a medium changer "class", which when instantiated, would become a medium changer object.

Finally, previously there was described an error handling procedure 92 for various API calls. Below is a set of medium changer API error possibilities, together with unrecoverable and recoverable errors, that the API can return. These are a very robust set of error codes, including many that are not implied in the SCSI-2 specification, such as errors related to portals, media code scanning, current device modes and firmware update warnings.

TOTAL MEDIUM CHANGER API ERROR POSSIBILITIES

COMPLETE_SUCCESS
MCERROR_HARDWARE
MCERROR_DEVICE_BUSY
MCERROR_BUS_WAS_RESET
MCERROR_INSUFFICIENT_MEMORY
MCERROR_INVALID_REQUEST
MCERROR_NO_SUCH_DEVICE
MCERROR_DRIVER_TIMEOUT
MCERROR_INCORRECT_BUFFER_SIZE
MCERROR_MEDIA_NOT_EJECTED
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_DESTINATION_ELEMENT_FULL
MCERROR_SOURCE_ELEMENT_EMPTY
MCERROR_NOTHING_MOVED
MCERROR_DISCONNECTED_DURING_CMD
MCERROR_TRANSPORT_PREVIOUSLY_FULL
MCERROR_DRIVE_DOOR_CLOSED

MCERROR_DEVICE_LOCKED
MCERROR_INVALID_RANGE
MCERROR_DOOR_IS_OPEN
MCERROR_DOOR_WAS_OPENED
MCERROR_PORTAL_IS_OPEN
MCERROR_PORTAL_WAS_OPENED
MCERROR_MAGAZINE_CHANGED
MCERROR_MAGAZINE_NOT_PRESENT
MCERROR_DRIVE_NOT_PRESENT
MCERROR_DRIVE_MEDIA_LOAD_FAILED
MCERROR_INCOMPATIBLE_MEDIA
MCERROR_INVENTORY_MAY_BE_CORRUPT
MCERROR_CANNOT_READ_LABEL
MCERROR_INCORRECT_MODE
MCERROR_MODE_WAS_CHANGED
MCERROR_FIRMWARE_WAS_CHANGED
MCERROR_RESERVATION_CONFLICT
MCERROR_VSD_DRIVER_CALL_FAILED
MCERROR_SCSI_CHANGER_INTERNAL
MCERROR_UNRECOGNIZED_STATUS

UNRECOVERABLE ERRORS:
MCERROR_HARDWARE
MCERROR_INSUFFICIENT_MEMORY
MCERROR_DRIVER_TIMEOUT
MCERROR_INVALID_REQUEST
MCERROR_NO_SCH_DEVICE
MCERROR_VSD_DRIVER_CALL_FAILED
MCERROR_SCSI_CHANGER_INTERNAL
MCERROR_UNRECOGNIZED_STATUS

RECOVERABLE ERRORS:
Common to all medium changer APIs:
MCERROR_DEVICE_BUSY
MCERROR_BUS_WAS_RESET
MCERROR_DISCONNECTED_DURING_CMD
MCERROR_DOOR_IS_OPEN
MCERROR_DOOR_WAS_OPENED
MCERROR_PORTAL_WAS_OPENED
MCERROR_MAGAZINE_CHANGED
MCERROR_INCOMPATIBLE_MEDIA
MCERROR_RESERVATION_CONFLICT

ChangerGetElementStatus()additional error possibilities:
MCERROR_INVENTORY_MAY_BE_CORRUPT
MCERROR_MAGAZINE_NOT_PRESENT
MCERROR_CANNOT_READ_LABEL

ChangerGetParameters() additional error possibilities:

ChangerGetStatus() additional error possibilities:

ChangerInitializeElementStatus() additional error possibilities:
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_INVALID_RANGE

ChangerManualAccess() additional error possibilities:

ChangerMoveMedium() additional error possibilities:
MCERROR_MEDIA_NOT_EJECTED
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_DESTINATION_ELEMENT_FULL
MCERROR_SOURCE_ELEMENT_EMPTY
MCERROR_NOTHING_MOVED
MCERROR_TRANSPORT_PREVIOUSLY_FULL
MCERROR_MAGAZINE_NOT_PRESENT
MCERROR_DRIVE_NOT_PRESENT
MCERROR_DRIVE_DOOR_CLOSED
MCERROR_DRIVE_MEDIA_LOAD_FAILED

ChangerPortalOperation() additional error possibilities:
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_DEVICE_LOCKED

ChangerPositionTransport() additional error possibilities:
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_DRIVE_NOT_PRESENT
MCERROR_MAGAZINE_NOT_PRESENT

ChangerReserveRelease() additional error possibilities:
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_INVALID_RANGE

ChangerScanMediaCode() additional error possibilities:

```
MCERROR_INVALID_ELEMENT_ADDRESS
MCERROR_SOURCE_ELEMENT_EMPTY
```

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments said with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

CLAIMS

1. An applications program interface (API) for use in a computer system for interfacing one or more applications programs with one or more medium changer devices, each one of the medium changer devices having a plurality of addressable physical elements and being capable of supporting at least one data storage medium, comprising:

(a) a plurality of applications program interface (API) calls for obtaining information about the status of the elements of any of the medium changer devices and the parameters of any of the devices, and for moving the medium from one of the elements to another of the elements of any one of the devices; and

(b) wherein one of said calls has a data structure including a plurality of first members, one of said first members being for storing information about the status of the elements, and another of said calls has a data structure including a plurality of second members, one of said second members being for storing information indicating one or more features that any one of the devices supports.

