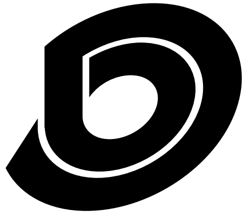


Revision: 5

B & E Software GmbH

1997-04-28

RagTime External Functions Software Development Kit

Table of Contents

1 External Functions Overview 3

2 Creating External Functions..4
2.1 Resource Setup 4
2.1.1 The 'BXDe' (External Function Description) Resource 4
2.1.2 The 'BXCa' (External Function Category) Resource 8
2.2 Coding an External Function..9
2.2.1 Overview 9
2.2.2 Entry Point..9
2.2.3 The RTXRecord 9
2.2.4 Passing parameters and results 11
2.2.5 Error Reporting..13
2.2.6 External Function Code Structure 14

3 Back Calls 17
3.1 GetLastBackCall..17
3.2 GetNextParam 18
3.3 GetCell..18
3.4 GetNextCell..18
3.5 SetCell 18
3.6 FindSheet..19
3.7 CalcAFunction..19
3.8 Text2Value 19
3.9 PrivateToDeskScrap..19
3.10 CalcNow 20
3.11 MailMerge 20
3.12 SaveDocument..21
3.13 PrintDocument..21
3.14 GetDocumentName 21
3.15 GetDataItem & DeleteDataItem 22
3.16 DocumentChanged 22
3.17 OpenPreferences..22
3.18 UpdateWindow..22
3.19 GetCellAlways 22
3.20 GetNextCellAlways..23

4 Tips & Caveats 24
4.1 Open Resource Files..24
4.1.1 Saving and Restoring..24
4.2 Reentrant Code 24
4.3 Memory Usage 24
4.4 The SANE Environment..25

5 Support & Marketing..26

1 External Functions Overview

In RagTime's function list you can sort all functions by collection or category. If you choose "Function Collections", you see the groups of functions which are included in the version that you have installed. Some of these groups are built-in, others are external.

External functions are those which are programed by developers and made available to an existing version of RagTime. An external function group is a set of external functions which share the same resources and code. This makes managing, installing and removing a group of functions easier. In addition, related functions can use common code without it being duplicated for each function.

Beginning with RagTime version 3.2, external functions can be contained in add-on files which are placed in the B & E Folder, or anywhere in the add-on search path (see your RagTime documentation).

External functions consist of resources which are made available to the RagTime application file. The instructions about the external function's name, what arguments it accepts, what spoken language it is written for, what its version number is and what services it needs from RagTime itself are contained in an external function description resource, which has the resource type '**BXDe**'.

The actual code that the external function developer writes is compiled into an external function code resource, with a resource type of **'BXCo'.** This resource is comparable to stand-alone code resources such as HyperCard XCMDs, which have one entry point that the host program can jump into.

Additionally, other resources may be used by the external function group. For example, if a developer wants to put up a dialog box giving the user a warning, a '**DLOG**'/'**DITL**' combination could be used.

Starting with RagTime 3.2, RagTime's Function List dialog allows you to limit the display of functions to a particular subset of all functions available through the use of categories. A category is a particular set of related functions, for example, all statistical functions. You can add your external function group to any existing category, or define a new category, through the use of an External Function Category Resource, which has the resource type **'BXCa'**.

These three resource types, '**BXDe**', '**BXCo**' and '**BXCa**', must always have the same resource ID number for a particular group of external functions.

This Software Development Kit (SDK) wants to show you how to create an external function group which then can be used with RagTime version 3.2 and all versions of RagTime 4. With RagTime 4 the add-on file "RagTime 3 Function Adapter" is needed to make an external function group available.

This SDK consists of this description, the interface files and a sample project for Metroworks CodeWarrior, created with CW11. For a more detailed description how to start with your own external functions based on the sample project see section 2.2 "Coding an External Function" on page 9.

2 Creating External Functions

2.1 Resource Setup

The easiest way to create the `'BXDe'` and `'BXCa'` resources for an external function group is to write a text file that can be run through the Rez tool. If the Rez tool is used, the developer must include at least the following three statements at the beginning of the text file:

```
#INCLUDE "Types.r"
#INCLUDE "SysTypes.r"
#INCLUDE "RTXFuTypes.r"
```

These included files define the rules for creating the `'BXDe'` and `'BXCa'` resources. The `Types.r` and `SysTypes.r` files should be included with Metroworks, while `RTXFuTypes.r` can be found in the RagTime external function Developer's Kit.

2.1.1 The 'BXDe' (External Function Description) Resource

This resource gives information to the RagTime external function adapter about what this external function group can do, along with what services it needs. The type definition of the `'BXDe'` resource (defined in `RTXFuTypes.r`) is included in this section, as well as an explanation of each of its fields:

```
type 'BXDe' {
     integer    kCurrentVersion = 5;   /* For RagTime 4 and RagTime 3.2 */
     longint = 0;                      /* Place holder for nextList.    */
     longint = 0;                      /* Place holder for codeHdl.     */
     longint = 0;                      /* Place holder for refCon.      */
```

`kCurrentVersion` determines that the external function group works with all RagTime versions since RagTime 3.2. There are other values for older RagTime versions. Certain back calls and services are only available in a particular version of RagTime, so your external function group must specify which version of RagTime it needs in order to operate correctly. A list of which back calls are available for which versions of RagTime may be found at the beginning of section 3 "Back Calls" on page 17.

The three `longint`s following the version definition are placeholders for data that RagTime puts in memory when the program is running. **Do not** address them when defining the `'BXDe'` resource.

```
     unsigned bitstring[13] = 0;           /* Unused flags. */
     boolean    noMailMerge, hasMailMerge;
     boolean    noMessages, wantsMessages;
     boolean    noTime, needsTime;
```

The `unsigned bitstring` contains flags which are not being used in this version of the interface. They are automatically set to zero. **Do not** address them when defining the `'BXDe'` resource.

`noMailMerge, hasMailMerge` determines whether this function group contains functions which need to be calculated during printing. This facilitates the creation of a custom Mailmerge function.

`noMessages, wantsMessages` indicates to RagTime whether it must inform this external function group when the current document is being opened, closed, activated or deactivated. The external function group can then respond to these situations appropriately.

`noTime, needsTime` signals whether this external function group needs to be called periodically. For example, a function that needs to poll a database periodically would request this service. If periodic calls are needed, the `needsTime` flag must be set. Otherwise, `noTime` should be used.

```
     pstring[79];     /* Name of external function group.     */
     integer;         /* Version number.                      */
     integer    Region;
```

```
    longint    noTime, asOftenAsPossible = 0;
```

`pstring[79]` must be filled in with the name of this group of external functions. This name is used in the External Function Installer to identify the external function group. It is also shown in the Function List dialog box when one of the functions from the external function group is chosen.

The following `integer` identifies the version number of this group of external functions. For example, the decimal value 101 indicates version 1.01. This version number is displayed in the External Function Installer when the external function group is being installed. It is also shown in the Function List dialog box when one of the functions from the external function group is chosen.

`Region` indicates the country or language for which this set of external functions is localized, for example, `verUS` for an American English version. The possible values may be found in the file `SysTypes.r`.

