Babelfish

Original specifications by Dave Hecker and Steve Stephenson
Additional modifications by Dave Hecker and Bill Tudor

Seven Hills Solutions Specialists
1254 Ocala Road
Tallahassee, FL 32304

Technical help: support@sevenhills.com
Everything else: sales@sevenhills.com

Notice

Thanks go to the Committee

The following people reviewed the specifications for Babelfish at one time or another: Bill Tudor • BrainStorm Software • Bryan “Zak” • Dave Hecker • Dave Lyons • Ewen Wannop • Jason Harper • Joe Wankerl • Lunatic E’Sex • Matt Deatherage • Mike Nuzzi • Richard Bennett • Steve “Diz” Disbrow • Steve Stephenson • Bill Heineman • Ian Brumby • Mike Westerfield • Tony Diaz • Paul Parkhurst

Table Of Contents

Babelfish

What Is It?

Babelfish is a tool that applications use to import and export (or load and save) data without knowing anything about a particular data format. Through Babelfish, the application relies upon *translators* to do all the specifics of reading and writing files.

An application can use standard functions to display the standard Babelfish interface to the user (including a “Progress Window”), or an application can use “advanced” functions to control import/export with no user interface. Babelfish even allows several translators to be active at a single time.

Babelfish functions equally well in both 320 and 640 modes so any application can use it. *NOTE: A particular translator might operate only in one mode; this is not a problem for Babelfish.*

Babelfish requires Apple IIGS System Disk 6.0 or later.

Why Use It?

One big benefit is saving repetitive coding time. Instead of writing three different routines to import three different file formats, you need to write only one chunk of code to import a *single* type of data. Through Babelfish you will instantly have access to many file formats. And your application’s capabilities will grow every single time a new translator is written.

Your application’s code is smaller because translators are separate entities. And because they’re separate they can be updated or enhanced without changing application code.

Using Babelfish will give a variety of applications a standard way to present import/export capabilities, making users more comfortable.

If you’re going to write code to import and/or export a certain type of file, if you write it as a *translator* instead of hard-coding it into your program, you’re making it easy for *any* application to read/write that same file format. This hopefully will promote goodwill and sharing among developers.

Miscellaneous Topics

Claris XTND

We’re aware of this on the Macintosh. In short, we think our implementation is better, more flexible, and better prepared for future growth.

GraphicWriter III Translators

For those of you familiar with the GraphicWriter III translator format, here is a brief list of differences between “Babelfish Translators” and “GW III Translators”:

- A translator now has several required resources which identify the translator to Babelfish.
- When importing with GW III, you first select the format of the file you’re interested in, then you pick the file (the translator selection drives the files you can choose). With Babelfish it’s done the proper way: The user is presented a list of files which can be imported, and his file selection drives which translators can be chosen. Usually the user has no idea what format a file is; by selecting the file first, *we* can tell *him* which translators apply!
- Babelfish does not create a direct page for translator use. If you need a DP you must create, maintain, and dispose of it yourself.
- In the old scheme if you were a text translator you passed the text record; if you were a graphic translator you passed the graphic record. Because Babelfish cannot know about the specific formats of data, there are now two main records to deal with: The first is a standard “Transfer Record” or XferRec for short. The second is the specific data format’s record.

  The XferRec has the same format and length for *all* translators. As before, the specific data format’s record is specific to the translator.
- With GW III only one translator is active at a time. With Babelfish, several translators can be open and active at the same time.

Overview—Using Babelfish

Communicating with Babelfish

Inter-Process Communication (IPC) is the method for communicating among the various programs involved in the translation process. The application sends requests to Babelfish. Babelfish sends requests to the translators. The translators may also send requests to Babelfish or to supporting subtranslators. *See also "Programmer's Reference for System 6.0", Chapter 30.*

Babelfish installs a message handler (AcceptRequests) in order to respond to the assigned requests made by applications and translators. Each translator also installs a message handler to respond to the assigned requests made by Babelfish. Support files also install a message handler in order to respond to the assigned requests made by applications and translators.

If the message receiver is not present when called, the Tool Locator will return error $0120 (`reqNotAccepted`).

Applications and translators communicate with BF by calling SendRequest with the proper request code, directing the request to "Seven Hills~Babelfish~", with a `sendHow` value of `sendToName + stopAfterOne`.

