About Atom Tables - Win32 apps

By jwmsft

Archived: 2026-04-05 22:49:41 UTC

An *atom table* is a system-defined table that stores strings and corresponding identifiers. An application places a string in an atom table and receives a 16-bit integer, called an *atom*, that can be used to access the string. A string that has been placed in an atom table is called an *atom name*.

The system provides a number of atom tables. Each atom table serves a different purpose. For example, Dynamic Data Exchange (DDE) applications use the global atom table to share item-name and topic-name strings with other applications. Rather than passing actual strings, a DDE application passes global atoms to its partner application. The partner uses the atoms to obtain the strings from the atom table.

Applications can use local atom tables to store their own item-name associations.

The system uses atom tables that are not directly accessible to applications. However, the application uses these atoms when calling a variety of functions. For example, registered clipboard formats are stored in an internal atom table used by the system. An application adds atoms to this atom table using the **RegisterClipboardFormat** function. Also, registered classes are stored in an internal atom table used by the system. An application adds atoms to this atom table using the **RegisterClass** or **RegisterClassEx** function.

The following topics are discussed in this section.

- Global Atom Table
- User Atom Table
- Local Atom Tables
- Atom Types
    - String Atoms
    - Integer Atoms
- Atom Creation and Usage Count
- Atom-Table Queries
- Atom String Formats

The global atom table is available to all applications. When an application places a string in the global atom table, the system generates an atom that is unique throughout the system. Any application that has the atom can obtain the string it identifies by querying the global atom table.

An application that defines a private DDE-data format for sharing data with other applications should place the format name in the global atom table. This technique prevents conflicts with the names of formats defined by the system or by other applications, and makes the identifiers (atoms) for the messages or formats available to the other applications.

In addition to the global atom table, the user atom table is another system atom table that is also shared across all processes. The user atom table is used for a small number of scenarios internal to win32k; for example, windows module names, well known strings in win32k, OLE formats, etc. Although applications do not interact with the user atom table directly, they call several APIs—such as RegisterClass, RegisterWindowMessage, and RegisterClipboardFormat—that add entries to the user atom table. The entries added by `RegisterClass` can be deleted by `UnregisterClass`. However, the entries added by `RegisterWindowMessage` and `RegisterClipboardFormat` do not get deleted until the session ends. If the user atom table has no more space and the string being passed in is not already in the table, the call will fail.

Many critical APIs, including CreateWindow, rely on user atoms. Therefore, space exhaustion in the user atom table will result in serious issues; for example, all applications may fail to launch. Here are some recommendations to ensure your application utilizes atom tables efficiently and preserves the reliability and performance of the application and system:

1. You should limit your app's usage of the user atom table. Storing unique strings using APIs like `RegisterClass`, `RegisterWindowMessage`, or `RegisterClipboardFormat` takes space in the user atom table, which is used globally by other apps to register window classes using strings. If at all possible, you should use AddAtom/DeleteAtom to store strings in a local atom table, or GlobalAddAtom/GlobalDeleteAtom if the atoms are needed cross-process.

2. If there is concern about the application causing user atom table issues, you can investigate the root cause by connecting the kernel debugger and breaking into the process on calls to `UserAddAtomEx` (`bae1 win32kbase!UserAddAtomEx /p <eprocess> "kc10;g"`). Look for `user32!` on the callstack to see which API is being called. The methodology is similar to the global atom table issue detection explained in Identifying Global Atom Table Leaks. Another way to dump the contents of the user atom table is by calling GetClipboardFormatName over the range of possible atoms from 0xC000 to 0xFFFF. If the total atom count steadily goes up while the application is running or does not return to baseline when the app is closed, there is a problem.

An application can use a local atom table to efficiently manage a large number of strings used only within the application. These strings, and the associated atoms, are available only to the application that created the table.

An application requiring the same string in a number of structures can reduce memory usage by using a local atom table. Rather than copying the string into each structure, the application can place the string in the atom table and include the resulting atom in the structures. In this way, a string appears only once in memory but can be used many times in the application.

