A `gdisk` Walkthrough

by Rod Smith, rodsmith@rodsbooks.com

Last Web page update: 4/18/2022, referencing GPT fdisk version 1.0.9

This Web page, and the associated software, is provided free of charge and with no annoying outside ads; however, I did take time to prepare it, and Web hosting does cost money. If you find GPT fdisk or this Web page useful, please consider making a small donation to help keep this site up and running. Thanks!

Donate $1.00	Donate $2.50	Donate $5.00	Donate $10.00	Donate $20.00	Donate another value
Donate	Donate	Donate	Donate	Donate	Donate

Note: This page is part of the documentation for my GPT fdisk program.

GPT fdisk consists of three programs:

- `gdisk` — An interactive text-mode program similar to `fdisk`
- `sgdisk` — A command-line program intended for use in scripts or by experts who need quick and direct access to a specific feature
- `cgdisk` — A curses-based interactive text-mode program similar to `cfdisk`

A fourth tool, FixParts, is part of the GPT fdisk source package but is different enough that I distribute its binaries separately. (Many Linux distributions package all four programs together in a package called `gdisk`, though.) This page documents use of the `gdisk` tool. Separate pages provide similar documentation for `cgdisk` and `sgdisk`, A `cgdisk` Walkthrough and An `sgdisk` Walkthrough.

The GPT fdisk package includes a man page that documents the `gdisk` program in the usual way. If you want to read up on all its options, please refer to that document. This page takes a different approach: It walks you through some common operations, explaining each one.

You must launch `gdisk` with administrative privileges on all platforms except when editing disk image files or, on some systems, removable disks. Under Linux, FreeBSD, or macOS, this is done by logging in as the `root` user, using `su` to acquire `root` privileges, or using the `sudo` utility. Under Windows, you can right-click the Command Prompt program and select the "Run as Administrator" option, then use the resulting window to run `gdisk`.

macOS users: macOS 10.11 ("El Capitan") adds a feature called System Integrity Protection (SIP), or less formally, "rootless." This system blocks access to certain critical aspects of the OS and hardware by third-party programs, including GPT fdisk. Thus, GPT fdisk's capabilities are limited under macOS 10.11 or later unless SIP is disabled. Specifically, low-level access to the system disk is forbidden, so you cannot repartition it. Access to USB flash drives remains possible, though. Disabling SIP is covered on several Web sites, including here and here. My rEFInd boot manager can disable SIP, as described here. Alternatively, you can run GPT fdisk from a Linux emergency disk.

You launch `gdisk` in much the same way as `fdisk`, although `gdisk` supports very few command-line arguments. (You may pass it the `-l` option to see the partition table and then quit, but that's it.) You must know the name of the device file that's used to access the disk. This varies from one OS to another:

- **Linux**—Device filenames typically take the form `/dev/sdx`, where *x* is a lowercase letter—`/dev/sda` is the first disk, `/dev/sdb` is the second disk, and so on. Sometimes, Linux device filenames take the form `/dev/hdx` rather than `/dev/sdx`. Newer

MMC-style SSDs often have filenames like `/dev/mmcblkx`, and NVMe devices usually have filenames of the form `/dev/nvme0nx`, where *x* is a number from `0` up.

- **FreeBSD**—Device filenames take the form `/dev/ad#` or `/dev/da#`, where *#* is a number, as in `/dev/ad0`.
- **macOS**—You specify disk devices as `/dev/disk#`, where *#* is a number, as in `/dev/disk0`.
- **Windows**—Windows disk device references officially take the form `\\.\physicaldrive#`, where *#* is a number, as in `\\.\physicaldrive0`. Since this is awkward, `gdisk` supports a simpler specification of `#:`, as in `0:` to refer to the first disk. Note that letters followed by a colon, as in `C:` or `D:`, refer to partitions, not whole disks; `gdisk` doesn't accept partition names as drive identifiers.