`noTime, asOftenAsPossible` depends on the definition of the `noTime, needsTime` flag defined earlier. If `noTime` is set, then this `longint` is ignored and should also be set to `noTime` to avoid confusion. However, if periodic calls are needed, the `needsTime` flag, must have previously been set, and this field must indicate how often the periodic calls should occur. The value of this field is expressed in sixtieths of a second (Macintosh ticks), so that if an external function needs to be called every 2 seconds, this field would be set to 120. If `asOftenAsPossible` is set, then RagTime calls the external function at every possible opportunity. The time between two calls may actually be longer than the specified time, depending on system load.

```
    integer = 0;          /* Place holder for 'resID'.      */
    integer = 0;          /* Place holder for 'groupStart'. */
    longint = 0;          /* Spare.'                        */
```

The two `integer`s above are place holders for data that RagTime puts in memory when the program is running, and the `longint` is not currently used. **Do not** address these fields when defining the `'BXDe'` resource.

```
    integer = $$Countof (RCXFuDescriptArray);
    array RCXFuDescriptArray {              /* Functions.    */
          byte = 0;                         /* Spare.        */
          unsigned bitstring[6] = 0;        /* Unused flags. */
          boolean noMailMerge, isMailMerge;
          boolean noRefNeeded, needsRef;
```

This part of the `'BXDe'` resource defines an array which sets up the individual functions in the external function group. The `integer = $$Countof (RCXFuDescriptArray)` is a field which holds the number of elements in the array, which is calculated by the Rez tool. The `byte` and `unsigned bitstring[6]` at the beginning of each function definition are automatically set to zero. **Do not** address these two values when defining the `'BXDe'` resource.

`noMailMerge, isMailMerge` depends on the definition of the `noMailMerge, hasMailMerge` flag defined earlier. If the `noMailMerge` was previously set, then the value in this field is ignored and should also be set to `noMailMerge` to avoid confusion. However, if this function needs to be calculated when RagTime is printing, the `hasMailMerge flag` must have previously been set, `isMailMerge` must be set here, and the `noRefNeeded, needsRef` flag described in the next paragraph must be set to `needsRef`.

`noRefNeeded, needsRef` indicates whether the `CalcAFunction` back call, documented in the section “Back Calls”, can be used to trigger recalculation of this function. `needsRef` must be set if the `CalcAFunction` back call should trigger recalculation of this function; otherwise `noRefNeeded` should be used.

```
    wide array [6] {
          boolean    notRequested, requested;
          unsigned bitstring[2]  empty,
                                 value,
                                 list,
                                 range;
          boolean recalc, noRecalc;
          unsigned hex bitstring[12];
    };
```

```
pstring[31];     /* Internal name of the function. */
```

The array definition above refers to the parameters that are needed by this external function. As the definition of the array indicates, the maximum number of parameters for an external function is limited to six.

`notRequested,requested` defines whether or not the external function must have a parameter in this position. If `notRequested` is set, then this is either an optional parameter or one that is not used. If the `requested` flag is set, then RagTime does not allow the user to complete entry of the formula until the expected parameter is inserted.

`empty, value, list, range` defines what type of parameter the external function expects in this position. RagTime makes sure that the parameters specified by the user conform to this definition when the function is edited. The `empty` setting used in conjunction with the `notRequested` flag indicates that no parameter is expected in this position. The `value` setting means that some type of value is expected: text, a number, a date and so on. The `list` setting indicates that one or more values, expressions or ranges, separated by semicolons, is expected. There can be only one `list` parameter per function, and it must be placed after all other `requested` parameters. The `range` setting means that this parameter expects a cell range within a spreadsheet, for instance A1:B5.

A function is always recalculated after the user edits the function itself, but for parameters that expect a range, the **`recalc, noRecalc`** flag is used to indicate if recalculation must be performed after the values lying in the specified range are changed. This flag is ignored if the type of parameter is not a range, but should be set to `noRecalc` in that case to avoid confusion.

If you have specified `empty` or `range` as the type of parameter, then the `unsigned hex bitstring[12]` after the `recalc, noRecalc` flag is ignored and should be set to `valNil`. However, if the function expects a `value` or a `list`, then you must use the constants defined in `RTFuTypes.r` to indicate what type of value is expected.

RagTime's spreadsheet makes sure the user enters the correct number of parameters. After the user completes entry of the values, RagTime checks to see if they agree with the expected values and, if a value does not agree, tries to convert it to the required type. If conversion to the required type is not possible, RagTime returns the error "VALUE!" without ever calling the external function group.

The possible value types are listed below:

```
#define valNil          $1
#define valNum          $2
#define valBool         $4
#define valDate         $8
#define valText         $10
#define valErr          $20
#define valIntNum       $80
#define valRTF          $400
#define valAll          $04BF
```

`valNil` must be used for `notRequested, empty` parameters. It can also be used in addition to another type if you want to differentiate between the default value returned from an empty cell and a cell which actually contains the default value. **`valNum`** indicates a SANE extended number, **`valBool`** means a Boolean value, **`valDate`** is a SANE extended value in the form of days since midnight, January 1, 1904, **`valText`** indicates unformatted Macintosh ASCII text, **`valErr`** indicates an error value, **`valIntNum`** is a 32-bit signed integer (Pascal type `LongInt`), **`valRTF`** indicates formatted text written in RTF file format, and **`valAll`** indicates that all possible parameter types should be passed back to the external function.

These flags may also be combined. For example, if you had a parameter that would accept either a number or a date, you would set up your parameter like this:

```
requested, value, noRecalc, valNum + valDate,
```

Any of the six parameters that are not needed by the function must be defined as follows:

```
notRequested, empty, noRecalc, valNil,
```

Functions can also be created which expect a variable number of parameters. This can be done by using a list parameter, as described above, or by creating one or more specific parameters of type `value` or `range` which you mark as notRequested, like this:

```
notRequested, value, noRecalc, valNum,
notRequested, range, recalc, valNil,
```

Notice that, by their nature, `list` and optional parameters can only be used after all other `requested` parameters and they can not be used together in the same function (with the exception of parameters which are defined as both `notRequested` and `list`).

Finally, the two `pstring`s defined at the end of the `'BXDe'` resource indicate the name that the user is be shown and the internal name of the function itself, respectively. The user name should be a language-localized string which gives some indication to the user of what the external function does. The internal name is a unique identifier and should not be changed when an external function group is localized to a different language. This internal name is what is stored in a document, and the user name is automatically determined based on the internal name in exactly the same manner as the built-in functions.

In order to avoid possible conflicts between function names, B & E Software employs a naming convention which is very close to Apple's convention for application types. The first four characters of the internal name of the function (excluding the length byte) should be the same for all functions in any given function group, and this four-character identification should be registered with B & E Software. All four-character identifiers that consist solely of lower-case letters are reserved by B & E Software for its own external functions. Instructions on how to register your external function group may be found in the section "Support and Marketing".