Parameters are passed between the sender and receiver in `dataIn` and `dataOut`. In all cases, `dataIn` and `dataOut` are pointers to buffers containing the parameters.

Even though some input parameters are less than 4 bytes and could be placed *directly* in `dataIn`, for consistency they will always be placed in a location that is *pointed to* by `dataIn`.

The Tool Locator fills in the first word of the structure pointed to by `dataOut` with the number of times the message was received. This is a meaningless value as all messages passed between participants in the translation process must send their messages with a `sendHow` that includes `stopAfterOne`; which means that the Tool Locator will either make this value a one if the message was accepted or return an error (`reqNotAccepted`).

Some messages don't need to return any data to the sender. Technically, these messages could pass NIL for `dataOut`; however, there would be no way to report any error or status information. (If there were an error, the receiver could only signal it by refusing to accept the message, which would not return an accurate picture to the sender.) Throughout Babelfish, all message traffic will use `dataOut`; and it will always point to a structure that begins with 2 words–the first is a throwaway for the Tool Locator, and the second is the receiver's result code–followed by any returned data.

Starting Babelfish

The application calls BFStartUp to “log in” with Babelfish.

BFStartUp follows these steps:

1) If Babelfish **is** started already, skip to step 3.

2) If Babelfish is **not** started:
   Initializes a “startups” counter to 0.
   Walks through a special folder where translators are stored, opening each translator’s resource fork and loading in the TrData (contains identifying information about the translator) and TrFilter (similar to a standard file filter) resources. It then uses this information to build an internal table of the state of the translator universe. *NOTE: Only properly-formatted translators are allowed; if a bad file is found it gets skipped (only unrecoverable errors will cause BFStartUp to fail).*

3) BFStartUp increments the “startups” counter and returns the XferRec size value.

NOTE: These steps are purely to make understanding the startup process easier; the actual method that Babelfish uses to start up and shut down is not discussed here.

You may start Babelfish only when you need it. However, for best speed performance it is highly recommended that you start up Babelfish only once (either at application startup or the first time Babelfish is needed) and then *leave it active* until your program quits.

If you use Babelfish in an NDA, it is recommended that you start up Babelfish during NDAOpen, or start it the very first time you need it. For speed you should then leave Babelfish started until DeskShutDown (and then call BFShutDown only if you successfully started it).

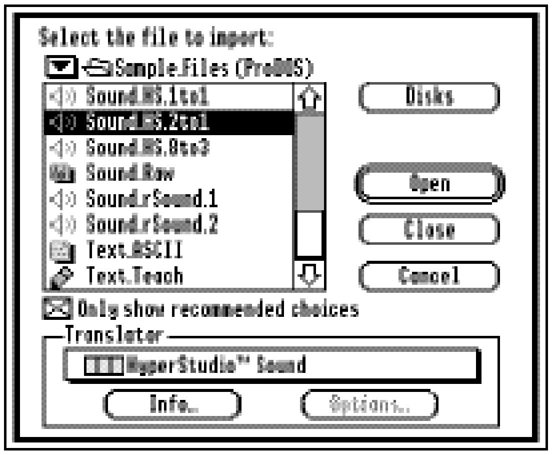
Note that Babelfish can be started at the same time by multiple applications and NDAs. However, Babelfish is not re-entrant and will not accept messages from another application while any request is still in progress (i.e. calling Babelfish from an interrupt routine won’t work). Further note that an individual translator may not be used by more than one application at the same time (a particular translator **can** be duplicated in the Finder, which creates a “unique” translator as far as Babelfish is concerned).

Importing a File

The application gets a block of memory for the XferRec (of the size returned from BFStartUp) and fills in some of the fields. Among other things, the application specifies “importing” and it specifies which Kinds of data it wants to receive.

BFGetFile

If the application does not have a filename already chosen for importation, it calls BFGetFile for the user to pick one. BFGetFile uses a custom SFPGetFile2 to interact with the user.

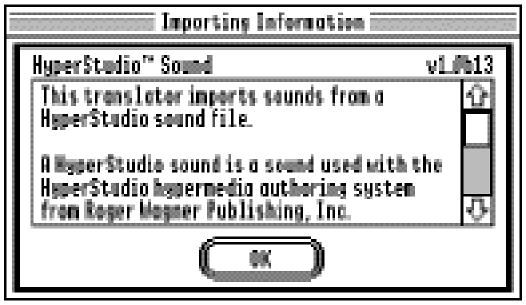


The displayed list of files is built by calling each translator’s TrFilter routine (TrFilter uses the data in the GetDirEntry record to form a rating value on its ability to translate the file). The checkbox determines what range of files is displayed.