In the Unix-style OSes, partition numbers are normally appended to the disk device filename, either directly (as in `/dev/sda1` to refer to the first partition on `/dev/sda`) or with a `p` between the device filename and partition number, as in `/dev/nvme0n1p2`, to refer to the second partition on `/dev/nvme0n1`). Although GPT fdisk will not complain if it's launched on a partition device, this usage is almost always *incorrect!* You must normally partition the *whole-disk device*. The main time you'd partition a partition would be if the partition is being used as a virtual disk device for a computer virtualization tool, such as VMWare or VirtualBox.

This walkthrough uses as a starting point the partitioning of a 14.6 GiB USB flash drive so that it contains three partitions: One FAT partition for data exchange among multiple OSes, one ext4fs partition for use by Linux alone, and one UFS partition for use by FreeBSD. This scenario is admittedly somewhat artificial, since most people don't use USB flash drives in this way; but this is similar to what you might do when setting up a hard disk for use by multiple OSes, or perhaps even one OS.

The starting point is a drive with a single FAT partition in an MBR partition table. This starting point illustrates what happens when you start `gdisk` on a disk with an existing MBR partition table:

```
# gdisk /dev/sdf
GPT fdisk (gdisk) version 1.0.9

Partition table scan:
  MBR: MBR only
  BSD: not present
  APM: not present
  GPT: not present


***************************************************************
Found invalid GPT and valid MBR; converting MBR to GPT format
in memory. THIS OPERATION IS POTENTIALLY DESTRUCTIVE! Exit by
typing 'q' if you don't want to convert your MBR partitions
to GPT format!
***************************************************************


Command (? for help):
```

When `gdisk` starts, it performs a scan for four types of existing partition tables and displays the results. MBR is the common Master Boot Record partitioning system; BSD is the Berkeley Standard Distribution (BSD) disklabel system used on some computers that run a BSD (FreeBSD, OpenBSD, etc.); APM is the Apple Partition Map used on 680x0- and PowerPC-based Macintoshes; and GPT is of course the GUID Partition Table. The BSD and APM scans report either `present` or `not present`, but GPT can report three states (`present`, `not present`, or `damaged`), and there are four MBR states (`MBR only`, `protective`, `hybrid`, or `not present`). The normal state for an MBR-only disk is as shown above, and the normal state for a GPT disk is `MBR: protective` and `GPT: present`. Hybrid MBRs (described on my Hybrid MBRs page) change the MBR state to `hybrid`. Other combinations are possible, some of which indicate that a disk has been re-partitioned for a new partitioning system without completely erasing the old partition table.

When starting `gdisk` on a disk with existing MBR or BSD disklabel partitions and no GPT, the program displays a message surrounded by asterisks about converting the existing partitions to GPT, as just shown. This message is intended to scare you away if you launch `gdisk` on the wrong disk by accident or if you don't know what you're doing.

Sometimes, when you launch `gdisk` on an MBR disk, you'll see a warning like the following:

```
Warning! Secondary partition table overlaps the last partition by
33 blocks!
You will need to delete this partition or resize it in another utility.
```

This can happen when the final partition of an MBR disk runs right up to the end, overlapping the backup GPT partition table. As the warning says, you can either delete that partition or resize it in another utility before converting to GPT form. If you're following along to this tutorial with your own USB disk (that holds no data you care about), you can ignore this warning and delete the partition, as described shortly.

The `Command (? for help):` prompt is taken straight from Linux `fdisk`, except that GPT fdisk uses `?` as the prompt for help, whereas `fdisk` uses `m`. If you type **`?`**, you'll see a list of available commands:

```
Command (? for help): ?
b       back up GPT data to a file
c       change a partition's name
d       delete a partition
i       show detailed information on a partition
l       list known partition types
n       add a new partition
o       create a new empty GUID partition table (GPT)
p       print the partition table
q       quit without saving changes
r       recovery and transformation options (experts only)
s       sort partitions
t       change a partition's type code
v       verify disk
w       write table to disk and exit
x       extra functionality (experts only)
?       print this menu
```