Below is an example `'BXDe'` resource:

```
resource 'BXDe' (kResourceBase, "RTXFunctExample", purgeable) {
     kCurrentVersion,      /* This should be compiled for the current version      */
     noMailMerge,          /* Mailmerge is not used in this sample                 */
     noMessages,           /* Messages are not used in this sample                 */
     noTime,               /* Periodic calls are not used in this sample           */
     "RTXFunctExample",    /* …that's me !                                         */
     101,                  /* my version; in human terms, this is version 1.01     */
     verUS,                /* localization language                                */
     noTime,               /* Since noTime is set above, this field should also    */
                           /* be set to noTime.                                    */
{
           /* kExampleAdd */
           noMailMerge,
           noRefNeeded,
           {     requested,       value,     noRecalc,  valIntNum,
                 notRequested,    value,     noRecalc,  valIntNum,
                 notRequested,    empty,     noRecalc,  valNil,
                 notRequested,    empty,     noRecalc,  valNil,
                 notRequested,    empty,     noRecalc,  valNil,
                 notRequested,    empty,     noRecalc,  valNil,
           },
           "ExampleAdd",          /* user name localized for current language        */
      "bexfExampleAdd",     /* Internal name. should stay the same for all     */
                                  /* language versions.  */

           /* kIdentifyType */
           noMailMerge,
           noRefNeeded,
           {     requested,       value,     noRecalc,  valAll,
```

```
                notRequested,    empty,     noRecalc,  valNil,
                notRequested,    empty,     noRecalc,  valNil,
                notRequested,    empty,     noRecalc,  valNil,
                notRequested,    empty,     noRecalc,  valNil,
            },
            "IdentifyType",         /* user name localized for current language        */
            "bexfIdentifyType",     /* Internal name. should stay the same for all     */
                                    /* language versions.  */
        },
};
```

This `'BXDe'` resource defines two functions which can be used with RagTime 3.2, RagTime 4 or newer (**`kCurrentVersion`**). The next three fields indicate that no special feature is used with this sample.

The next two fields identify the function group's name and version number. This information is displayed in RagTime's Function List dialog box when a function from the external function group is chosen. The next field identifies which language this external function group is written for, using the constants defined in the file `SysTypes.r`. Since `noTime` is set for the `needsTime, noTime` flag, `noTime` is the next integer set.

Following the external function group information, the `'BXDe'` resource contains an array describing each of the functions available in the external function group. The functions in this group must have their `noMailMerge, isMailMerge` flags set to `noMailMerge`. Neither of these functions need recalculation by the `CalcAFunction` back call, so their `noRefNeeded, needsRef` flags are set to `noRefNeeded`. The first function expects at least one `valIntNum`, and, optionally, a second `valIntNum`. The second parameter is optional because it has its `notRequested, requested` flag set to `notRequested`. The second function expects only one parameter which can be any value type.

2.1.2 The 'BXCa' (External Function Category) Resource

Starting with RagTime 3.2, the Function List dialog in RagTime has been expanded to include categories. A category is a group of related functions which makes locating a particular function easier. An external function group is automatically assigned a category based on either the name of its `'BXDe'` resource or, if no name exists, from the name assigned to the group within the `'BXDe'` resource. This category name is placed in the last part of the category pop-up menu in the new Function List dialog.

Additionally, by defining a `'BXCa'` resource, a function group can include itself into a predefined category or create a new category. The structure of the `'BXCa'` resource is shown here:

```
type 'BXCa' {
     literal longint; /* The category this function belongs to. */
     pstring;         /* Name of the category. Ignored if a built-in categorie is used. */
};
```

The `literal longint` is a four-character type which identifies the category. This category type can either be a predefined RagTime category or a new category type. The predefined categories are listed below:

```
'fina' - Financial Functions
'stat ' - Statistical Functions
'math' - Mathematical Functions
'trig' - Trigonometric Functions
'text' - Text Functions
'date' - Date Functions
'srch' - Search Functions
'prnt' - Print Functions
'misc' - Miscellaneous Functions
'cnst' - Constants
```

Care should be used when defining new category types. In order to avoid possible conflicts between cate-

application types. The four characters of the category type should be registered with B & E Software. All four-character identifiers that consist solely of lower-case letters are reserved by B & E Software for its own category types. Instructions on how to register your category types may be found in the section “Support and Marketing”.

When a predefined category is used, the contents of the `pstring` at the end of the `'BXCa'` resource is ignored. However, when a new category type is being defined, then this string is used as the category name in the category pop-up menu in the Function List dialog box. Below is an example `'BXCa'`.

```
resource 'BXCa' (10000, purgeable) {
     'xfex',
     "RTXFunctExample Group",
};
```

2.2 Coding an External Function

Currently, B & E is distributing C/C++ interfaces for external functions.With this package you got a Metroworks CodeWarrior example project. The developer is, of course, free to translate the C/C++ interface to any other language or to use different development tools.

If you open the sample project and choose the make command in CodeWarrior’s “Project” menu you should get the file “First Sample” containing the external function group which is described here. Put the file in RagTime‘s “B & E” folder and (re-)start RagTime to make the external functions available.

The sample project has an access path to the interface files which are included with this package. If you like to start with your own external function without changing the sample, you can duplicate the sample folder and modifiy the copy as you like.

Note: The sample project is an Macintosh 68K project. The external function technology was created for RagTime 3 before the PowerPC was invented. Therefore theinterface is 68K only, no Code Fragment Manager calls are done when the code resource is loaded. Of course, the “RagTime 3 Function Adapter” takes care that the 68K external functions work with the PowerPC version of RagTime 4.

2.2.1 Overview

When RagTime needs to call an external function group, it loads the `'BXCo'` resource into memory, places a pointer to an `RTXRecord` on the stack and jumps to the first byte of the `'BXCo'` resource. This first byte is referred to as the entry point of your `'BXCo'` resource. The entry point should be set up as a Pascal function expecting a `RTXPtr` as an argument and returning an `Integer`. RagTime and the external function group communicate through the `RTXRecord`. The `RTXRecord` contains a selector which tells the external function group why it is being called and, when necessary, where it can return a value. A series of back calls, which are routines executed by RagTime in order to aid the external function group, are also provided.

For a good overview of how external code resources work in general, please read the MacDTS technical note “Stand-Alone Code, ad nauseam”. See also the Metroworks CodeWarrior reference books.

2.2.2 Entry Point

The Pascal declaration for the entry point of the `'BXCo'` resource is a main function, declared as:

```
pascal short main (RTXPtr callBlock);
```

Make sure that the entry point of your code resource is at the beginning.

2.2.3 The RTXRecord

During a call to a connection tool, RagTime will pass a pointer to an `RTXRecord` having the following structure. Fields that are filled in by RagTime are indicated by the symbol `-->` in the comment, fields that are

filled in by the external function are marked with the symbol `<--`, and fields that are changed by both are marked with the symbol `<->`.

```
struct RTXRecord {
     long       refCon;          // <-> Used by the external function. Set to zero before the
                                 // first call. Preserved by RagTime between calls.
                                 // and will be saved for the next call.
     short      selector;        // --> which function is called
     RTValuePtr parameter;       // --> points to the actual parameter
     RTValuePtr resultDest;      // <-- Function result. RagTime creates the pointer, but the exter-
                                 // nal function must fill the record pointed to by the pointer.
     ProcPtr    backPtr;         // --> RagTime entry point for back calls
     long       document;        // --> the current document for this call
                                 // If NIL, the call is document independent (like kInitSelector)
     long       sheet;           // --> the current sheet for this call.
                                 // If NIL, the call is sheet independent (like kOpenDocSelector)
     RTCell     cell;            // --> the cell of the formula. 0,0 if the call is cell independent
     short      resId;           // --> id of the 'BXDe' resource for this function group
     short      groupStart;      // --> for the 'CalcAFunction' back call
};
typedef struct RTXRecord RTXRecord, *RTXPtr;
```

`refCon` is provided for the external function group to use in any way that it wants. The field is set to zero by RagTime before the first call to the external function group. The developer can then use this field in any way necessary, usually to point to a block of memory holding any values that may be needed. RagTime preserves the value of this field until the application quits. While RagTime preserves the value of this field between calls, the developer is free to change it in any way while the program is running.

