AmigaMail

COLLABORATORS

	TITLE : AmigaMail		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 19, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaMail

1.1   IV-101: Introduction to the Datatypes Library

Introduction to the Datatypes Library

by Dan Baker

The latest version of the Amiga operating system, Release 3.0,
includes the new datatypes library. The purpose of the datatypes
library is to provide software tools for handling data in an
object-oriented way. The object-oriented approach means that your
application can work with numerous data file standards without having
to worry about the complex details of each one. Instead you need
only understand the simple conventions of the datatypes library.

The datatypes library is built on Intuition's boopsi facility (boopsi
is an acronym for Basic Object-Oriented Programming System for
Intuition). Although not required, it is very helpful to know a
little about how boopsi works before trying to use the datatypes
library. For information on boopsi, refer to chapter 12 and appendix
B of the Amiga ROM Kernel Reference Manual: Libraries (ISBN
0-201-56774-1). Some familiarity with object-oriented theory and
practice is also helpful, though not required.

Since the datatypes library uses the TagItem structure for passing
parameters to functions, you will have to understand how TagItems
work before you can call the functions in the datatypes library. For
more information on TagItems refer to chapter 37 of the Libraries
manual.

Why Use the Datatypes Library?

One practical benefit of the datatypes library is that it allows you
to quickly add support for IFF data files (this article will show you
how). However, the goals of the datatypes library are much more
ambitious than that. Here's a summary:

    Consistent, simple handling of multiple data standards -
    Most of the details of dealing with the various data

standards are hidden. Once you have learned how to handle
one type of data with the datatypes library, you will find
that the other types are handled in much the same way.

Extensible - You can add your own types of data objects to
those already supported by the datatypes library.
Datatypes has functions that allow other applications to
find out about and work with your data object, without
having to understand the internal details of the data.

Automatic support of IFF and clipboard - The initial
version of the datatypes library (V39) provides support for
8SVX sound data and ILBM graphic data. These are the two
most widely used data file standards on the Amiga.
Developers who want to support these IFF standards no
longer have to become IFF experts. Similarly, the
datatypes library provides a consistent and easy-to-use
interface to the Amiga's clipboard device to encourage data
sharing between applications.

Intuition gadget support - Because the datatypes library is
implemented with boopsi, the data objects it handles can
also be treated as gadgets. Gadget operations can be
performed on data objects within Intuition's task context,
the same as other boopsi gadgets.

Automatic conversion from one format to another - Future
versions of the datatypes library will support other types
of data objects. Conversion from one format to another
will be automatically handled by the library.

Validation - Datatypes lets you easily check if a given
file is a valid instance of one of the data objects it
supports. For example, you can check to see if a file is a
valid ILBM or not.

Classes, Objects and Methods

The jargon used to describe the datatypes library may be a little
confusing if you have never worked with object-oriented systems
before. For instance, the kinds of data supported by the library are
divided into ``classes'' and ``sub-classes''. The term ``class'' is
used here in a familiar way; the members of a class simply have a
common set of properties. The members of a sub-class have all the
properties of the parent class and additional properties specific to
the sub-class. (Each sub-class could be further broken down into
sub-sub-classes and so on.)

Class: Ungulate   Has hooves, can run.
Sub-class: Cow   Has udder, can be milked (also has hooves and can run).
Object: Daisy   An instance of class Cow; can run and can be milked.

An actual instance of a class or sub-class is referred to as an
``object''.  The term ``object'' is appropriate because in general we
want to ignore the details of each individual case and concentrate

instead on what we can do with an object based on its class. In the
example above the Daisy object can run and can be milked. The
operations that can be performed with an object are referred to as
``methods'' and the object is said to ``inherit'' the methods and
other attributes of its parent class (which in turn inherits the
methods and attributes of its parent class, if it has one).

Currently, there are only four object classes (see Table 1) in the
datatypes library. More will be implemented in future versions of
the Amiga OS.

Table 1: Datatypes Library Object Classes in Release 3.0 (V39)

Object Classes and Sub-classes	Autodoc File Showing the Methods Supported	Type of Data Object
Picture class	<picture_dtc.doc>	
ILBM sub-class	<ilbm_dtc.doc>	IFF graphic image file
Sound class	<sound_dtc.doc>	
8SVX sub-class	<8svx_dtc.doc>	IFF audio sample file
Text class	<text_dtc.doc>	
ASCII sub-class	<ASCII_dtc.doc>	ASCII characters
AmigaGuide class	<amigaguide_dtc.doc>	Hypertext databases

The examples programs listed below demonstrate how to perform some
basic ``methods'' on ILBM and 8SVX class objects.

Datatypes Class Attributes

Datatype library classes have other attributes in addition to the
methods (operations) that they support. For each attribute, there is
a corresponding TagItem defined in the datatypes library that you can
use to examine or set that attribute in a particular object

For example, picture objects have a display mode attribute. The tag
that controls this attribute is named PDTA_ModeID and is described in
the Autodoc file picture_dtc.doc. See the Autodoc files for each
class (as shown in Table 1) for a complete list of all class
attributes.

The class attribute descriptions in the include files also have a set
of codes that indicate the ``applicability'' of the attribute. The
codes are as follows:

I - Initialize. You can initialize the attribute when the object is created.
S - Set. You can set the attribute to a new value after the object is created.
G - Get. You can get the value of the attribute after the object is created.
N - Notify. Changing the attribute triggers the object to send notification.
U - Update. Attribute can be set using the object's OM_UPDATE method.