`fdisk` users will recognize many of these commands, such as `d`, `n`, and `p`. Although some of these don't work in *exactly* the same way as in `fdisk`, most of them are very similar. For instance, suppose you want to print the disk's partition table (**`p`**) to verify that you're working on the disk you think you're working on and then delete the partition (**`d`**):

```
Command (? for help): p
Disk /dev/sdf: 30720000 sectors, 14.6 GiB
Model: STORE N GO
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): ADBD4AAA-F174-4ADF-A74E-C5BBF0B29526
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 30719966
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048        30717951   14.6 GiB    0700  Microsoft basic data

Command (? for help): d
Using 1

Command (? for help):
```

The disk data shown by `p` is a bit different from `fdisk`'s, but it's similar. One salient point is that, because GPT knows nothing about CHS geometry, `gdisk` measures all partitions in sectors (aka blocks). The original converted MBR partition begins at sector 2048 and ends at sector 30,717,951. The disk is 30,720,000 sectors in size, and the last usable sector is 30,719,966 — GPT stores a backup partition table at the end of the disk, so there are normally 33 sectors at the end of the disk devoted to this use.

The `Code` column of the output shows the partition type code. Because GPT employs a 16-byte GUID number for storing partition type codes, displaying this raw code would be both space-consuming and difficult for people to parse. Thus, `gdisk` translates the GUID code into a variant of MBR type codes. In particular, the codes used by `gdisk` (when they're equivalent to MBR type codes) are the MBR codes multiplied by hexadecimal 0x0100, and displayed in hexadecimal. The 0x0700 code shown above is therefore equivalent to an MBR code of 0x07, which is the code for HPFS/NTFS.

"But wait," you say, "I thought the disk had a FAT partition!" Indeed it does. On GPT disks, Windows uses a single GUID code for all its data partitions, be they FAT or NTFS. In the past, the same code has been used in Linux for its data partitions. (More on this shortly....) Thus, in this case several different MBR codes are all translated into a single GPT GUID code. GPT fdisk uses, somewhat arbitrarily, the 0x0700 code (or more precisely, EBD0A0A2-B9E5-4433-87C0-68B6B72699C7) for all of these.

The `Name` column displays a free-form text string that you can modify. Starting with version 0.4.0, `gdisk` places the name associated with the partition type code in this field when it creates or converts partitions, but you can change this default, as described shortly. Note that GPT fdisk accesses the *GPT data structure's* name for a partition. Most filesystems also support a name that's part of the *filesystem's data structure*. The two can match, but they don't have to, and GPT fdisk makes no attempt to synchronize the two names. Confusingly, some partitioning tools, including GParted under Linux and both Disk Utility and diskutil under macOS, display the filesystem name and hide the GPT name.

With the existence of this single partition as (presumably) sufficient identification that this is the disk you want to modify, the `d` command deletes the partition. Since only one partition exists, `gdisk` doesn't prompt you for a number; it just responds `Using 1` and then displays another command prompt.

At this point you presumably want to begin adding your new partitions. As with `fdisk`, you can use `n` to do this task:

```
Command (? for help): n
Partition number (1-128, default 1):
First sector (34-30719966, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-30719966, default = 30717951) or {+-}size{KMGTP}: +8G
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300): L
Type search string, or  to show all codes: data
0700 Microsoft basic data               4200 Windows LDM data
4201 Windows LDM metadata               a005 Android metadata
a008 Android data                       a03a Android user data
a200 Atari TOS basic data               a580 Midnight BSD data
c001 HP-UX data                         e900 Veracrypt data
f101 Fuchsia durable mutable encrypted  f103 Fuchsia factory ro system data
f104 Fuchsia factory ro bootloader data f106 Fuchsia verified boot metadata (sl
f10a Fuchsia Data                       f115 Fuchsia verified boot metadata (A)
f116 Fuchsia verified boot metadata (B) f117 Fuchsia verified boot metadata (R)

Hex code or GUID (L to show codes, Enter = 8300): 0700
Changed type of partition to 'Microsoft basic data'
```