2. An applications program interface (API) according to Claim 1, wherein another of said plurality of calls provides the applications program with the ability to initialize the status of one or more of the elements of one of the devices, element-by-element.

3. An applications program interface (API) according to Claim 1, wherein another of said calls provides the applications program with the ability to move the

medium, including inverting the medium prior to moving the medium.

4. An applications program interface (API) according to Claim 1, wherein said second members of said another of said calls are for storing information about a vendor of any one of the devices and about the device itself.

5. An applications program interface (API), according to Claim 1, wherein said another of said calls may return information in addition to information identified in SCSI-2 specifications.

6. An applications program interface (API) according to Claim 5, wherein said additional information includes PORTALS PRESENT and EJECT MEDIA BEFORE MOVE.

7. A method in a computer system for interfacing any of a plurality of applications programs with any of a plurality of medium changer devices supporting a set of SCSI commands, using an applications program interface (API), comprising:

(a) receiving from the applications program a pointer to a data structure storing changer device parameters from which the applications program can configure itself to utilize a given medium changer device;

(b) getting information about the vendor of the device, the specific device itself, and the features supported by the changer; and

(c) building an inventory as a database of the information obtained from the API.

8. A multilayered software architecture for use in a computer system for supporting one or more medium changer devices comprising:

(a) an applications program;

(b) an applications program interface layered beneath said applications program;

(c) an I/O manager of an operating system layered beneath said applications program interface;

(d) a plurality of different vendor specific drivers layered beneath said I/O manager;

(e) a SCSI changer driver layered beneath said plurality of vendor specific drivers for driving the one or more medium changer drivers; and

(f) a database interfacing with said applications program and said SCSI changer driver for storing information.

9. A multilayered software architecture according to Claim 8 wherein said applications program interface is structured such that said applications program views each of the medium changer devices as an object with a handle.