Applications can also use local atom tables to save time when searching for a particular string. To perform a search, an application need only place the search string in the atom table and compare the resulting atom with the atoms in the relevant structures. Comparing atoms is typically faster than comparing strings.

Atom tables are implemented as hash tables. By default, a local atom table uses 37 buckets for its hash table. However, you can change the number of buckets used by calling the **InitAtomTable** function. If the application calls **InitAtomTable**, however, it must do so before calling any other atom-management functions.

Applications can create two types of atoms: string atoms and integer atoms. The values of integer atoms and string atoms do not overlap, so both types of atoms can be used in the same block of code.

Several functions accept either strings or atoms as parameters. When passing an atom to these functions, an application can use the **MAKEINTATOM** macro to convert the atom into a form that can be used by the function.

The following sections describe atom types.

- String Atoms
- Integer Atoms

When applications pass null-terminated strings to the **GlobalAddAtom**, **AddAtom**, **GlobalFindAtom**, and **FindAtom** functions, they receive *string atoms* (16-bit integers) in return. String atoms have the following properties:

- The values of string atoms are in the range 0xC000 (MAXINTATOM) through 0xFFFF.
- Case is not significant in searches for an atom name in an atom table. Also, the entire string must match in a search operation; no substring matching is performed.
- The string associated with a string atom can be no more than 255 bytes in size. This limitation applies to all atom functions.
- A *reference count* is associated with each atom name. The count is incremented each time the atom name is added to the table and decremented each time the atom name is deleted from it. This prevents different users of the same string atom from destroying each other's atom names. When the reference count for an atom name equals zero, the system removes the atom and the atom name from the table.

Integer atoms differ from string atoms in the following ways:

- The values of integer atoms are in the range 0x0001 through 0xBFFF (**MAXINTATOM**– 1).
- The string representation of an integer atom is *#dddd*, where the values represented by *dddd* are decimal digits. Leading zeros are ignored.
- There is no reference count or storage overhead associated with an integer atom.

An application creates a local atom by calling the **AddAtom** function; it creates a global atom by calling the **GlobalAddAtom** function. Both functions require a pointer to a string. The system searches the appropriate atom table for the string and returns the corresponding atom to the application. In the case of a string atom, if the string already resides in the atom table, the system increments the reference count for the string during this process.

Repeated calls to add the same atom name return the same atom. If the atom name does not exist in the table when **AddAtom** is called, the atom name is added to the table and a new atom is returned. If it is a string atom, its reference count is also set to one.

An application should call the **DeleteAtom** function when it no longer needs to use a local atom; it should call the **GlobalDeleteAtom** function when it no longer needs a global atom. In the case of a string atom, either of these functions reduces the reference count of the corresponding atom by one. When the reference count reaches zero, the system deletes the atom name from the table.

The atom name of a string atom remains in the global atom table as long as its reference count is greater than zero, even after the application that placed it in the table terminates. A local atom table is destroyed when the associated application terminates, regardless of the reference counts of the atoms in the table.

An application can determine whether a particular string is already in an atom table by using the **FindAtom** or **GlobalFindAtom** function. These functions search an atom table for the specified string and, if the string is there, return the corresponding atom.

An application can use the **GetAtomName** or **GlobalGetAtomName** function to retrieve an atom-name string from an atom table, provided the application has the atom corresponding to the string being sought. Both functions copy the atom-name string of the specified atom to a buffer and return the length of the string that was copied. **GetAtomName** retrieves an atom-name string from a local atom table, and **GlobalGetAtomName** retrieves an atom-name string from the global atom table.

The **AddAtom**, **GlobalAddAtom**, **FindAtom**, and **GlobalFindAtom** functions take a pointer to a null-terminated string. An application can specify this pointer in one of the following ways.

String format	Description
#dddd	An integer specified as a decimal string. Used to create or find an integer atom.
string atom name	A string atom name. Used to add a string atom name to an atom table and receive an atom in return.