These codes may seem a little mysterious until you have actually
tried using the datatypes library. The N and U codes in particular

are for special applications that want to implement their own object
classes, an advanced topic beyond the scope of this article.

Basic Functions of the Datatypes Library

If all these new concepts seem a little daunting, rest assured; the
datatypes library uses conventional C language function calls to get
the job done. The calls you will be using most often are listed
below. Notice that for each of these basic functions of the V39
datatypes library there is an equivalent boopsi call in the V37
Intuition library.

```
 Function Name           Library             Purpose
-----------------    -----------------    -----------------------------------


NewDTObject()        datatypes.library    Create a datatype object in
NewObject()          intuition.library    memory from a file or clip.

DisposeDTObject()    datatypes.library    Free an object created earlier with
DisposeObject()      intuition.library    NewDTObject() (or NewObject() ).

GetDTAttrs()         datatypes.library    Get attributes of a datatype object.
GetAttr()            intuition.library

SetDTAttrs()         datatypes.library    Set attributes for a datatype object.
SetAttrs()           intuition.library

DoDTMethod()         datatypes.library    Perform the given method (operation)
DoMethod()           amiga.lib            with a datatype object.
```


In a typical application the sequence of calls might be performed
like this:

1. Use NewDTObject() to create an object in memory from given data.
2. Get (or perhaps set) attributes of the object using GetDTAttr()
    (or SetDTAttrs()).
3. Perform ``methods'' (operations) with the object using
    DoDTMethod().
4. Free the object and any memory or other resources it was using
    with the DisposeDTObject() call.

Basic Structures of the Datatypes Library

There are a lot of structures used with datatypes library function
calls; too many to summarize in this article. However, here's a
listing of the relevant include files that contain the structure
definitions of interest to class users.

```
<datatypes/datatypes.h>          Group IDs, error numbers plus library overhead
<datatypes/datatypesclass.h>     Defines datatype methods and associated structures
<datatypes/picture.h>            Structures specific to the picture class
<datatypes/sound.h>              Structures specific to the sound class
<datatypes/text.h>               Structures specific to the text class
<libraries/amigaguide.h>         Structures and methods for AmigaGuide databases
```

```
<intuition/classusr.h>          Defines general boopsi object methods
<intuition/gadgetclass.h>       Defines gadget methods and associated structures
```

The two most important definitions in these include files appear in
<intuition/classusr.h>. The objects used with datatypes library
functions (and the boopsi functions in Intuition) are defined as
follows:

```
    typedef ULONG   Object;      /* abstract handle */
```

Since we want to treat objects as black boxes and don't really care
how they are implemented, this definition is very appropriate. When
a method is performed with an object, the parameter used to identify
the method is a Msg structure defined as follows:

```
    typedef struct {
        ULONG MethodID;
        /* method-specific data goes here */
    }        *Msg;
```

Some methods require more information than just the method
identifier. Such methods have a custom structure defined in the
include files. All method structures, however, begin with a field
that contains the method ID.

A Simple Datatypes Example

The example program listed here should clarify some of the concepts
discussed so far. Suppose you have a communications program and want
to add the capability of playing back a user-specified 8SVX sample
file for the bell sound (Ctrl-G). The program below shows how to
play a sound with the datatypes library.

In this program, objects are of class 8SVX (a sub-class of the sound
datatype). The method performed with the object is named DTM_TRIGGER
(described in the Autodoc file sound_dtc.doc). The DTM_TRIGGER
method (with type set to STM_PLAY) causes a sampled sound to be
played on the Amiga's audio hardware. Since the DTM_TRIGGER method
requires other information in addition to the method ID, a dtTrigger
structure is used. This structure is defined in
<datatypes/datatypesclass.h>.

Note that if the sound datatype is enhanced to support other types of
sound files in a future version of the Amiga OS, the code given here
will automatically support the new type. This example expects the
file name and path to a sound file.

 dt.c

In addition to playing back a sampled sound, the datatypes library
allows sound objects to become gadgets (the library includes default
imagery for a sound gadget). Since all datatypes object classes are
implemented as a sub-class of the boopsi ''gadget'' class, they all
support the methods of gadget objects as described in the boopsi
chapter of the Libraries manual.

A Picture Class Example

Here is a second, more complex example showing how to use all the
datatypes library functions described so far. In this example, the
objects used are of class ILBM, a sub-class of picture.

Two methods will be performed with the object, DTM_PROCLAYOUT and
DTM_FRAMEBOX. Both these methods have associated structures
(gpLayout and dtFrameBox respectively). DTM_PROCLAYOUT makes the
object available within the context of your application task (as
opposed to Intuition's). DTM_FRAMEBOX queries the display
environment required by the picture.

Other attributes of the picture are obtained with a call to
GetDTAttrs() and then a matching Intuition screen is created and the
ILBM object is displayed. This example expects the file and path
name of a picture file.

 dtpic.c

As with 8SVX objects, the datatypes library allows ILBM objects to be
treated as gadgets. Remember that all datatypes object classes a
sub-class of the boopsi ''gadget'' class and therefore support the
gadget methods described in the boopsi chapter of the Amiga ROM
Kernel Reference Manual: Libraries.