After the file list is built and displayed, the user interacts with the dialog until a file is chosen. When a folder is highlighted, the translator list is empty and the Info and Options buttons are dim.

When a file is highlighted, the translator list fills with an alphabetical list of all the translators that can import the file. The translator with the highest ranking becomes the selected item and is scrolled to, if necessary. The Info button is active; the Options button is active only if the highlighted translator has special import options that can be set.

If Info is selected, Babelfish displays a modal window using the translator’s TrVersion and TrImportInfo resources:

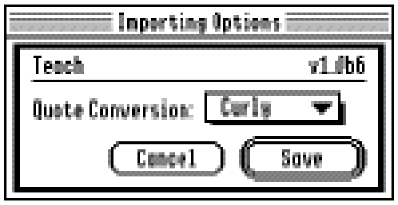

Importing Information

HyperStudio™ Sound v1.0b13

This translator imports sounds from a HyperStudio sound file.

A HyperStudio sound is a sound used with the HyperStudio hypermedia authoring system from Roger Wagner Publishing, Inc.

If Options is selected, Babelfish calls TrImportOptions so that translator can interact with the user. The translator presents a dialog of its own making that gathers whatever preference info is needed to do the importation. For example:

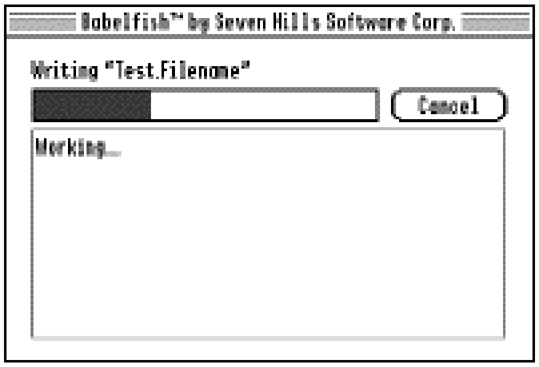


BFImportThis

Once a file is chosen (either through BFGetFile or some other means), the application calls BFImportThis. Babelfish loads the specified translator along with its stored options, then calls TrStartUp for importation.

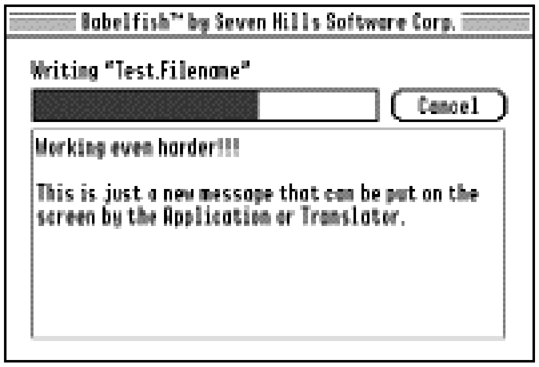
The translator's TrStartUp sets up buffers and variables and opens the file. It also sets the kind of data (in the XferRec) to reflect the single, exact kind of data that will be returned to the application.

If the application specified that progress was allowed, Babelfish opens the Progress Window. The Progress Window consists of a thermometer and an area in which the *translator* may display messages for the user (during importation *only the translator* may update the message):



Working…

The message area keeps the user occupied/informed during the importation. For example, a "Super Hi-Res Graphics" translator that can import multiple file formats (Unpacked Screens, Packed Screens, Apple Preferred Files, etc.) might want to state the exact kind of file being imported, as well as updating the message during the importation:



Working even harder!!!

This is just a new message that can be put on the screen by the Application or Translator.