`selector` indicates why the external function is being called. Numbers greater than zero indicate a **function selector**, which is a function from the external function group. The order in which you identify your external functions in the `'BXDe'` resource indicates the order in which the selector calls them, with the first function defined as 1. There is also a predefined group of **special selectors** defined in the file `RTXFunctsIntf.h` which RagTime may call the external function group with.

```
enum {
     // special purpose selectors
     kInitSelector         =      0,   // Selector used for the initilization call
     kQuitSelector         =     -1,   // Selector used for the quit call

     // the following four messages are only sent if 'kWantsMessagesFlag' is set
     kOpenDocSelector      =     -2,   // a document was just opened
     kCloseDocSelector     =     -3,   // a document is about to be closed
     kActivateSelector     =     -4,   // a document is activated
     kDeactivateSelector   =     -5,   // a document is deactivated
     kTimeSelector         =     -6    // Selector used for periodic calls (See 'kNeedsTimeFlag')
};
```

When the program is booted, each function group is called once with `selector =` **`kInitSelector`**. This is the time to allocate globals, open connections or to initialize whatever your functions need. A call with `selector =` **`kQuitSelector`** is made before RagTime quits. This is the time to close files, databases and so on. It is not absolutely necessary to dispose of any handles because the heap zone is later reinitialized, but doing so is considered good style. You must not rely on your variables being initialized when you are called to quit. It could be possible that you are called to quit without ever having been called to initialize. `refcon` is guaranteed to have a value of zero unless your external function group changes it.

If the `wantsMessages` flag in the `'BXDe'` resource is set, then RagTime calls the external function while opening or closing a document, or when a document is activated or deactivated. It does this through the use of the **`kOpenDocSelector`**, **`kCloseDocSelector`**, **`kActivateSelector`** and **`kDeactivateSelector`**, and the **`document`**

If an external function group has the **needsTime** flag set, then RagTime calls the function group periodically with `selector` equal to **kTimeSelector**.

Warning: Other negative selectors are reserved for future extensions! Please make sure that your function does nothing if called with an unknown selector.

parameter is a pointer to the actual parameter of the function. Parameters are passed one by one, with the back call GetNextParam being used to get the next parameter. On entry to the function, `parameter` already points to the first parameter or is `NIL` if there are no parameters. This method is generally faster then passing a list of parameters (no memory manager usage) and avoids memory problems if a large list is processed.

resultDest points to a memory location where the result of a function from the external function group must be placed. When one of the functions from a function group is called, **resultDest** points to a `RTValue` record which RagTime allocates. When the function group is called with `selector` set to a special selector (`selector` < zero), `resultDest` is a `NIL` pointer and must not be used.

backPtr contains the entry point into RagTime, which is needed for every back call.

document contains a reference number to the current document. This number is actually the value of a handle which contains data pertaining to this document. While a particular document is open, `document` always contains the same value. You can take advantage of this fact to determine if a particular `document` is the same as another, even between two calls to the external function group. Although it is technically possible that the value of `document` is assigned for one document, the document closed and the same value being assigned to another `document`, the chances of this happening are extremely small. Do not, however, used a saved value of `document` over two calls to the external function group, as it is very possible that the `document` in question has been closed between the two calls.

`document` is equal to

- `NIL` for calls with selector values of `kInitSelector` and `kQuitSelector`,

- a reference number of the document in question for selector values of `kOpenDocSelector`, `kCloseDocSelector`, `kActivateSelector` and `kDeactivateSelector`,

- `NIL` for calls with a selector value of `kTimeSelector`,

- a reference number of the document containing the called function for all function selectors.

sheet contains a reference number to the current spreadsheet. This number is actually the value of a handle which contains data pertaining to this spreadsheet. While a particular frame exists in an open document, `sheet` always contains the same value. Exercise care, however, when using a `sheet` value over two calls to a function group, as it is possible that the document in question is closed, or the component deleted, between one call and the next.

`sheet` is equal to

- zero for all of the special selectors,

- a reference number of the spreadsheet containing the called function for all function selectors.

cell contains zero when `selector` is equal to one of the special selectors and contains the position of the called function within the `sheet` when `selector` is equal to a function selector. The `cell` field is of type `RTCell`, which is defined below.

```
struct RTCell {
     short r;          // the row, 1 is row 1 in the range
     short c;          // the column, 1 is column 1
};
typedef struct RTCell RTCell;
```

resId contains the resource ID of the `'BXCo'`, `'BXDe'` and `'BXCa'` resources for the function group.

groupStart contains the starting number of your external function group from within a list of installed external functions which RagTime maintains. `groupStart` has significance only within RagTime itself and should

be used by the external function group only as a parameter to the **CalcAFunction** back call.

2.2.4 Passing parameters and results

Both the result of a function call from an external function group and the individual parameters for the function are passed through a variable of type `RTValue`.

```
struct RTValue {
     short valKind;
     union {
          TExtended80  eNumber;    // valNum, valDate (dates are stored as days since 1/1/1904)
          long         eIntNum;    // valIntNum
          Boolean      eBool;      // valBool
          struct {                 // valText:
               short  eTextLen;   // length of the text
               Handle eTextHdl;   // the text is in this handle. If you pass me a value, you
          } t;                     // have to allocate and deallocate the handle. The exception
                                   // is the result (of course). If I pass you a parameter, I
                                   // will deallocate (or reuse) the handle the next time you
                                   // call GetNextParam. To avoid this, copy   eTextHdl to a local
                                   // variable and set it to nil. If you call GetCell or GetNextCell,
                                   // you have to deallocate the handle
          RTErrVal     eErr;       // valErr
          struct {                 // cellRange:
               long   eDocument;  // the document which contains the range
               long   eSheet;     // the spreadsheet which contains the range
               Rect   eRange;     // the range
          } r;
     };
};
typedef struct RTValue RTValue, *RTValuePtr;
```

valKind is an integer which defines what type of value is held in the record. The possible values for `valKind` are the various labels in this `CASE` statement. The only possible `valKind` value not represented in the `CASE` statement is **valNil**, since setting `valKind` to `valNil` is the only assignment necessary to complete a `RTValue` variable.

When `valKind` is set to **valNum**, **eNumber** is used and represents a number of type `TExtended80`,this is an 80 bit floating point type. The exact definition of this value depends on the compiler type and settings. In RagTime 4 most numerical values will be passed as **valNum** even if they could be **valIntNum**.

When `valKind` is set to **valDate**, **eNumber** is used and represents an extended type holding the time in days since midnight, January 1, 1904.

When `valKind` is set to **valIntNum**, **eIntNum** is used and represents a whole number limited to the range -2 147 483 648 to 2 147 483 648.

When `valKind` is set to **valBool, eBool** is used to hold a boolean value.

When `valKind` is set to **valText**, **eTextLen** is used as a count of bytes for the text held in the block pointed to by **eTextHdl**. This `RTValue` presents an interesting problem, since a question could arise as to which function should create and dispose of the heap blocks associated with the `eTextHdl`.

Follow this general rule: if RagTime creates the handle, RagTime disposes of it, and if the external function group creates it, then the external function group should dispose of it.

There are, however, two exceptions to this rule:

First, if a function group needs to return a text result in the `resultDest` field of the `RTXRecord`, then the function group must create the handle, and RagTime disposes of it.