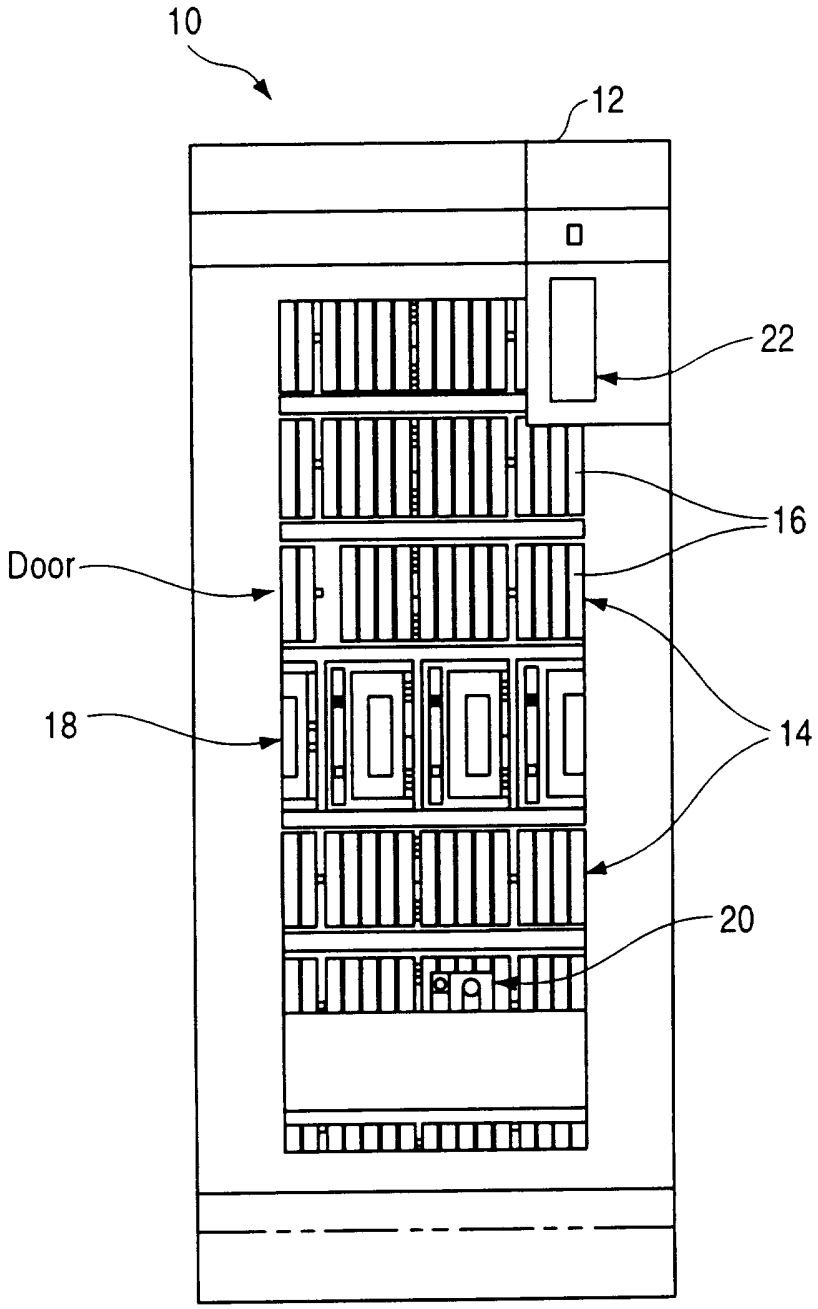


FIG. 1

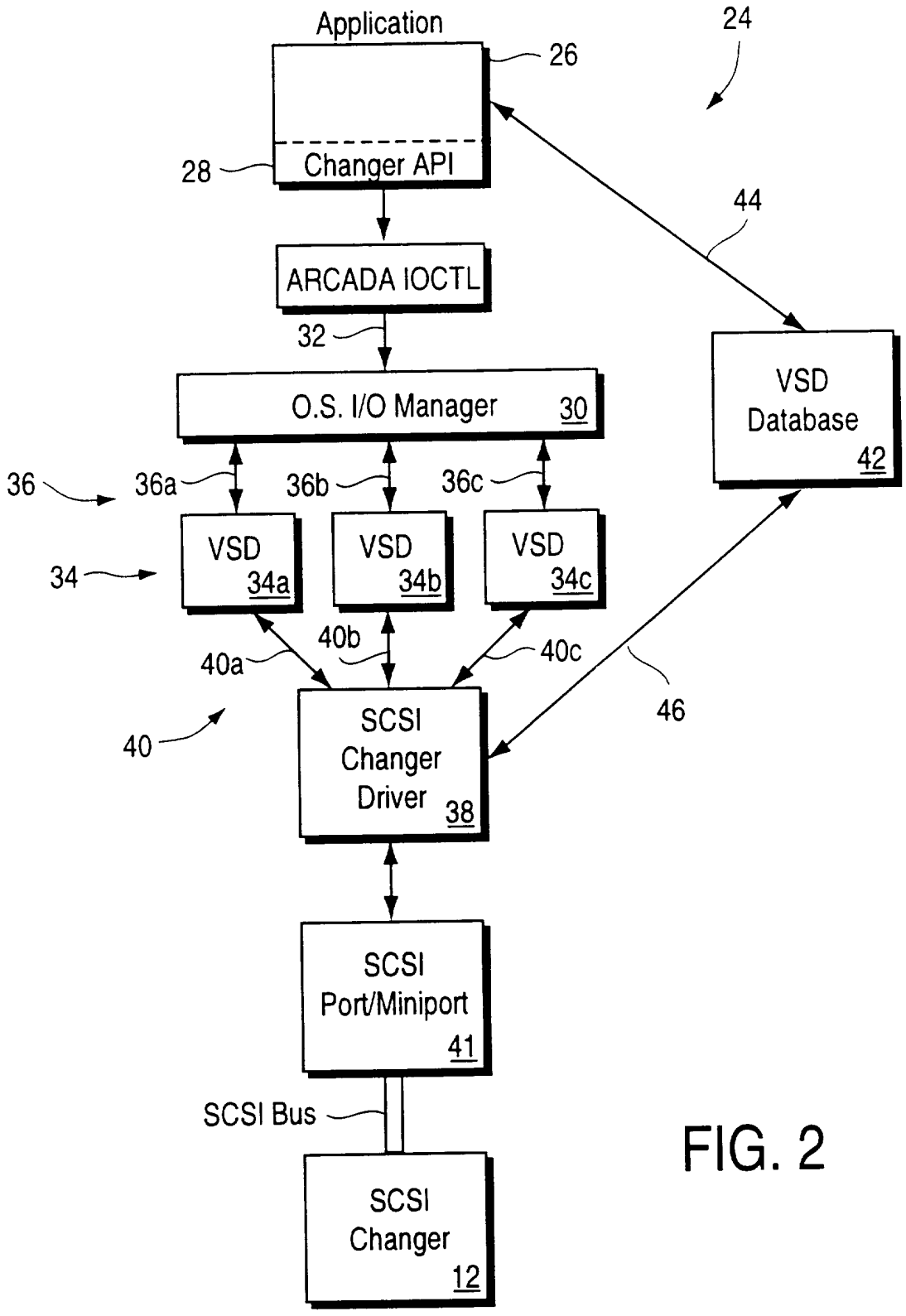

Application
26
24
28
Changer API
44
ARCADA IOCTL
32
O.S. I/O Manager
30
VSD
Database
42
36
36a
36b
36c
VSD
34a
VSD
34b
VSD
34c
34
40b
40a
40c
46
40
SCSI
Changer
Driver
38
SCSI
Port/Miniport
41
SCSI Bus
SCSI
Changer
12

