The API is dead...

...long live the API

A little ooRexx API history

- Originally designed for the 16-bit Classic Rexx release in OS/2 1.2
  - ints were 16-bit, longs were 32-bit
  - longs were used universally for string lengths
  - short ints were used in many places
- Largely a translation of VM/CMS APIs into C.
- For compatibility, the API was kept largely unchanged when migrated to 32-bit OS/2 2.0

History continued

- As designed, this API was never intended to be a cross-platform API
  - Somehow, it became a de facto standard
- Designed for the “everything in Rexx is a string” world, it served well for many years.

Fast forward 18(!) years

- The string-only style places serious limitations on what native-code extensions can do
  - Not everything is easily mapped to string values
  - Objects get translated to their “string value”.
- Choices made in 1988 for data types don't translate well to 64-bit platforms
  - Non-portable long ints
  - Pointer sizes no longer match int sizes

The 64-bit cleanup effort

- Need for new object APIs was long recognized
  - Originally intended as a follow-on effort after the 64-bit version.
- The 64-bit type compatibility issues forced a decision
  - Implement a 64-bit “clean” version of string interfaces
  - or, skip ahead to the longer-range solution
- We decided to “go for it”.

Some notes about compatibility

- On 32-bit systems, old native libraries will continue to function
- On 64-bit systems, libraries will need to be converted to the new APIs
  - Either new type-clean versions of the legacy APIs, or
  - The new object API set.

Type-clean legacy APIs

- Similar to existing APIs
  - Library function names have changed (ooRexxVariablePool vs. RexxVariablePool)
  - Function arguments and structure fields redefined to use abstract types:

```
typedef struct _VariableRequest {     /* shvb */
    struct _VariableRequest *shvnext; /* pointer to the next block   */
    RxString            shvname;       /* Pointer to the name buffer  */
    RxString            shvvalue;      /* Pointer to the value buffer */
    RexxStringLength   shvnamelen;    /* Length of the name value    */
    RexxStringLength   shvvaluelen;   /* Length of the fetch value   */
    RexxNumber         shvcode;       /* Function code for this block*/
    RexxNumber         shvret;        /* Individual Return Code Flags*/
}   VariableRequest;
```

Object APIs

- Modeled after existing API styles
  - Java JNI
  - PHP Zend
- Rather than call RexxStart to run program, an interpreter instance is created
  - Environment persists between calls
  - Able to hold references to ooRexx objects between program calls
  - Similar to Java JNI_CreateJavaVM() function
  - Additional threads can be attached to an instance
  - Exit handlers apply to an interpreter instance

Packages

- Rexx function packages are self-describing extension libraries that can declare a set of registered functions and/or native methods
- Loaded automatically by the ::package directive
  - library must be available and loadable for the program to run

Table declared routines

```
   ooRexxFunctionEntry rxsock_functions[] ={
    REXX_TYPED_FUNCTION( SockDropFuncs      , SockDropFuncs            )
    ...
    REXX_TYPED_FUNCTION( SockVersion        , SockVersion              )
};

ooRexxPackageEntry rxsock_package_entry ={
    STANDARD_PACKAGE_HEADER
    "RXSOCK",                          // name of the package
    "1.3",                             // package information
    rxsock_functions,                  // the exported functions
    NULL                               // no methods in this package
};

// package loading stub.
OOREXX_GET_PACKAGE(rxsock);
```

Typed function declarations

- In addition to the legacy string-based functions, you can create functions with type declarations
  - ooRexx does data-type conversions on both arguments and return value
  - Performs checks for required arguments
  - Provides “reasonable” defaults for omitted optional arguments
  - Frequently MUCH easier to implement stub functions

Compare this...

```
LONG APIENTRY SysFileCopy(
  PSZ       name,                      /* Function name             */
  LONG      numargs,                   /* Number of arguments       */
  RXSTRING  args[],                    /* Argument array            */
  PSZ       queuename,                 /* Current queue             */
  PRXSTRING retstr )                   /* Return RXSTRING           */
{

  if (numargs != 2)                    /* we need two arguments     */
    return INVALID_ROUTINE;            /* raise an error            */

                                       /* copy the file             */
  if (!CopyFile(args[0].strptr, args[1].strptr, 0))
      RETVAL(GetLastError())           /* pass back return code     */
   else
      RETVAL(0)
}
```

...to this

```
RexxFunction2(int, SysFileCopy, CSTRING, fromFile, CSTRING, toFile)
{
    return CopyFile(fromFile, toFile, 0) ? 0 : GetLastError();
}
```

Many different types available

- Rexx object
- int
- Rexx String object
- CSTRING
- Rexx Array
- Rexx Stem
- double
- float
- various int sizes
- boolean

Context...

the difference between roadkill and somebody's lunch

API context pointers

- Context pointers are pointer vectors providing access to API functions
  - Similar to the Java JNI env pointer
- Multiple context types, which expose different functions
  - Thread context – implements access to object capabilities
  - Function context – provides thread context functions plus access to function environment
  - Method context – thread context functions plus access to method/object env.
  - Exit context – thread context plus exit environment.

Thread context

- Available:
  - After creating Rexx interpreter instance
  - After attaching a thread to an instance
  - Passed to function, method, and exit calls

Thread context functions

- Object reference handling
- Object method invocation
- Data conversion functions
- Utility functions for common object manipulation (Array, Stem, String, Directory, etc.)
- Condition access
- Environment access
- Method loading/resolution functions
- Useful predefined objects (.nil, .true, .false)

Some examples

```
return context->NewStringFromAsciiz(temp);

context->SetStemArrayElement(stem, count, context->NewString(ibuf, vlen));

return context->NullString();

return context->NumberToObject(ERROR_NOMEM);

char *classStr = context->ObjectToStringValue(classArg);
```

Function context

- Passed to native function calls
- Thread context functions plus
  - Argument access
  - Caller context variable access
  - Numeric setting access

Some Examples

```
RexxSupplierObject vars = context->GetAllContextVariables();

context->InvalidRoutine();

context->SetContextVariable(“RC”, context->NumberToObject(rc));
```

Method context

- Passed to all native method calls
- Thread context functions plus
  - Argument access
  - Object variable access
  - Self, super access
  - Super class message sends
  - Guard functions
  - Context class resolution

Exit context

- Passed to all native method calls
- Thread context functions plus
  - Context variable access

Questions?