$Revision: 1.4 $ $Author: death $ $Date: 93/04/04 23:45:14 $

ResultObject

INHERITS FROM Object

DECLARED IN ResultObject.h

CLASS DESCRIPTION

ResultObject derives, logically, from the now obsolete classes ErrorInfo and Reply. The goal of those two classes was to provide a standard, and hopefully frequently used, way to get the results of a method call back to the caller. In the case of ErrorInfo, this meant to get error codes and descriptive error strings returned. Reply was a sublcass of ErrorInfo, and thus provided the previously mentioned error code returns, plus ways to get data returned as well. The idea behind them was that objects would return, say, a Reply object with complete error information as well as any data the caller had requested. This made it easier for an object to provde needed data as well as error information without relying on things like external global variables for error codes, or what have you. Put another way, one could now *always* provide error codes when returning data without any additional work, thus potentially providing more information to the caller. Being objects, there was also room to subclass them and provide even more specialized and detailed error codes, or to return multiple pieces of non error-related data at once. These class proved, however, to be more cumbersome than they were worth. In all too many of the cases where one was returned, the programmer didn't want to deal with checking the error code. This resulted in cumbersome effortrts

to dispose of the result object as soon as it was returned. Thus, in a majority of the cases, an object was created, initalized, filled, returned, and then deallocated, without even referencing any of the information in the returned object. The result was undoubtedly slower code that was considerably harder to read.

The ResultObject class seeks to provide the same services that Reply did, without its cumbersome aspects. Rather than having new classes return a Reply or ErrorInfo object, the class tree is set up so that new classes decend from ResultObject rather than Object. ResultObject provides the internal storage to remember error information for the last method called, as well as storage of data. Classes decended from ResultObject can behave normally, and return self or a single data result. Additionally, methods can store error information, and additional data in the object, and allow the caller to retrieve it at its conveniance.

ResultObject allows one to store an error code and a textual error string, as well as several pieces of data (up to five, presently) that might be requested by a caller. It recognizes only 7 data types presently: Character, Cstring, Integer, PositiveInteger, Object, Pointer and Boolean. Support for ByteString, float types, and others may (or may not) be forthcomming. It also provides one standard error code: ERR_OK, which corresponds to the number 0, and should always be used when no notable errors or problems occured.

ResultObject has a large set of straightforward methods to set and retrieve result data, as well as error information. See descriptions below for specific details.

The methods can be divided into two groups: those that store data, and those that retrieve it. The methods that store data should be called only by the class decended from ResultObject. These methods allow one to store an error code and an error string, as well as store items of particular data types (e.g. a method to store an Integer). There are two sub-sets of methods for storing the data. One set, the **Store¼** methods, implicitly store their data in the first storage area. The **Put¼:Into:** methods store their data into any of the storage areas. Additionally, there is one method that clears all waiting stored data.

The methods to retrieving data are much the same as the storage ones. There is a method to

retrieve an error code, and one to retrieve the error string. There are a set of Get¼ methods that always retrieve the data item in the first slot, and a set og Get¼From: methods to retrieve from arbitrary return positions.

One can check if an error occurred when storing or retrieving a data value by using the **Store** and **Get MyError** methods.

Note that when passed an Integer, Character, PositiveInteger, or Boolean, the object will always make a copy of the data internally, and copies will always be returned. Objects are always stored and returned as pointers; their data is not copied. For Cstrings, error strings, and Pointers, the default **Store¼** methods will store only references, but there are **Copy¼** methods provided which will copy the referenced data (Note: if the pointer points to a block that in turn points to other blocks of memory, those secondary blocks will not be copied). When retrieved, Pointers are always returned merely as poiners, not as copies of the data the poiners point to, while CStrings and error strings are always copied before being returned.

If an illegal storage area reference is given with a Put method, the data will be ignored, and no errors will be generated. If one tries to retrieve a data type that isn't in a storage area, or to request data from a storage area that doesn't exist, you'll get a null value returned; the error codes will be modified to reflect this.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in ResultObject	array of StorageArea	ResultVals;

ResultVals

Used to store all data values to be returned, as well as the internal error code, and the error code and text the subclassing object is returning.

METHOD TYPES

Creating/Initializing, and Freeing	- init - free
Resetting values	- ResetResults
Setting Error values	- StoreErrorCode:AndText: - StoreErrorCode:AndCopyOfText: - StoreMyError:
Retrieving Error values	- GetErrorCode - GetErrorText - GetMyError
Setting values	- PutInteger:Into: - PutPositiveInteger:Into: - PutCharacter:Into: - PutCstring:Into: - PutObject:Into: - PutPointer:Into: - PutBoolean:Into: - CopyCString:Into: - CopyPointer:WithLength:Into: - StoreInteger: - StorePositiveInteger: - StoreCharacter: - StoreCstring: - StoreObject: - StorePointer: - StoreBoolean: - CopyCString: - CopyCString:Into:

Retrieving values

- CopyPointer:WithLength:
- CopyPointer:WithLength:Into:
- PutData:WithType:theTypeInto:DoIOwn:
- GetIntegerFrom:
- GetPositiveIntegerFrom:
- GetCharacterFrom:
- GetCstringFrom:
- GetObjectFrom:
- GetPointerFrom:
- GetBooleanFrom:
- GetInteger
- GetPositiveInteger
- GetCharacter
- GetCstring
- GetObject
- GetPointer
- GetBoolean
- GetDataWithType:From:

CLASS METHODS

none

INSTANCE METHODS

CopyCString:

- (Object) **CopyCString:** (CString) *data*

Makes a copy of the specified CString. This is in constrast to Store**CString:** which only stores a reference to its CString. Returns self.