FIG. 2

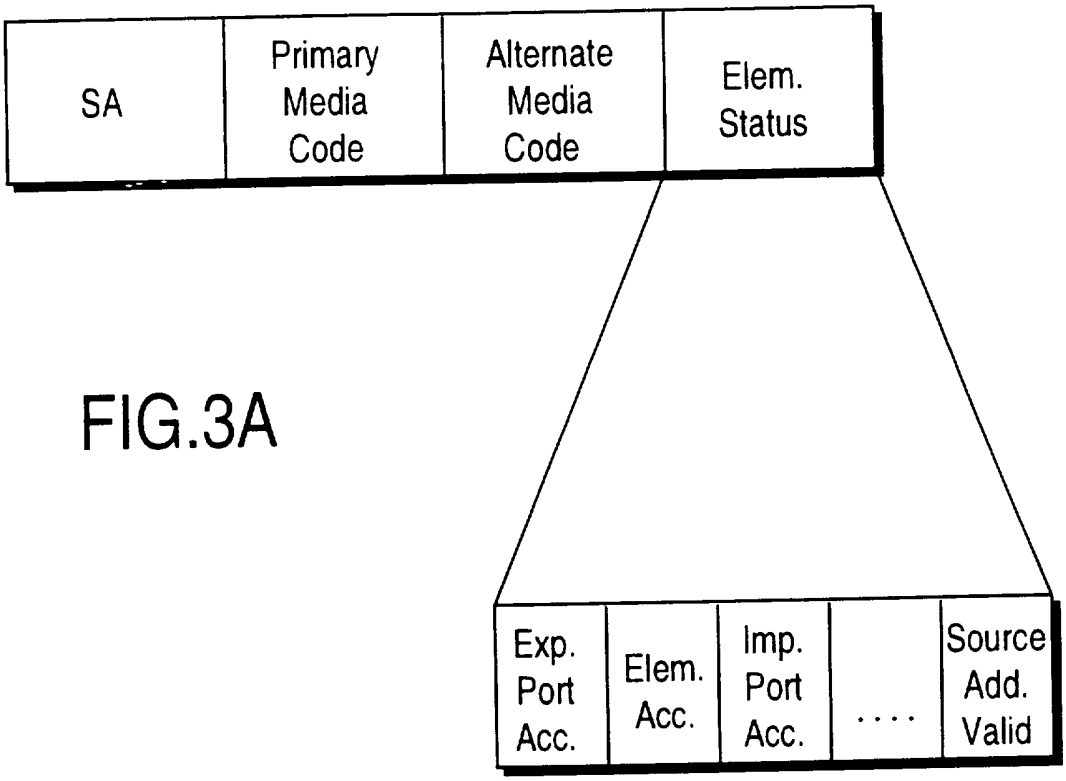

SA
Primary Media Code
Alternate Media Code
Elem. Status
Exp. Port Acc.
Elem. Acc.
Imp. Port Acc.
....
Source Add. Valid

FIG.3A

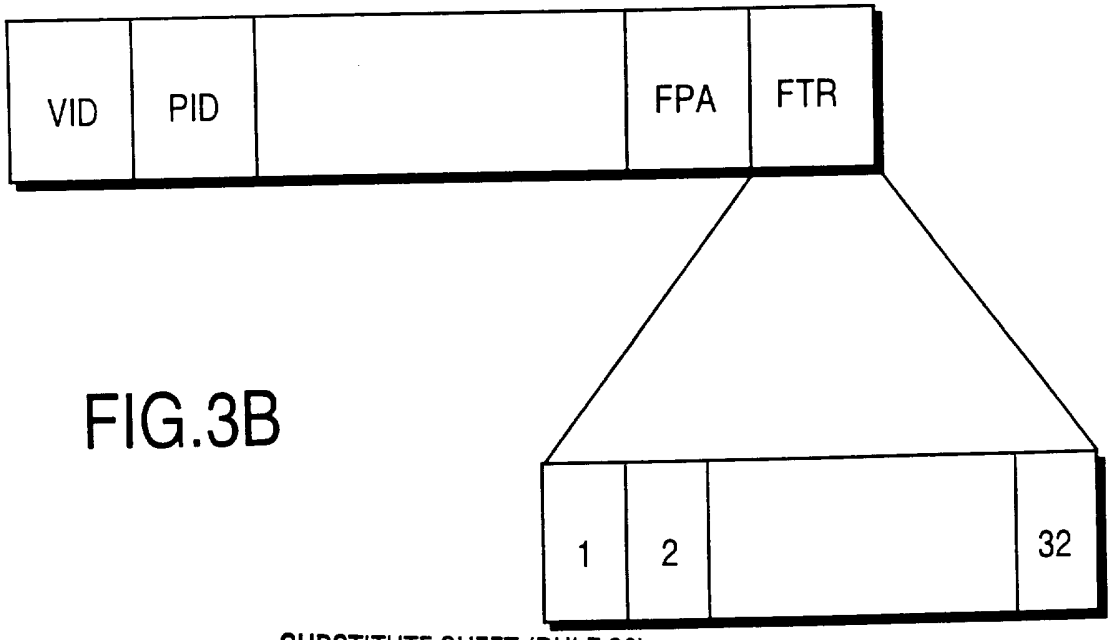
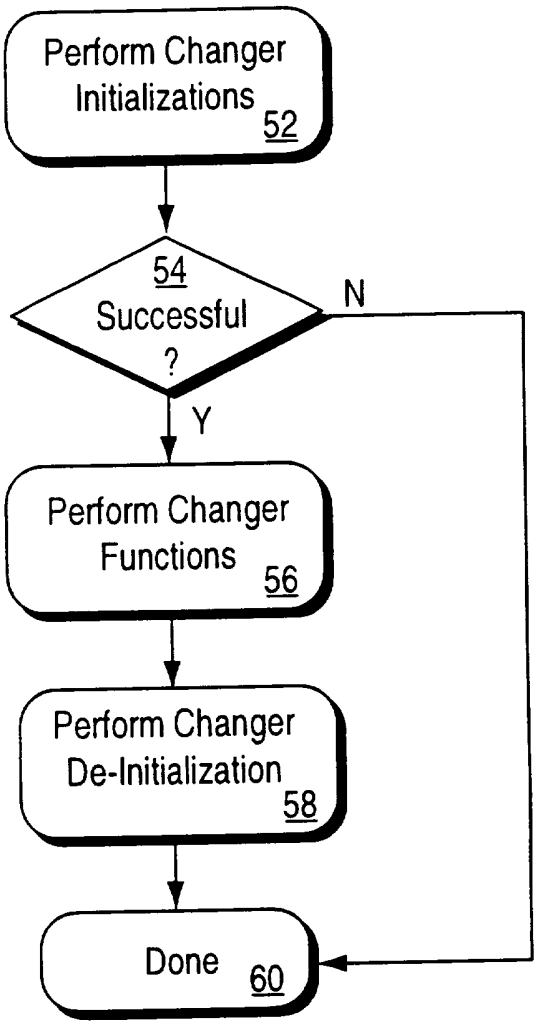