Control is returned to the application. If an error is returned from BFImportThis then the application must cancel its import process. Otherwise, the import process continues…

If the application originally stated it could receive more than a single kind of data, it must now check the transfer record to determine exactly which kind of data is being returned.[1]

The application prepares itself to receive a particular data format, then enters the following loop:

1) The application sets Status and calls BFRead.
2) Babelfish checks Status to see if the application needs to abort for some reason (if so, Babelfish calls TrShutDown and returns to the application…see step 4).
3) Babelfish calls the translator's TrRead routine so the translator can load/process a chunk of the file. TrRead fills in the appropriate XferRec fields (Status, progress thermometer value, user message, data, etc.) and returns to Babelfish.
4) Babelfish checks Status:
   If "done" or "continue" Babelfish updates the Progress Window.
   If "done" or "abort" Babelfish calls TrShutDown.
   Return to the application.
5) The application checks Status:
   If "continue" then store the data into private structures and go to step 1.
   If "done" then all the data has been received and you're ready to go!
   If "abort" then dispose of any imported data because it's not valid.

After receiving "done" or "abort," the application does not make any further calls to Babelfish (except when he's completely done using Babelfish he will call BFShutDown). Once Babelfish says "done" or "abort," the importation is finished as far as Babelfish is concerned (the translator has already been shut down, the Progress Window is closed if necessary, etc.).

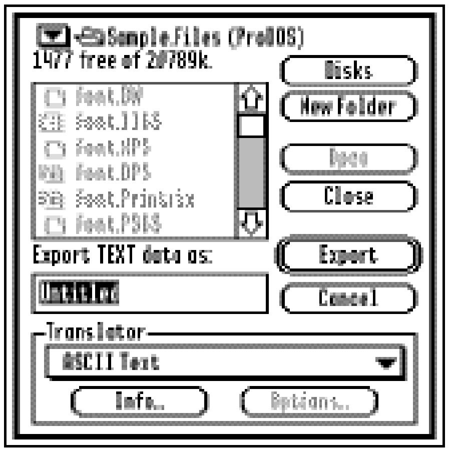
[1] Even if you only plan to support a single data format, it is recommended that you still follow the same flow outlined here. This provides consistent visual interface to the user, and it makes it easier for you to support additional formats at a later date (even if you think you might never support another format, you still might, so why no make it easier on yourself?).

Exporting a File

The application gets a block of memory for the XferRec (of the size returned from BFStartUp) and fills in some of the fields. Among other things, the application specifies “exporting” and it specifies the single, exact Kind of data it will be exporting.

BFPutFile

If the application does not have a filename already chosen for exportation, it calls BFPutFile for the user to specify one. BFPutFile uses SFPPutFile2 to interact with the user:



Sample.Files (ProDOS)
1477 free of 20789k.

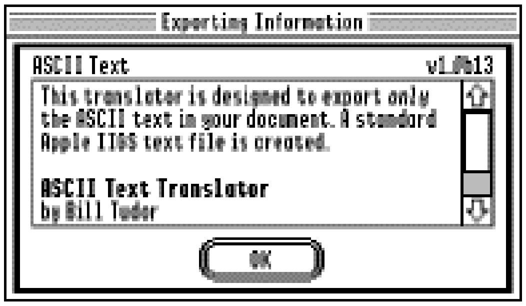
font.DW
font.IIGS
font.HPS
font.DPS
font.Printrix
font.PSGS

Export TEXT data as:
Untitled

Translator
ASCII Text

The translator list box displays an alphabetical list of all the translators that can export the specified kind of data. Because all export translators should be able to handle a particular data format equally well, the first item on the list is selected. The Info button is active; the Options button is active only if the highlighted translator has special export options that can be set.

If Info is selected, Babelfish displays a modal window using the translator’s TrVersion and TrExportInfo resources:



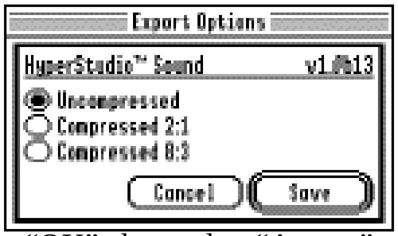
Exporting Information

ASCII Text v1.0b13

This translator is designed to export only the ASCII text in your document. A standard Apple IIGS text file is created.