Second, if the function group calls the back calls `GetCell` or `GetNextCell` documented in the section “Back Calls”, and a text value should be returned, then RagTime creates the text handle, but the function group is responsible for disposing of it. The function group must also be sure to set `eTextHdl` to `NIL` if it ever disposes of the heap block associated with `eTextHdl`.

When `valKind` is set to **`valErr`**, **`eErr`** is used and represents one of the possible RagTime external function errors. The declaration for the `RTErrVal` type is included here.

```
enum RTErrVal    {
     errNoErr    = 0,                // no error
     errDivZ,                        // division by zero in the SANE meaning
     errNum,                         // numerical error other then 'errDivZ'
     errNA,                          // not available
     errRef,                         // reference error
     errVal,                         // wrong type of value
     errDate,                        // wrong date
     errRange,                       // wrong index
     errComplex,                     // calculation stack overflow
     errUnimpl                       // unimplemented function
};
typedef enum RTErrVal RTErrVal;
```

Type `RTErrVal` contains all possible error types that the RagTime spreadsheet can return. `errDivZ` should only be used as defined by SANE. `errNum` should only be used for real numeric errors, not for situations like a number being out of a desired range, in which case `errRange` would be used. In the same way, a date located outside of an acceptable range would get the error `errRange` rather than `errDate`. Please refer to the RagTime manual for a more complete explanation of the various error values.

When `valKind` is set to **`cellRange`**, **`eDocument`** contains a reference number to the document containing the range, **`eSheet`** contains a reference number to the spreadsheet containing the range, and **`eRange`** is a `Rect` which contains the dimensions of the range in row and column coordinates. The back calls `GetCell`, `GetNextCell` and `SetCell` would then be used to read or write value in `eRange`.

When one of the functions from a function group is called (that is, when the function group is called with the `selector` field of the `RTXRecord` set to a number greater than zero), the first parameter to the function is pointed to by the `parameter` field of the `RTXRecord`. Additional values (if any) can be retrieved one at a time by using the `GetNextParam` back call. Each call to `GetNextParam` places its value into the record pointed to by the `parameter` field of the `RTXRecord`. Each new parameter overwrites the old contents of the `RTXRecord`, so any values that are needed should be copied to local variables before calling `GetNextParam`. In particular, if RagTime passes the function an `RTXPtr` which points to an `RTXRecord` containing a `valText` parameter, and you need to preserve this value, you should copy the `eTextHdl` to a local variable and set `eTextHdl` to `NIL`. This text handle now belongs to the external function group, and cleanup of the handle should be addressed there. If `eTextHdl` is not set to `NIL`, then RagTime assumes that it still contains a valid, allocated handle and disposes of or reuses the handle on the next call to `GetNextParam`, or when the external function returns to RagTime, whichever occurs first.

Every call to a particular function in the external function group must return a result in the record pointed to by the `resultDest` field of the `RTXRecord`. Although it is possible to return a `valNil` result, this practice is not recommended, as it goes against the RagTime spreadsheet interface. Built-in functions always return some value, which makes it easier for a user to determine where functions are within the spreadsheet. An external function which returns a value of `valNil` may confuse a user expecting a value to be returned.

2.2.5 Error Reporting

To report errors to RagTime during a call to the external function group with the selector value equal to `kInitSelector`, set the value of the entry point function. If the value returned to RagTime is anything other than zero ([illegible]), RagTime does not load the external function group, the functions are not displayed in the

Function List, and any references to those functions that are already in a document will return the error "ILLEGAL!". If the value returned is negative, RagTime interprets this number as an error as defined in the file `Errors.p` and then displays a dialog reporting that the external function group was not loaded and explaining briefly why. If a value greater than zero is returned, RagTime does not report the error, so the external function group is responsible for telling the user why the external function group was not loaded.

During a call to the external function group with the `selector` value equal to a function selector (in other words, `selector` > zero), the result of the entry routine is currently ignored, but should be set to zero (`noErr`) to avoid problems with future versions of the interface. During these calls, the `resultDest` field of the `RTXRecord` should be used to report errors. For more information, see the section "Returning Results".

For all other selectors (that is, `selector` < zero), the entry point function value is ignored. As with the function selectors, the return value of the entry point function should be set to zero (`noErr`) to avoid problems with future versions of the interface. For all special selectors, the `resultDest` pointer is `NIL` and cannot be used for error values. Since these special selectors are only information calls, and since there is no clean way for RagTime to report an error when one occurs, no method for passing an error during these calls is provided.

2.2.6 External Function Code Structure

Typically, the code for an external function is structured in the following manner:

```
#include "FirstSample.h"
#include "RTXFunctsIntf.h"
#include "RTXBackCalls.h"
#include <Memory.h>
#include <OSUtils.h>
#include <Resources.h>
#include <ToolUtils.h>

//--------------------------------------------------------------------------------
//        constant for string list defined in 'BXRl' resource, plus constants
//        for the individual strings in the string list.
//--------------------------------------------------------------------------------
enum {
     kValNilStr            = 1,
     kValNumStr,
     kValBoolStr,
     kValDateStr,
     kValTextStr,
     kValIntNumStr,
     kUnknownValStr
};

//--------------------------------------------------------------------------------
//        Prototypes
//--------------------------------------------------------------------------------

void FillInResultText (short theStrList, short theStrIndex,
                                    RTValuePtr theValPtr);
```

The header file `RTXFunctsIntf.h` contains the types which were discussed above and the prototype for the function main as the entry point to the code resource. `RTXBackCalls.h` defines back calls that the external function can call. The back calls are defined in the next chapter. `FirstSample.h` defines constants for this example which are used in the C++ file and in the Rez file. This is how `FirstSample.h` looks like:

```
#define kResourceBase       10000
```

```
#define kExampleAdd        1
#define kIdentifyType      2
```

Back to FirstSample.cpp. Here is the function **`main`**:

```
//-------------------------------------------------------------------------------
//        main
//-------------------------------------------------------------------------------

pascal short main (RTXPtr callBlock) {

     short selector = callBlock->selector;

     switch (selector) {
           case kInitSelector:
           case kQuitSelector: {
                 // nothing to do here
                 break;
           }
           case kOpenDocSelector:
           case kCloseDocSelector:
           case kActivateSelector:
           case kDeactivateSelector: {
                 // these messages are only sent if 'kWantsMessagesFlag' is set
                 // in the 'BXDe' resource.
                 break;
           }
           case kTimeSelector: {
                 // this messages is only sent if 'needsTime' is set
                 // in the 'BXDe' resource.
                 break;
           }
           case kExampleAdd: {
                 // This function can accept either one or two parameters. If only one is
                 // received, then the result is 1 plus the parameter, otherwise the two
                 // parameters are added together.
                 callBlock->resultDest->valKind = valIntNum;
                 long firstLongInt = callBlock->parameter->eIntNum;
                 ::GetNextParam (callBlock->backPtr);
                 if (callBlock->parameter == NULL) {
                       callBlock->resultDest->eIntNum = 1 + firstLongInt;
                 } else {
                       callBlock->resultDest->eIntNum =  firstLongInt +
                                                         callBlock->parameter->eIntNum;
                 }
                 break;
           }
           case kIdentifyType: {
                 // This function returns text identifying the type of of the parameter.
                 callBlock->resultDest->valKind = valText;
                 short paramKind = callBlock->parameter->valKind;
                 short stringNo = kUnknownValStr;  // in case of unknown value type
                 if (paramKind == valNil) stringNo = kValNilStr;
                 else if (paramKind == valNum) stringNo = kValNumStr;
                 else if (paramKind == valBool) stringNo = kValBoolStr;
                 else if (paramKind == valDate) stringNo = kValDateStr;
                 else if (paramKind == valText) stringNo = kValTextStr;
                 else if (paramKind == valIntNum) stringNo = kValIntNumStr;
                 FillInResultText (kResourceBase + 1, stringNo, callBlock->resultDest);
```

```
            }
      }
      return noErr;
};

//--------------------------------------------------------------------------------
// Fill in the RTValueRec pointed to by theValPtr with the text
// specified by the string list and index.
//--------------------------------------------------------------------------------
void FillInResultText (short theStrList, short theStrIndex,
                                        RTValuePtr theValPtr) {
      Str255 theString;

      ::GetIndString (theString, theStrList, theStrIndex);
      short length = (short) theString[0];
      theValPtr->t.eTextLen = length;
      theValPtr->t.eTextHdl = ::NewHandle (length);
      ::BlockMove (&(theString[1]), *(theValPtr->t.eTextHdl), length);
};
```