VID
PID
FPA
FTR
1
2
32

FIG.3B



INVOKE ERROR HANDLER may or may not notify the user of the error.

FIG.4

FIG. 5A

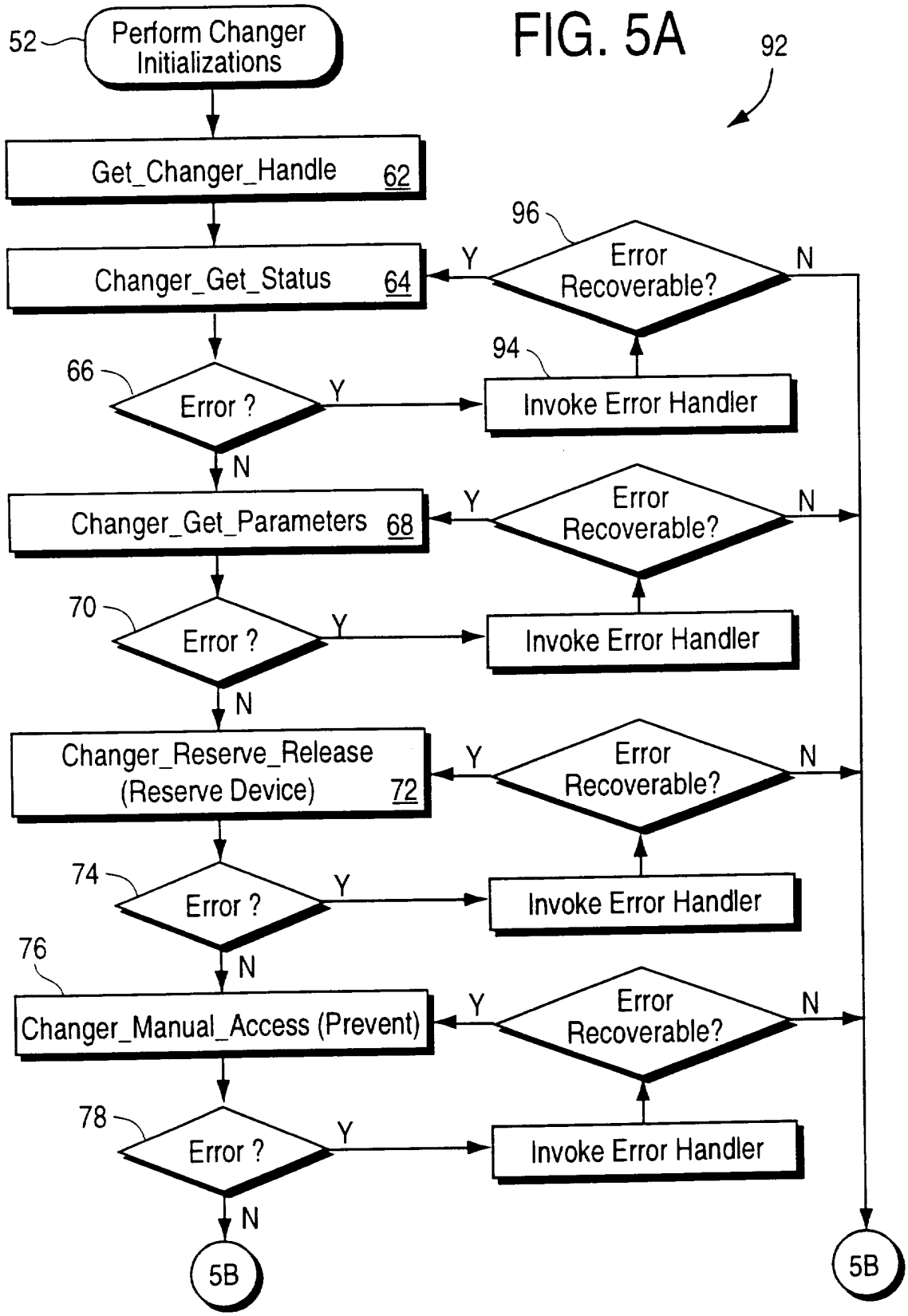
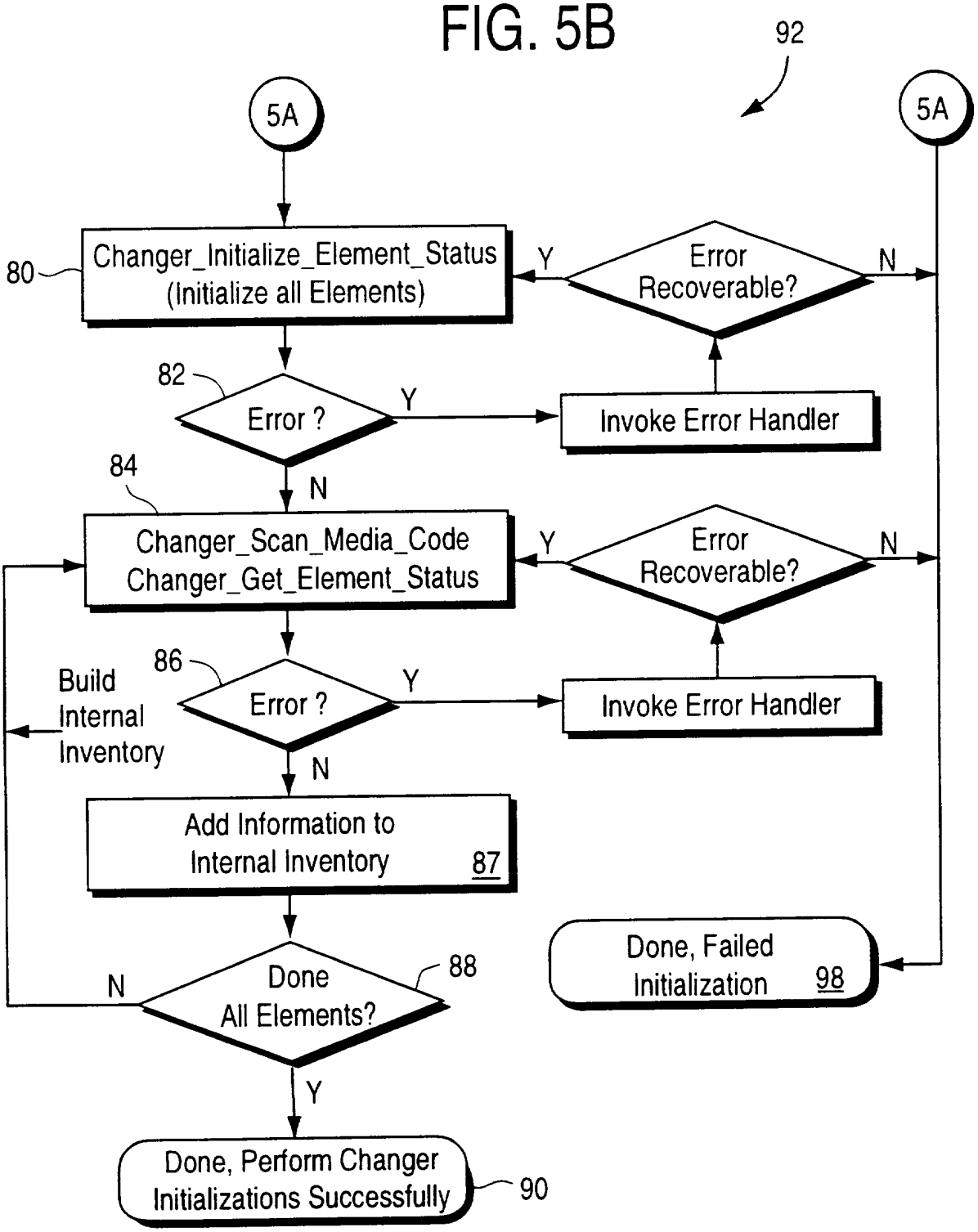

52
Perform Changer Initializations
92
Get_Changer_Handle 62
96
Changer_Get_Status 64
Y
Error Recoverable?
N
66
Error ?
Y
94
Invoke Error Handler
N
Changer_Get_Parameters 68
Y
Error Recoverable?
N
70
Error ?
Y
Invoke Error Handler
N
Changer_Reserve_Release (Reserve Device) 72
Y
Error Recoverable?
N
74
Error ?
Y
Invoke Error Handler
N
76
Changer_Manual_Access (Prevent)
Y
Error Recoverable?
N
78
Error ?
Y
Invoke Error Handler
N
5B
5B