In this example, I pressed the Enter key to accept the default partition number and first sector values. The default starting sector is at the start of the largest contiguous block of free space, but it rounded up to the current alignment value, which defaults to 2048 sectors. In this case the default start point is the beginning of all free space. I then typed **`+8G`** to create an 8 GiB partition. I typed **`L`** to see a list of partition type codes for this first partition, just so you could see them. Beginning with GPT fdisk 1.0.4, `gdisk` enables searching/filtering type codes based on partition descriptions. I typed **`data`** so as to see several "data" partitions. You could instead type **`Linux`** to see Linux type codes, **`Apple`** to see Apple type codes, and so on; or press the Enter key to see all the type codes—but the list is quite long! Note that the search/filter operation is case-sensitive. Since the first partition should be a FAT partition in this example, I opted to give it a 0x0700 type code, which `gdisk` translates into the correct GUID.

Partition locations and sizes can be entered in either absolute or fully relative form. In absolute form, you enter an absolute location or end point in sectors (or KiB, MiB, GiB, or TiB), as in **`2048`** for sector 2048 or **`4M`** for the sector that corresponds to the 4 MiB position on the disk. Relative values are specified by preceding the number with a plus sign (+) or minus sign (-), in which case the value is taken as relative to the start or end, respectively, of the free disk space area noted in the prompt. You can use this feature to create partitions with gaps between them, or to leave just enough space after a partition to create another one of a given size. For instance, you can specify **`+128M`** as the first sector to produce a 128MiB gap between the partition you're creating and the previous one; or you can specify **`-1G`** as the last sector to fill all but 1GiB of the contiguous free space.

With that task done, it's now time to create two more partitions, but these will have to be smaller:

```
Command (? for help): n
Partition number (2-128, default 2):
First sector (34-30719966, default = 16779264) or {+-}size{KMGTP}:
Last sector (16779264-30719966, default = 30717951) or {+-}size{KMGTP}: +4G
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300): L
Type search string, or  to show all codes: Linux
8200 Linux swap                          8300 Linux filesystem
8301 Linux reserved                      8302 Linux /home
8303 Linux x86 root (/)                  8304 Linux x86-64 root (/)
8305 Linux ARM64 root (/)                8306 Linux /srv
8307 Linux ARM32 root (/)                8308 Linux dm-crypt
8309 Linux LUKS                          830a Linux IA-64 root (/)
830b Linux x86 root verity               830c Linux x86-64 root verity
830d Linux ARM32 root verity             830e Linux ARM64 root verity
830f Linux IA-64 root verity             8310 Linux /var
8311 Linux /var/tmp                      8312 Linux user's home
8313 Linux x86 /usr                      8314 Linux x86-64 /usr
8315 Linux ARM32 /usr                    8316 Linux ARM64 /usr
8317 Linux IA-64 /usr                    8318 Linux x86 /usr verity
8319 Linux x86-64 /usr verity            831a Linux ARM32 /usr verity
831b Linux ARM64 /usr verity             831c Linux IA-64 /usr verity
8500 Container Linux /usr                8501 Container Linux resizable rootfs
8502 Container Linux /OEM customization  8503 Container Linux root on RAID
8e00 Linux LVM                           fd00 Linux RAID

Hex code or GUID (L to show codes, Enter = 8300): 8300
Changed type of partition to 'Linux filesystem'

Command (? for help): n
Partition number (3-128, default 3):
First sector (34-30719966, default = 25167872) or {+-}size{KMGTP}:
Last sector (25167872-30719966, default = 30717951) or {+-}size{KMGTP}:
Current type is 8300 (Linux filesystem)
Hex code or GUID (L to show codes, Enter = 8300): L
Type search string, or  to show all codes: FreeBSD
a500 FreeBSD disklabel                   a501 FreeBSD boot
a502 FreeBSD swap                        a503 FreeBSD UFS
a504 FreeBSD ZFS                         a505 FreeBSD Vinum/RAID
a506 FreeBSD nandfs
Hex code or GUID (L to show codes, Enter = 8300): a503
Changed type of partition to 'FreeBSD UFS'
```

The second partition, which will hold a Linux ext4 filesystem, receives the 0x8300 type code. Quite a few specialized Linux type codes have sprung up, most associated with the Discoverable Partitions Specification (DPS), which is intended to enable a Linux OS to figure out where to mount partitions even in the absence of an `/etc/fstab` file, which is the traditional way to handle this task. As of 2022, most distributions still create an `/etc/fstab` file and set the Linux filesystem type code (shown as 0x8300 in `gdisk`) on all Linux filesystem partitions. (LVM, swap, and some other partitions get their appropriate type codes.)