A fairly straightforward way to structure your main routine is to have a `switch` statement which performs code depending on which selector is called. Notice that some of the `case` labels are only needed if the `'kWantsMessagesFlag'` or `'kNeedsTimeFlag'` flags are set in the `'BXDe'` resource. Also, any unknown selectors are correctly ignored.

3 Back Calls

To aid in the programming of add ons, the RagTime add-on interface provides a series of back calls, which are utility and access routines executed by RagTime. Please note that the `backPtr` parameter expected at the end of all the back calls may be found in the `RTXRecord` for external functions and in the `RTXToolRecord` for connection tools.

Not every back call is available in every version of RagTime, and some back calls are available only for external functions. Be sure to use only those functions available to you for your particular add on, as using other back calls may cause RagTime to report an error to the user or crash. A table of back calls available for each add on/RagTime combination you are using is provided here.

Back Call	No.	Available in RagTime version…			
		3.1	3.1|7	3.2	4.x
GetLastBackCall	0			•	•
GetNextParam	1	•	•	•	•
GetCell	2	•	•	•	•
GetNextCell	3	•	•	•	•
SetCell	4	•	•	•	•
FindSheet	5	•	•	•	• 2)
CalcAFunction	6	•	•	•	•
Text2Value	7	•	•	•	•
PrivateToDeskScrap	8		•	•	•
CalcNow	9		•	•	•
MailMerge	10		•	•	•
SaveDocument	11			•	•
PrintDocument	12			•	• 3)
GetDocumentName	13			•	•
GetDataItem	14			•	• 3)
DeleteDataItem	15			•	• 3)
DocumentChanged	16			•	•
OpenPreferences	17			•	•
UpdateWindow	18			• 1)	•
GetCellAlways	19			• 1)	•
GetNextCellAlways	20			• 1)	•

1) These back calls are not available in early shipped versions of RagTime 3.2. Use `GetLastBackCall` to determine whether these back calls are defined.

2) This back call finds RagTime 3 spreadsheets only, see the description for more information.

3) These back calls are defined but do not work with RagTime 4.

3.1 GetLastBackCall

```
pascal short GetLastBackCall (ProcPtr backPtr);
```

This back call returns the number of the last implemented back call. This can be used to check for the existence of a specific back call in different versions of RagTime. `GetLastBackCall` allows for the addition of back calls in the future without having to change the interface version number. See the back call table above

to determine the number of a particular back call.

Note: This back call is only available in RagTime 3.2 or newer.

3.2 GetNextParam

```
pascal void GetNextParam (ProcPtr backPtr);
```

This procedure gets the next parameter for external functions during calculation and places it into the record pointed to by the `parameter` field of the `RTXRecord`. If the `parameter` field is a `NIL` pointer after this call, then there are no more parameters. Please refer to the section “Passing Parameters and Results” for more information.

3.3 GetCell

```
pascal void GetCell (long sheet, long document,
                     short col, short row,
                     RTValue *value, short typeMask, ProcPtr backPtr);
```

This procedure gets the contents of a particular cell. `sheet` is a reference number to a particular spreadsheet, `document` is a reference number to a particular document, `col` and `row` indicate the location within the spreadsheet, `value` is an `RTValue` record to which the result is returned, and `typeMask` indicates to RagTime which data types the caller is interested in.

If a text value is received in the `value` parameter, then it is the external function group’s responsibility to dispose of the block associated with the handle `eTextHdl`.

The `typeMask` parameter can contain one or more of the value type masks defined at the beginning of the interface. All values are converted to a value which is allowed in the `typeMask`. If conversion is not possible, the error value `valErr` is returned, unless `valMaskErr` is not in the mask. In this case, `GetCell` does not return, and the error “VALUE!” is returned by the function. If the cell is empty, the value `valNil` is returned, unless `valMaskNil` is not in the mask, in which case RagTime returns a default value for the cell. The default values are zero for valMaskIntNum and valMaskNum, an empty string for valMaskStr, and “false” for valMaskBool. If the cell referred to contains an error value, then RagTime does not return, no matter what is defined in the mask, and the result returned in the spreadsheet for your function is the same error value as that of the cell in question.

3.4 GetNextCell

```
pascal Boolean GetNextCell (long sheet, long document,
                            short *col, short *row,
                            const Rect *range, RTValue *value,
                            short typeMask, ProcPtr backPtr);
```

This function searches the `range` of `sheet` in `document`, starting at `col, row`, and sets `col, row` to the first non-empty cell found that contains a value allowed by `typeMask`. The function searches one row at a time, checking each column in the row. Any values outside the range are automatically adjusted, so that a search can be continued simply by adding 1 to the `col` parameter. If `range` is `NIL`, then the whole sheet is searched. `valMaskErr` is used in the same way as in `GetCell`. Unlike `GetCell`, however, `GetNextCell` ignores `valNil` if it is in the `typeMask`.

3.5 SetCell

```
pascal void SetCell (long sheet, long document,
                     short col, short row,
```

```
                    RTValue *value, ProcPtr backPtr);
```

This procedure puts value into cell col, row of the spreadsheet sheet in document document. Even though value is a VAR parameter, it is not changed by this back call, and if value contains a text value, then the external function group is responsible for disposing of the handle.

3.6 FindSheet

```
pascal void FindSheet (short pageNum, short frameNum,
                       StringPtr path, Boolean openDocu,
                       long *sheet, long *document,
                       ProcPtr backPtr);
```

This procedure searches for the document with name *path. If path is NIL, the current document is used. If *path is just a file name without any colons, only documents which are open and visible to the user are returned. If path^ contains any colons, it is taken as a full or partial path name. The partial path name starts at the location of the current document (or the program, if the document is not saved yet). In this case, all open documents, whether visible to the user or not, are always returned. Closed documents are opened if openDocu is TRUE. If a document is found, the spreadsheet in the frame with the number frameNum on page pageNum is searched. If found, it is returned in sheet, otherwise, sheet is set to zero.

Note: In RagTime 4 a spreadsheet does not belong to a single container. This back call can find spreadsheets which are named like those which were imported from RagTime 3 documents. That means, that the page and frame number can be found in the spreadsheet's name.