FIG. 5B


92
5A
5A
80
Changer_Initialize_Element_Status
(Initialize all Elements)
Y
Error
Recoverable?
N
82
Error ?
Y
Invoke Error Handler
N
84
Changer_Scan_Media_Code
Changer_Get_Element_Status
Y
Error
Recoverable?
N
86
Error ?
Y
Invoke Error Handler
Build
Internal
Inventory
N
Add Information to
Internal Inventory
87
Done, Failed
Initialization
98
N
Done
All Elements?
88
Y
Done, Perform Changer
Initializations Successfully
90

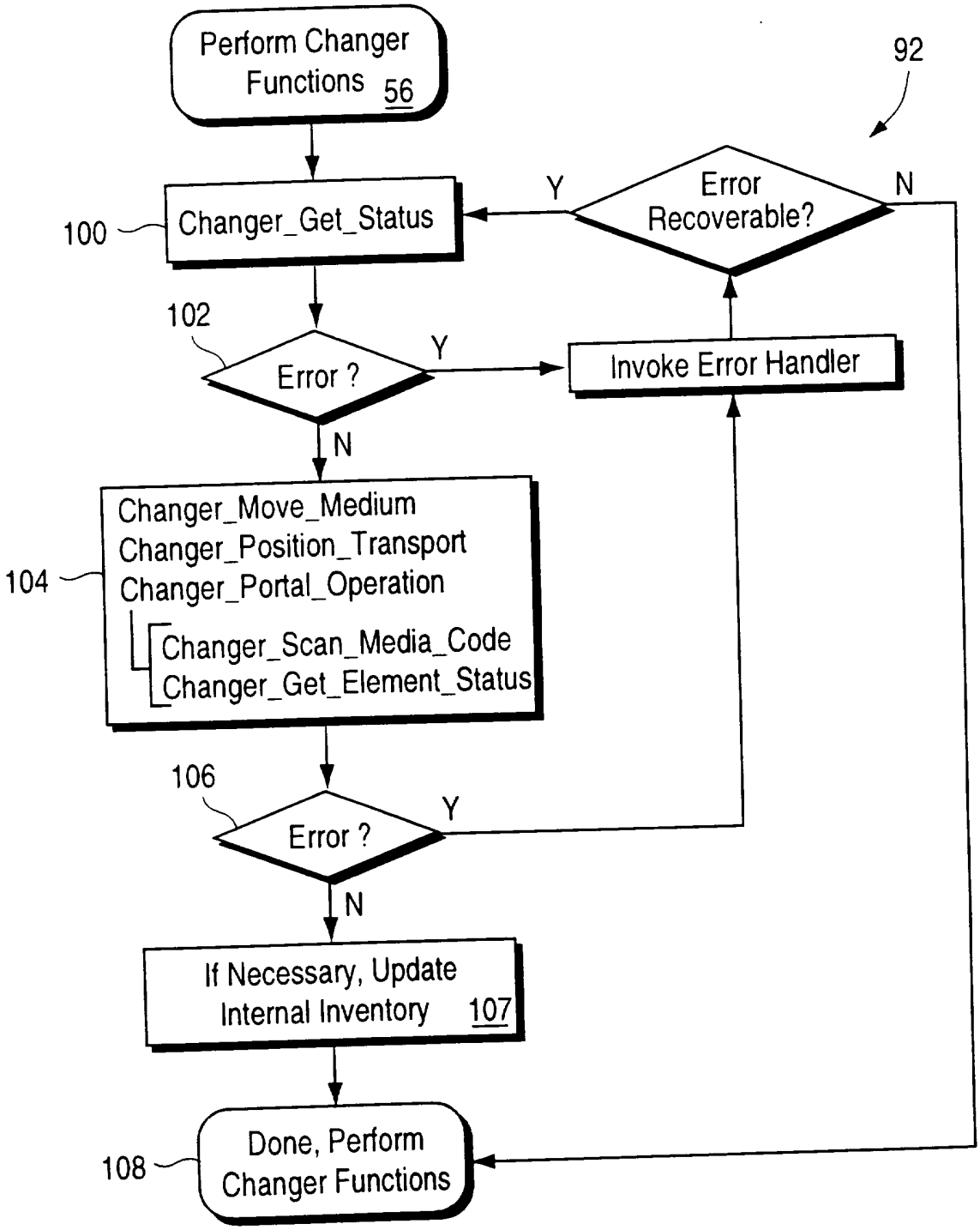

Perform Changer Functions 56
92
Error Recoverable?
Y
N
100
Changer_Get_Status
102
Error ?
Y
Invoke Error Handler
N
104
Changer_Move_Medium
Changer_Position_Transport
Changer_Portal_Operation
Changer_Scan_Media_Code
Changer_Get_Element_Status
106
Error ?
Y
N
If Necessary, Update Internal Inventory 107
108
Done, Perform Changer Functions