ASCII Text Translator
by Bill Tudor

If Options is selected, Babelfish calls TrExportOptions so that translator can interact with the user. The translator presents a dialog of its own making that gathers whatever preference info is needed to do the exportation. For example:



Export Options

HyperStudio™ Sound v1.0b13

● Uncompressed
○ Compressed 2:1
○ Compressed 8:3

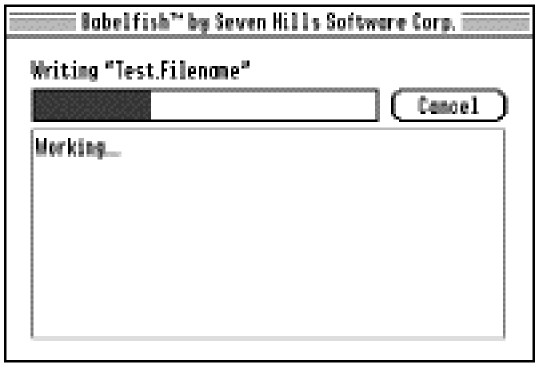
“OK” changed to “Accept”

BFExportThis

Once a file name and location are specified (either through BFPutFile or some other means), the application calls BFExportThis. Babelfish loads the chosen translator along with its stored options, then calls TrStartUp for exportation.

The translator's TrStartUp sets up buffers and variables and creates the file (deleting any existing file first…user permission to replace would have happened at BFPutFile time, and permission is assumed to have been given if the application directly calls BFExportThis).

If the application specified that progress was allowed, Babelfish opens the Progress Window. The Progress Window consists of a thermometer and an area in which the *application* may display messages for the user (during exportation *only the application* may update the message):



Working…

The message area keeps the user occupied/informed during the exportation.

Control is returned to the application. If an error is returned from BFExportThis then the application must cancel its export process. Otherwise, the export process continues…

The application the enters the following loop:

1) The application fills in the appropriate XferRec fields (Status, progress thermometer value, optional user message, data, etc.) and calls BFWrite and BFProgress if the progesss window has been turn on.

2) Babelfish checks Status:
   If "done" or "continue" Babelfish updates the Progress Window.
   If "done" or "abort" Babelfish calls TrShutDown (which closes—and possible deletes—the file) then returns to the application.
   If "continue" Babelfish continues…

3) Babelfish calls the translator's TrWrite. TrWrite processes/writes a chunk of the file, then returns to Babelfish.

4) Babelfish checks Status. If "continue" then return to the application; otherwise it's an "abort" so Babelfish calls TrShutDown (which deletes the export file) then returns to the application.

5) The application checks Status. If "continue" then go to step 1; otherwise it's an "abort" so the application must abort its exporting process.

After the application calls BFWrite with an "abort" or "done" status, the exportation is complete as far as Babelfish is concerned (the translator is shut down, the Progress Window is closed if necessary, etc.).

Definitions and Constants

Documentation Structure

The “Description” section gives a quick overview of the routine’s purpose.

The “Parameters” section shows the `dataIn` and `dataOut` values for the message request calls; or the stack picture for those calls that pass their parameters on the stack.

The “Errors” section states what errors can be returned in the result word of `dataOut`. Message senders must check this result first; if it is non-zero, the remaining data in `dataOut` is invalid.

The “Logic” section sometimes gives an overview of what happens in code. It is not complete (e.g. error checking is omitted) and it’s not intended to list every single step that will be required.

The “Helpful Illustrations/Examples” section contains…Helpful Illustrations and/or Examples!

Miscellaneous

An “application,” as used in this document, refers to *any* program that issues calls to Babelfish (not necessarily a standalone application…it could be an NDA or other compatible environment).

An *import translator* is one that can *send data to* an application. “Importing” is when a translator sends data to an application.

An *export translator* is one that can *receive data from* an application. “Exporting” is when an application sends data to a translator.

Storage Units

Boolean, Byte, Double Long, Fixed, Flag <unit>, Long, Reverse, Word, and so on are all the usual definitions, as defined in Apple’s “Apple II File Type Notes.”

Except where specified otherwise, “String” is a Pascal-type string. This means it’s a length byte followed by *up to* 255 bytes of ASCII data (note that some String fields place a lower limit on the number of characters).

A C1InputString is the same as a WString. It is a length word followed by up to 65,535 bytes of data.