3.7 CalcAFunction

```
pascal void CalcAFunction (short groupStart, short functionIndex,
                           long document, long sheet,
                           ProcPtr backPtr);
```

This call is used to trigger recalculation of individual functions from an external function group. In order for this back call to work correctly, you must first set up the function that you want recalculated so that its needsRef flag is set in the 'BXDe' resource. Recalculation of the function is started by passing the groupStart field of the RTXRecord, and the number of the function that needs to be recalculated in document, sheet.document and/or sheet may be zero. When document is zero, all occurrences of the function in question in all sheets in all documents are recalculated. When document contains a valid number and sheet is zero, all occurrences of the function in question in all sheets of the document are recalculated.

This routine only sets a flag in RagTime that calculation must be carried out. Calculation does not actually begin until after the external function returns.

3.8 Text2Value

```
pascal Boolean Text2Value (Ptr textPtr, short textLen,
                           RTValue *result, ProcPtr backPtr);
```

This is the RagTime text parser. It tries to convert the given text to a number or a date. The rules are exactly the same as during manual data entry. If a conversion is not possible, Text2Value returns FALSE and result remains unchanged.

3.9 PrivateToDeskScrap

```
pascal void PrivateToDeskScrap (ProcPtr backPtr);
```

Before reading the desk scrap, an external function group should call this procedure to make sure that RagTime converts its private scrap to the desk scrap.

3.10 CalcNow

```
pascal void CalcNow (ProcPtr backPtr);
```

Triggers recalculation of everything. If calculation is set to automatic, calculation starts after the call; otherwise, it starts as soon as the user selects the command Calculate All. This back call may only be used if the function group is called with one of the special purpose selectors. This function was added for HandiWorks/ RagTime Classic 2, and is here for compatibility only. It triggers a recalculation of **ALL** formulas. This interface provides the function `CalcAFunction`, which provides a more specific means of controlling recalculation than `CalcNow`.

As opposed to HandiWorks/RagTime Classic 2, which does the calculation synchronously, `CalcNow` only triggers recalculation in RagTime 3 and RagTime 4 and returns before calculation actually starts.

3.11 MailMerge

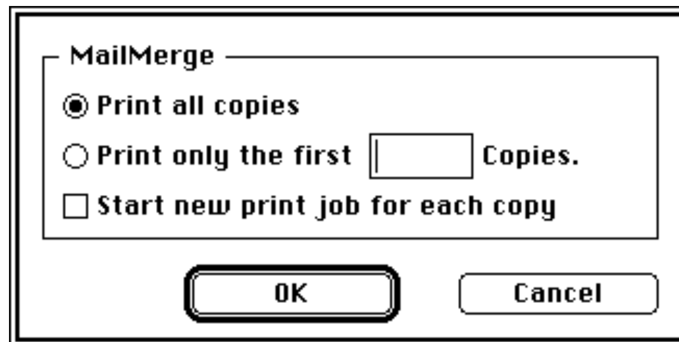
```
pascal short MailMerge (Boolean printStop, ProcPtr backPtr);
```

This function returns the number of the current print in the print cycle. It stops printing if the `stopPrinting` argument is set to `TRUE`, and always returns 1 if printing is not in progress. Currently there is no clean way to tell whether printing is in progress, and you shouldn't need to know. Don't forget to set the `kIsMailMerge` and `kNeedsReference` flags for every function using `MailMerge` and the `kHasMailMerge` group flag if you use `MailMerge` at all. RagTime uses these flags to determine whether it should show the mail merge dialog before printing.

A short example of how this function could be used is provided below. It assumes that your resources have been set up correctly as described above. Use of this example function would cause up to 5 copies of the document containing the function to be printed, with the function result being the number of the print.

```
switch (selector) {
     case kMyMailMerge: {
           callBlock->resultDest->valKind = valIntNum;
           short printNumber = ::MailMerge (false, callBlock->backPtr);
           if (printNumber < 5) {
                 callBlock->resultDest->eIntNum = printNumber
           } else {
                 callBlock->resultDest->eIntNum = 1;
                 printNumber = ::MailMerge (true, callBlock->backPtr);
           }
           break;
     };
}
```

If the user has at least one of these example functions in a document, selecting the command **Print** brings up this dialog:



Your external function is then called at the beginning of the printing of each page until either the maximum number of copies indicated by the user is reached or your function calls the `MailMerge` backcall with `printStop` set to `True`, at which point printing stops.

3.12 SaveDocument

```
pascal OSErr SaveDocument (long document, FSSpecPtr dest,
                           Boolean askForName, FSSpecPtr actualFile,
                           ProcPtr backPtr);
```

This function saves `document` to disk, returning any error as the function result. If the `dest` argument is `NIL`, then this back call works just like the save commands from the File menu. If the argument `askForName` is set to `FALSE`, this is treated just like a selection of the **Save…** menu item, and if `askForName` is set to `TRUE`, `SaveDocument` performs the same action as a selection of the **Save as…** menu item. A function return value of 1 indicates that the user cancelled the operation.

If the argument dest is a pointer to an `FSSpec` record, then document is saved to that file, overwriting the file without warning if it already exists (regardless of what type of file it is). The argument `askForName` is ignored in this case. When this type of document save is performed, the `document's` dirty flag is not changed.

Regardless of what type of save is done, successful completion sets `actualFile` to the file that was used. If `dest` is specified, then the FSSpec pointed to by `actualFile` always has the same values as the FSSpec pointed to by `dest`.

3.13 PrintDocument

```
pascal OSErr PrintDocument (long document, long flags,
                            THPrint hPrint, ProcPtr backCallFunction,
                            long backCallRefCon, ProcPtr backPtr);
```

This back call does not work in RagTime 4. Nevertheless it is defined and returns `OSErr iPrAbort (=128)` to indicate a requested canceling by the user or the application. If you want to use this back call with RagTime 3, please contact us, if you need further information.

3.14 GetDocumentName

```
pascal void GetDocumentName (long document, StringPtr docuName,
                             Boolean fullPath, ProcPtr backPtr);
```

Returns the name of `document` in the Pascal string pointed to by `docuName`. The Pascal string pointed to by `docuName` must be provided by the caller. If `fullPath` is `TRUE`, then the full path to the document is returned, in which case the string pointed to by `docuName` must have a size of at least 256 bytes (that is, type `Str255`). If `fullPath` is `FALSE`, then only the name of the document is returned, in which case the string pointed to by

docuName must have a size of at least 64 bytes (that is, type Str63).

3.15 GetDataItem & DeleteDataItem

```
pascal Handle GetDataItem (long document, OSType theType,
                           short theID, long size,
                           ProcPtr backPtr);
pascal void DeleteDataItem (long document, OSType theType,
                            short theID, ProcPtr backPtr)
const short kDontCreateItem = -1;
```

These back calls do not work in RagTime 4. Nevertheless they are defined and GetDataItem returns NIL. If you want to use these back call with RagTime 3, please contact us, if you need further information.

3.16 DocumentChanged

```
pascal void DocumentChanged (long document, ProcPtr backPtr);
```

This routine marks the document document as being changed. This is the same as setting the dirty flag, and it enables the **Save…** menu item in the File menu of RagTime. Call this after changing a data item. Creation of a data item automatically marks the document as changed.

3.17 OpenPreferences

```
pascal OSErr OpenPreferences (short *resRefNum, ProcPtr backPtr);
```

This function creates the RagTime preferences file if necessary and opens the resource fork. An operating system error is returned if there are problems performing this operation. If noErr is returned from the call to OpenPreferences, then the external function is responsible for closing the preferences file with a call to CloseResFile (resRefNum) before returning to RagTime.