FIG. 6

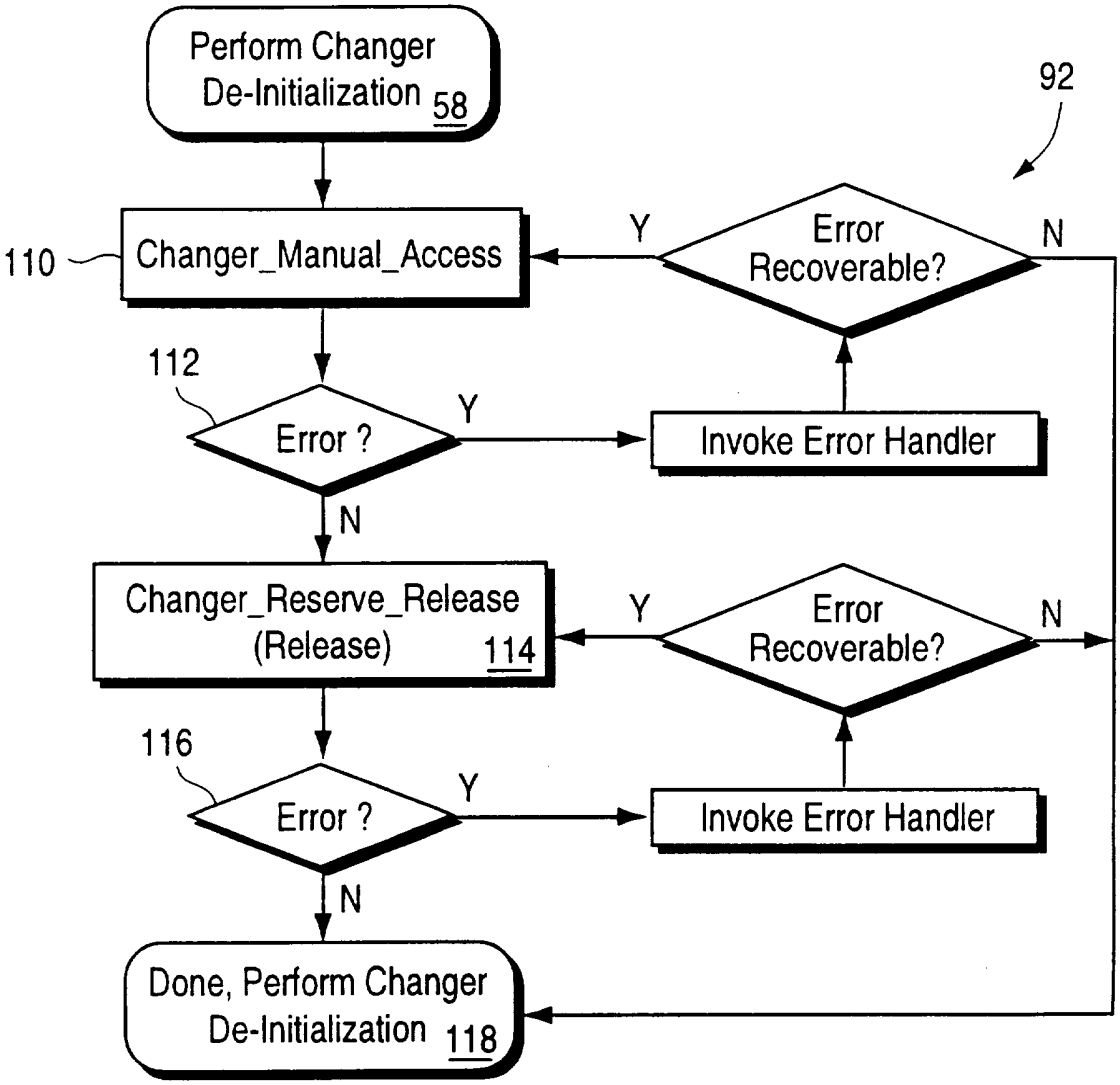

Perform Changer De-Initialization 58
92
110
Changer_Manual_Access
Y
Error Recoverable?
N
112
Error ?
Y
Invoke Error Handler
N
Changer_Reserve_Release (Release) 114
Y
Error Recoverable?
N
116
Error ?
Y
Invoke Error Handler
N
Done, Perform Changer De-Initialization 118

FIG. 7

INTERNATIONAL SEARCH REPORT

Inter nal Application No
PCT/US 95/10763

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44 G06F13/10

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP,A,0 371 941 (IBM) 6 June 1990 see column 6, line 1 - column 10, line 50; figures 1-4	1,7,8
A	PROCEEDINGS SUPERCOMPUTING 1993, 15 - 19 November 1993 PORTLAND, US, pages 462-471, GALBREATH ET AL 'Applications-Driven Parallel I/O' see page 464, right column, paragraph 5 - page 468, left column, paragraph 2; figure 1	1,7,8
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 8, August 1994 NEW YORK US, pages 483-486, 'Message Logging Services for Personal Computer Systems' see the whole document	1,7,8

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 November 1995	27.12.95

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+ 31-70) 340-3016	Gill, S

INTERNATIONAL SEARCH REPORT

. ıformation on patent family members

Inten ıal Application No
PCT/US 95/10763

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-371941	06-06-90	US-A-	5097533	17-03-92
		JP-A-	2201653	09-08-90