Constants

Error Codes

$0000	bfNoErr	No error
$5300	bfBFNotStarted	Haven’t done BFStartUp
$5301	bfBFBusy	Babelfish is busy
$5302	bfMissingTools	Required tools are not started
$5303	bfNoTransErr	No translators available
$5304	bfTransBusy	Specified translator is busy
$5305	bfNotSupported	Translator function not supported
$5306	bfSupportNotFound	Support routine was not found
$5307	bfBadUserID	You called a routine you weren’t supposed to

Status Codes

{non-zero means stop the process}

$0000	bfContinue	No error
$8000	bfDone	Successful import/export completion
$8001	bfUserAbort	Abort; user requested
$8002	bfBadFileErr	Abort; during import it was discovered the file could not be imported by the translator (either bad structure or turned out not to be the translator’s file)
$8003	bfReadErr	Abort; media error reading from the disk
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Abort; undefined error occurred

The Transfer Record

The transfer record (or XferRec) is always created and disposed of by the application, regardless of whether an import or export process is beginning. The transfer record is updated by the application, Babelfish, and the translator. When BFStartUp is called, one of the things it does is return the size of this record (in bytes). The application must use this returned value to create the transfer record.

The XferRec is the primary means of communication between the application and a translator. Babelfish monitors the activity so it can perform certain housekeeping tasks (e.g. update the Progress Window, shut down a translator when it's done importing, etc.).

$00	ParmCount		Word–Number of Parameters (12)
$02	Status	1	Word–The status of the translation
$04	MiscFlags	2	Flag Word–Flags controlling the translation
$06	DataKinds	3	8 Flag Bytes—Flags for the kind of data
$0E	TransNum	4	Word–The translator number to use
$10	UserID	5	Word–The memory ID assigned to the translator
$12	ProgressAction	6	Flag Word–Flags controlling the progress display
$14	FullTherm	7	Word–The thermometer's scale
$16	CurrentTherm	8	Word–The thermometer's mercury
$18	MsgPtr	9	Long–Pointer to C1inputString of message to display
$1C	DataRecordPtr	10	Long–Pointer to data record
$20	FilePathPtr	11	Long–Pointer to C1inputString of import/export file path
$24	FileNamePtr	12	Long–Pointer to pString of import/export file name

Status

Indicates the current status of the translation. See "Status Codes" in the "Constants" section for a complete list of codes which can be used.

MiscFlags

Various flags controlling the translation:

Bit0	ShowProgessBit	0=suppress progress dialog 1=show progress dialog (the usual and recommended setting)
Bit1	ShowErrorsBit	0=suppress error dialogs 1=show error dialogs (the usual and recommended setting)
Bit2	MatchingBit	0=Loose 1=Standard (the usual and recommend setting)
Bit3	DirectionBit	0=importing 1=exporting
Bit4	SilentModeBit	0=normal operation 1=silent operation

Bits5-15 reserved (must be zero)

Silent Mode: If your translator must ask the user questions during import/export, examine this bit first. If this bit is set you should not ask the user any questions, but instead default to the most logical answers.

DataKinds

While Babelfish itself doesn't treat different classes of data differently, individual translators do. In order to distinguish different kinds of data, all data is categorized into types. Each data type is assigned a single bit out of the 64 bits available in the Kinds of Data flag field:

Data Type	Bit Set	Value
0-No Data	<none>	0
1-Text	0	1
2-Pixel Map	1	2
3-True Color	2	4
4-QDII Pict	3	8
5-Font	4	16
6-Sound	5	32
7-Animation	6	64
8-Publishing	7	128

By using *bits* to indicate data kinds, an application has the option of specifying whether it wants to import one specific kind of data, or that it can import several different kinds of data. For example, when a desktop publishing program asks Babelfish to get a file for importation, the application might specify that it can import text, PixelMaps, and QuickDraw pictures (3 bits are set). Babelfish lets the user select one of those kinds of files. After selection, Babelfish starts up the appropriate translator, and the translator adjusts the Kind to indicate the one specific kind of data the application should prepare to receive (1 bit is set).

TransNum

The TransNum is a relative index number assigned by Babelfish during BFStartup. As the Babelfish universe is being built, each translator is numbered. Babelfish uses the TransNum internally to distinguish between translators.

For applications wishing to use the advanced routines and select a specific translator, the functions BFTransNum2Name and BFName2TransNum may be helpful.

UserID

This is the base ID for the translator to use for all memory requests. The translator must use this ID and must not call for a New ID. Aux IDs may be formed from this base number if the translator desires.

ProgressAction

This word is a collection of bits that Babelfish uses to determine which fields are valid and/or what updating to perform in the Progress Window. Multiple bits may be used on a single call.