For the final partition, I used the default end partition number rather than enter a value explicitly; however, you may note that the default end point is 30,717,951, whereas the last legal value is 30,719,966 — a discrepancy of 2,015 sectors! This is because, beginning with GPT fdisk 1.0.9, the program attempts to align the end points of partitions much like it aligns their start points, and using the same alignment value (2,048 sectors by default). This rarely affects anything but the final partition, since defining partition sizes with notations like **+4G** (as with the second partition) will usually create a partition with an end-point that's aligned in this way. Aligning the end point is done to keep the LUKS disk encryption software happy; it can misbehave when partition end points are not aligned to its encryption block size (normally 512 bytes or 4096 bytes). If you want to use up to the final legal sector, you can; you must type (or cut-and-paste) the value shown in the prompt and it will be accepted. I gave the partition a 0xa503 type code, which flags it with the GUID for FreeBSD UFS data. Note that you don't need to type **L** to see the type codes; you can skip straight ahead to entering the type code if you already know what it is.

You can now examine the partitions you've created:

```
Command (? for help): p
Disk /dev/sdf: 30720000 sectors, 14.6 GiB
Model: STORE N GO
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): ADBD4AAA-F174-4ADF-A74E-C5BBF0B29526
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 30719966
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048        16779263   8.0 GiB     0700  Microsoft basic data
   2        16779264        25167871   4.0 GiB     8300  Linux filesystem
   3        25167872        30717951   2.6 GiB     A503  FreeBSD UFS
```

The `Name` column holds default names, but you can assign your partitions more descriptive names using the `c` command:

```
Command (? for help): c
Partition number (1-3): 1
Enter name: Shared (FAT) data

Command (? for help): c
Partition number (1-3): 2
Enter name: Linux ext4fs data

Command (? for help): p
Disk /dev/sdf: 30720000 sectors, 14.6 GiB
Model: STORE N GO
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): ADBD4AAA-F174-4ADF-A74E-C5BBF0B29526
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 30719966
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048        16779263   8.0 GiB     0700  Shared (FAT) data
   2        16779264        25167871   4.0 GiB     8300  Linux ext4fs data
   3        25167872        30717951   2.6 GiB     A503  FreeBSD UFS
```

You can now tell by using the `p` command what the partition's purpose is. (GNU Parted and many other tools also display partitions' names.) Partition names may theoretically contain any legal UTF-16 character, at least on Linux, FreeBSD, and macOS. In practice, though, you must have your locale set correctly and you must be using a font that supports the characters you plan to use. Windows is likely to garble non-ASCII names.

Before saving your changes, you may want to verify that there are no glaring problems with the GPT data structures. You can do this with the `v` command:

```
Command (? for help): v

No problems found. 4029 free sectors (2.0 MiB) available in 2
segments, the largest of which is 2015 (1007.5 KiB) in size.
```

In this case, all but 2 MiB of the disk is allocated to partitions and no problems were found. That unused space is at the start and end of the disk and is a result of the default alignment value of 2048. See the Partitioning Advice page for more on why partition alignment is important. If `gdisk` found problems, such as overlapping partitions or mismatched main and backup partition tables, it would report them here. Of course, `gdisk` includes safeguards to ensure that you can't create such problems yourself. The `v` option's sanity checks are intended to help you spot problems that might result from data corruption.