Please notice that opening the preferences file changes the current chain of open resource files. (See section 4 “Tips & Caveats” on page 24 below for more information about the chain of open resource files.) To avoid problems with this situation, you should save the current resource file, call OpenPreference, perform whatever actions need to be performed while you have access to the preferences file, close the preferences file, then set the current resource back to the one that you saved before calling OpenPreferences.

3.18 UpdateWindow

```
pascal void UpdateWindow (WindowPtr whichWindow, ProcPtr backPtr);
```

This back call updates the window pointed to by whichWindow. If whichWindow is not a RagTime window, this back call is ignored. UpdateWindow is usefull if you set up a movable window in front of a RagTime window.

Note: This back call is not defined in all versions of RagTime 3.2. Use GetLastBackCall to determine whether this back call is available.

3.19 GetCellAlways

```
pascal Boolean GetCellAlways (long sheet, long document,
                              short col, short row,
                              RTValue *value, short typeMask, ProcPtr backPtr);
```

This back call is identical to GetCell with the following exception. If valMaskErr is included in typeMask, then both conversion exceptions and cells which contain errors will return value of type valErr. This function will return FALSE in case of a conversion error. which allows callers to differentiate between conversion er-

rors and cells which have an error value.
GetCellAlways **always returns if** valErr is in typeMask. If the cell is an error, or there is a conversion error, and valErr is not in typeMask, this call will not return.

Note: This back call is not defined in all versions of RagTime 3.2. Use GetLastBackCall to determine whether this back call is available.

3.20 GetNextCellAlways

```
pascal Boolean GetNextCellAlways (long sheet, long document,
                                  short *col, short *row,
                                  const Rect *range, RTValue *value,
                                  short typeMask, Boolean* conversionWasSuccessful,
                                  ProcPtr backPtr);
```

This back call is identical to GetNextCell with the following exception. If valMaskErr is included in typeMask, then both conversion exceptions and cells which contain errors will return value of type valErr. This function will set conversionWasSuccessful to FALSE in case of a conversion error, which allows callers to differentiate between conversion errors and cells which have an error value.
GetNextCellAlways **always returns if** valErr is in typeMask. If the cell is an error, or there is a conversion error, and valErr is not in typeMask, this call will not return.

Note: This back call is not defined in all versions of RagTime 3.2. Use GetLastBackCall to determine whether this back call is available.

4 Tips & Caveats

4.1 Open Resource Files

Starting with RagTime 3.2, B & E has tried to make RagTime easier for the user by allowing add-on files to be used without having to be installed into the program. In order to make use of these files, RagTime must open them and keep them open while running. Since the code and support resources are contained in the resource fork of the add-on file, RagTime must open this fork. Due to the way the resource manager handles this situation, accessing resources from multiple open resource files can causes a number of complications for both RagTime and the add ons which developers must be aware of. In order to understand the problem, the developer should first read the Resource Manager chapter of **Inside Macintosh**, and then refer to the notes below.

4.1.1 Saving and Restoring

RagTime always makes sure that, when the external function is called, the current resource file is the one in which the external function group is contained. However, you must exercise care in creating external function groups which open extra resource files. A problem could occur if an add on opens another resource file and then assumes that this resource file will be still be in the chain of searched resources the next time the add on is called. This will not be the case, since RagTime sets the current res file to the add-on file itself (or the RagTime file, if the add on is installed) when calling the add on, leaving the file opened by the add on out of the searched resource chain (see illustration above).

The recommended solution to this problem is the following: whenever you have to access a resource from another resource file, first save the current res file, then set the current res file to the open resource file in question, access your resource, and finally set the current resource file back to the saved one. Also, in order to avoid resource numbering conflicts during printing (see above), it is recommended that you keep extra resource files open only as long as is necessary.

4.2 Reentrant Code

Be aware that your code resource must be reentrant because of a few situations that could arise. First, if you call the `GetNextParam` back call, the next parameter may be the result of another instance in the spreadsheet of the function you are currently executing, which means that your function would be called to calculate before `GetNextParam` returns. The other situation in which your code may be reentered is if you call a back call which causes a failure condition in RagTime (an out-of-memory error, for example). If this is the case, RagTime might not return from the back call, but call your function group again with `selector` set to `kQuitSelector`.

Usually, the only thing that you have to do to make your code reentrant is to make sure that anytime you need to lock a handle, you first check the handle's current state (lock or unlocked), save that state, set it to the state that you need it to be in and later restore it to its old state.

4.3 Memory Usage

Just to overstate the obvious, developers should keep in mind that they are working within the RagTime environment and should be careful not to use more stack or heap space than is absolutely needed by the add on. RagTime is a large program to begin with and there will always be users that do not give it enough

running, be sure to advise users of this situation so that they can give RagTime as much memory as is necessary.

4.4 The SANE Environment

Developers should be aware that RagTime implements its own SANE halt-handling routines in the 68K versions. If a SANE exception occurs during the execution of your code, it is possible that RagTime will take over, and the remainder of your code will not be executed.

This situation can be avoided by temporarily disabling RagTime's SANE halt-handling routines. This can be done by the following code at the beginning of your code:

```
Environment e;
::GetEnvironment (&e);
::SetHalt (invalid + underflow + overflow + divByZero + inexact, false);
```

On exit from the routine, you should, of course, restore RagTime halt-handling routines by doing the following:

```
::SetEnvironment (&e);
```

However, the situation is just a little more complicated than that. Whenever you call a back call, RagTime takes over to perform calculations, retrieve data, and so on and expects that the halt-handling routines are available. If you change the SANE environment, it is very important that you reset it to RagTime's environment before calling any back calls. If you do disable the RagTime SANE halt-handling routines at the beginning of your code, all back calls should follow the following steps:

```
::SetEnvironment (&e);          // Restore the RagTime SANE halt-handling routines.
::GetNextParam (backPtr);       // This is just one of many back calls you could be using.
::GetEnvironment (&e);          // Save the current environment again.
// Disable halt handling routines.
::SetHalt (invalid + underflow + overflow + divByZero + inexact, false);
```

More information on SANE and the halt mechanism may be found in the Apple Numerics Manual, Second Edition.

5 Support & Marketing

If you are planning to develop external functions for RagTime, please send us your name, your company name, your full address (including phone, fax, and e-mail if applicable), and a brief description of your external functions. Please include the four-character external function group identification as described in the section “`'BXDe'` (External Function Description) Resource”.

B & E Software can be contacted at:

B&E Software GmbH
Itterpark 5
40724 Hilden
Germany

E-Mail: DevSupport@BESoftware.com
WWW: http://www.BESoftware.com/
Fax: [49] (2103) 96 57 96
Phone: [49] (2103) 96 57 0

As a registered developer of RagTime add ons, you will receive support and information about any new development about the RagTime add on interface.

If you need developer support on questions concerning add ons, or if you have suggestions concerning the functionality or interface, please feel free to contact us.

If you send us a description of your external function group (including prices, language version(s) and availability), we will forward the information to all RagTime distributors and will add information to our WWW-Pages.

RagTime is a registered trademark of B & E Software GmbH. Metrowerks and CodeWarrior are registered trademarks of Metrowerks Inc. All other trademarks are hereby recognized as the property of their respective owners.