CopyCString:Into:

- (Object) **CopyCString:** (CString) *data* **Into:** (Integer) *reference*

Makes a copy of the specified CString, and stores it in the specified storage area. This is in constrast to **StoreCString:** which only stores a reference to its CString. Returns self.

CopyPointer:

- (Object) **CopyPointer:** (Pointer) *data* **WithLength:** (PositiveInteger) *length*

This makes a copy of the specified pointer's data, and stores that. This is in contrast to **StorePointer:** which only stores a reference to the data. Returns self.

CopyPointer:WithLength:Into:

- (Object) **CopyPointer:** (Pointer) *data* **WithLength:** (PositiveInteger) *length* **Into:** (Integer) *reference*

This makes a copy of the specified pointer's data, and stores a reference to the copy in the storage area referenced by *reference*. This is in contrast to **StorePointer:** which only stores a reference to the data. Returns self.

init

- (Object) **init**

Unsurprisingly, this initalizes a ResultObject object. It clears the internal variables to default values, and returns self (all this after having initalized its parent, of course).

free

- **free**

Disposes of the instance values, freeing those it owns, and then frees the object.

GetBoolean

- (Boolean) **GetBoolean**

Returns a copy of the Boolean in the first storage area.

GetBooleanFrom:

- (Boolean) **GetBooleanFrom:** (Integer) *reference*

Returns a copy of the Boolean in the storage area specified by reference (use the defined constants from below).

GetCharacter

- (Character) **GetCharacter**

Returns a copy of the character in the first storage area.

GetCharacterFrom:

- (Character) **GetCharacterFrom:** (Integer) *reference*

Returns a copy of the character in the storage area specified by reference (use the defined constants from below).

GetCString

- (CString) **GetCString**

Returns a copy of the character in the first storage area.

GetCStringFrom:

- (GetCString) **GetCStringFrom:** (Integer) *reference*

Returns a copy of the Cstring in the storage area specified by reference (use the defined constants from below).

GetDataWithType:From:

- (Genericype) **GetDataWithType:** (Integer) *thetype* **From:** (Integer) *reference*

Retrieves a data item of type theType from the storage pointed to by reference. Return the type as a generic type. If there is an error, store an error code via **StoreMyError:.**

GetErrorCode

- (Integer) **GetErrorCode**

Returns a copy of the error code that the object is currently storing

GetErrorText

- (CString) **GetErrorText**

Returns a copy of the Cstring in the error text storage area. Note that this is a full *copy* of the string. One can not obtain a mere reference to it. One is responsibe for freeing the text using the usual **free**() call.

GetInteger

- (Integer) **GetInteger**

Returns a copy of the Integer in the first storage area.

GetIntegerFrom:

- (Integer) **GetIntegerFrom:** (Integer) *reference*

Returns a copy of the Integer in the storage area specified by reference (use the defined constants from below).

GetMyError

- (Integer) **GetMyError**

Returns a copy of the error code that was stored while last setting or getting a result.

GetObject

- (Object) **GetObject**

Returns a copy of the Object in the first storage area.

GetObjectFrom:

- (Object) **GetObjectFrom:** (Object) *reference*

Returns a copy of the Object in the storage area specified by reference (use the defined constants from below).

GetPointer

- (Pointer) **GetPointer**

Returns a copy of the Pointer in the first storage area.

GetPointerFrom:

- (Pointer) **GetPointerFrom:** (Pointer) *reference*

Returns a copy of the Pointer in the storage area specified by reference (use the defined constants from below).

GetPositiveInteger

- (PositiveInteger) **GetPositiveInteger**

  Returns a copy of the PositiveInteger in the first storage area.

GetPositiveIntegerFrom:

- (PositiveInteger) **GetPositiveIntegerFrom:** (PositiveInteger) *reference*

  Returns a copy of the PositiveInteger in the storage area specified by reference (use the defined constants from below).

PutBoolean:Into:

- (Object) **PutBoolean:** (Boolean) *data* **Into:** (Integer) *reference*

  Stores a copy of the Boolean in the storage area specified by reference (use the defined constants from below). Returns self.

PutCharacter:Into:

- (Object) **PutCharacter:** (Character) *data* **Into:** (Integer) *reference*

  Store a copy of the character in the storage area specified by reference (use the defined constants from below). Returns self.