If problems *had* been detected, you might be able to correct them using various options on the recovery & transformation menu, which is a second menu of options available by typing **`r`**:

```
Command (? for help): r

recovery/transformation command (? for help): ?
b       use backup GPT header (rebuilding main)
c       load backup partition table from disk (rebuilding main)
d       use main GPT header (rebuilding backup)
e       load main partition table from disk (rebuilding backup)
f       load MBR and build fresh GPT from it
g       convert GPT into MBR and exit
h       make hybrid MBR
i       show detailed information on a partition
l       load partition data from a backup file
m       return to main menu
o       print protective MBR data
p       print the partition table
q       quit without saving changes
t       transform BSD disklabel partition
v       verify disk
w       write table to disk and exit
x       extra functionality (experts only)
?       print this menu
```

The "Repairing GPT Disks" page describes use of this page's recovery options in more detail.

Transformation options enable you to perform partition table transformations that are not handled automatically when the program starts: convert a GPT to MBR (`g`), create a hybrid MBR (`h`), or convert BSD disklabels in a container partition into GPT partitions (`t`). These options are described in more detail on the Converting to GPT and Hybrid MBR pages of this document.

A third menu, the experts' menu, can be accessed by typing **`x`** in either the main menu or the recovery & transformation menu:

```
recovery/transformation command (? for help): x

Expert command (? for help): ?
a       set attributes
b       byte-swap a partition's name
c       change partition GUID
d       display the sector alignment value
e       relocate backup data structures to the end of the disk
f       randomize disk and partition unique GUIDs
g       change disk GUID
h       recompute CHS values in protective/hybrid MBR
i       show detailed information on a partition
j       move the main partition table
```

```
l       set the sector alignment value
m       return to main menu
n       create a new protective MBR
o       print protective MBR data
p       print the partition table
q       quit without saving changes
r       recovery and transformation options (experts only)
s       resize partition table
t       transpose two partition table entries
u       replicate partition table on new device
v       verify disk
w       write table to disk and exit
z       zap (destroy) GPT data structures and exit
?       print this menu
```

You can do some low-level edits, such as changing partition or disk GUIDs (`c` and `g`, respectively). The `z` option immediately destroys the GPT data structures, which you should do if you want to re-use a GPT disk using some other partitioning scheme. If these structures aren't erased, some partitioning tools can become confused by the apparent presence of two partitioning systems.

Generally speaking, the options on both the recovery & transformation menu and the experts' menu shouldn't be used by anything but GPT experts. Non-experts might be forced to use these menus if a disk is damaged, though. Before taking drastic actions, you should use the `b` main-menu option to create a backup in a file, and store that file on a USB flash drive or some other place that's *not* on the disk you're modifying. That way, you'll be able to recover your orginal configuration if you damage your partitions.

Once you're done making your changes, be they relatively simple partition creations or recovery from disk damage, you can exit from `gdisk`. The `q` command exits without saving your changes, while `w` writes your changes to disk. Both commands exist on both the main and experts' menus.

```
Expert command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT) to /dev/sdf.
The operation has completed successfully.
```

Note for FreeBSD users: A limitation of the FreeBSD version of `gdisk` is that it can't ordinarily write to a disk if any partitions from that disk are mounted. If you want to change your boot disk, you'll need to do so from an emergency system or type **`sysctl kern.geom.debugflags=16`** prior to launching `gdisk`.

You're now back at your shell prompt. Because `gdisk` doesn't create filesystems, you'll need to do that at your shell prompt. In the case of the 3-partition disk in this example, you can create two filesystems in Linux:

```
# mkdosfs /dev/sdf1
# mkfs -t ext4 /dev/sdf2
```

These commands work just as they would on MBR partitions. The example disk's third partition, which is intended for use in FreeBSD, is better formatted in that OS than in Linux.

Caution: If `gdisk` reports that the kernel is still using the old partition table, *do not* try to create filesystems on or otherwise use your new partitions until you've rebooted! Your new filesystem may be written to the wrong partition (possibly one with valuable data) or written to a location on the disk where you won't be able to find it.

Go on to "A `cgdisk` Walkthrough"

Return to "GPT fdisk" main page

If you have problems with or comments about this web page, please e-mail me at rodsmith@rodsbooks.com. Thanks.

Return to my main web page.