PutCString:Into:

- (Object) **PutCString:** (Cstring) *data* **Into:** (Integer) *reference*

  Stores a **reference** (pointer) to the CString in the storage area specified by reference (use the defined constants from below). Returns self.

PutData:WithType:Into:DoIOwn:

- (Object) **PutData:** (GenericType) *theData* **WithType:** (Integer) *theType* **Into:** (Integer) *reference*
  DoIOwn: (Boolean) *ownit*

Stores the specified data with the specified type into the specified storage area (and indicate whether we are just pointing tothe data, or if we have our own copy), if possible. If it has trouble, an error is stored via StoreMyError:. Returns self.

PutInteger:Into:

- (Object) **PutInteger:** (Integer) *data* **Into:** (Integer) *reference*

  Stores a copy of the Integer in the storage area specified by reference (use the defined constants from below). Returns self.

PutObject:Into:

- (Object) **PutObject:** (Object) *data* **Into:** (Integer) *reference*

  Stores a **reference** (pointer) to the Object in the storage area specified by reference (use the defined constants from below). Returns self.

PutPointer:Into:

- (Object) **PutPointer:** (Pointer) *data* **Into:** (Integer) *reference*

  Stores a **reference** (pointer) to the Pointer in the storage area specified by reference (use the defined constants from below). Returns self.

PutPositiveInteger:Into:

- (Object) **PutPositiveInteger:** (PositiveInteger) *data* **Into:** (Integer) *reference*

  Stores a copy of the PositiveInteger in the storage area specified by reference (use the defined constants from below). Returns self.

ResetResults

- (Object) **ResetResults**

This clears the error code, error text, and all data items in the storage area. It returns self.

StoreErrorCode:AndText:

- (Object) **StoreErrorCode:** (Integer) *code* **AndText:** (CString) *text*

Stores a copy of the error code, a **reference** to the error text, and returns self.

StoreErrorText:AndCopyOfText:

- (Object) **StoreErrorCode:** (Integer) *code* **AndCopyOfText:** (CString) *text*

Stores a copy of the error code, and a **copy** of the error text, and returns self.

StoreBoolean:

- (Object) **StoreBoolean:** (Boolean) *data*

Stores a copy of the Boolean in the first storage area.

StoreCharacter:

- (Object) **StoreCharacter:** (Character) *data*

Stores a copy of the character in the first storage area.

StoreCString:

- (Object) **StoreCString:** (CString) *data*

Stores a **reference** (pointer) to the character in the first storage area.

StoreInteger:

- (Object) **StoreInteger:** (Integer) *data*

Stores a copy of the Integer in the first storage area.

StoreMyError:

- (Object) **StoreMyError:** (Integer) *errorcode*

Stores a copy of the specified error in thestorage area for the internal error.

StoreObject:

- (Object) **StoreObject:** (Object) *data*

Stores a **reference** (pointer) to the Object in the first storage area.

StorePointer:

- (Object) **StorePointer:** (Pointer) *data*

Stores a **reference** (pointer) to the Pointer in the first storage area.

StorePositiveInteger:

- (Object) **StorePositiveInteger:** (PositiveInteger) *data*

Stores a copy of the PositiveInteger in the first storage area.

BUGS

There is no way provided to make an actual copy of an object, rather than just copy a reference.
Dealing with its own errors has not been examined closely, and they may be incomplete.
We are also happy and content to just overwrite any data items already there. (this may be a feature)

ENHANCEMENT IDEAS

Everything should be collapsed into object types!
The distinction between copying the actual data, and only copying a reference seems a bit

cumbersome, and some way to flatten this should be provided eventually.
Perhaps add a set of **Store¼** methods that would automatically store in the next available storage area?
Make the storage area set a dynamically sized linked list, so that one needn't worry about running out of storage areas

CONSTANTS AND DEFINED TYPES

```
/* Types used by ResultObjects */
#define ERR_OK                  0
#define ERR_PEACHY              0
#define ERR_ALLISWELL           0
#define ERR_GROOVY              0

#define ERR_NOSUCHAREA          -1234
#define ERR_NOSUCHTYPE           -1235
#define ERR_CANTSTORE           -1236

#define FIRST_RESULT            3
#define SECOND_RESULT           4
#define THIRD_RESULT            5
#define FOURTH_RESULT           6
#define FIFTH_RESULT            7

#define MYERROR_RESULT          0
#define ERRORCODE_RESULT        1
#define ERRORTEXT_RESULT        2
```

MODIFICATION HISTORY

$Log: ResultObject.rtf,v $Revision 1.4 93/04/04 23:45:14 deathSun Apr 4 23:45:14 PDT 1993Revision 1.3 93/01/10 15:08:46 deathSun Jan 10 15:08:46 PST 1993Revision 1.2 92/07/26 13:59:21 deathUpdate of the result object... (prob no changes here)Revision 1.1

92/04/27 20:51:47 deathInitial revision
Revision 0.0 92/02/09 14:01:00 death
    Use this sample format as you check out future revisons...