Bit0	NewMercury	1=CurrentTherm is valid; update the thermometer's setting 0=Ignore CurrentTherm
Bit1	SetScale	1=FullTherm is valid; set the thermometer's scale 0=Ignore FullTherm
Bit2	NewMessage	1=Update the message using the following bits 0=Leave the same message displayed
Bit3	UserMsg	1=MsgPtr is valid; display this message (and ignore the next bit) 0=Ignore MsgPtr; use the next bit instead
Bit4	StandardMsg	1=Show "Working..." message 0=Show no message (blank)

Bits5-15 reserved (must be zero)

FullTherm

The progress thermometer's scale. When the thermometer is full, CurrentTherm will equal FullTherm.

CurrentTherm

The progress thermometer's mercury (current setting). When the thermometer is full, CurrentTherm will equal FullTherm. *NOTE: CurrentTherm must either stay the same or increase…it must never decrease!*

MsgPtr

During import or export, messages can be displayed in the Progress Window. Do not get carried away with long strings and large fonts as the area for display is limited and does not support scrolling.

To have Babelfish display a standard "Working…" message, set the NewMessage and the StandardMsg bits in ProgressAction.

To display a message, pass a pointer to a WString (a length word followed by the string data bytes). The string data bytes must be in the form of LETextBox2 (see TB1, page 10-44). Also set the NewMessage and the UserMsg bits in ProgressAction.

To display no message at all, set the NewMessage bit and clear the UserMsg and StandardMsg bits in ProgressAction..

To display a new message, pass a pointer to the new string and set the NewMessage and the UserMsg bits in ProgressAction. Babelfish erases the old message and draws the new message.

After drawing the specified string Babelfish clears all of the bits in ProgressAction. This way, if no changes are made to ProgressAction, the message shown will remain the same. This should help prevent the same string from being drawn repeatedly.

DataRecordPtr

This is a pointer to a specific data record. The format of the data record depends upon the particular kind of data being imported/exported. The specific data record formats are defined and discussed in the "Standard Data Format Definitions" section.

FilePathPtr

This is a pointer to the path (C1InputString) of the file currently being imported/exported.

FileNamePtr

This is a pointer to the name (pString) of the file currently being imported/exported.

Babelfish’s Routines

The following sections describe Babelfish’s housekeeping, general, and advanced routines.

When Babelfish completes a routine, it returns the most appropriate error code in the BFresult word in the `dataOut` buffer. If the call has an XferRecPtr as one of its inputs, the Status field will contain the most appropriate status (whether or not an error occurred).

The BFresult and Status work together. Applications should check the BFresult first to determine if a serious error has occurred. If none has, the application should then check the Status and do whatever is appropriate. For example, when an importation is completed successfully, the last BFRead call would not return an error; however, the Status field would indicate that the importation was complete (bfDone).

In order to use Babelfish the following tools must be started: Desk Manager (which includes the 11 tools needed for using NDA’s), Font Manager, List Manager, and Text Edit. Babelfish will start up and shut down Standard File as needed.

Housekeeping Routines

This section describes Babelfish's housekeeping routines.

BFStartUp ($9001)

Description

Starts up Babelfish for use by an application.

NOTE: Your application must make this call before it makes any other Babelfish calls!

NOTE: If this call is successful you ***must*** *balance it with a BFShutDown call at some point!*

NOTE: If BFStartUp returns an error you must ***not*** *call BFShutDown!*

When an application is ready to import or export a file, it must create and dispose of a block of memory the size of XferRecSize.

Parameters

dataIn — Pointer to this structure:

$00	UserID	Word–The memory ID of the application

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code
$04	XferRecSize	Word-Size of the transfer record

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5302	bfMissingTools	Required tools are not started
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

BFShutDown ($9002)

Description

Shuts down Babelfish. You must make this call *(once and only once)* if you successfully called BFStartUp. You must **not** call BFShutDown if you did not call BFStartUp, or if you did call BFStartUp but it returned an error.

Parameters

dataIn — Pointer to this structure:

$00	UserID	Word–The memory ID of the application

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code

Errors

None

BFVersion ($9004)

Description

Returns the version number of Babelfish in Apple IIGS Long Version Format (see Tech Note #100). This call could also be named "askBFAreYouThere".

Parameters

dataIn — NIL

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code
$04	– version –	Long-The version number

Errors

None

General Routines

This section describes Babelfish's general routines. Most applications will use these calls.

BFChooseTrans ($9011)

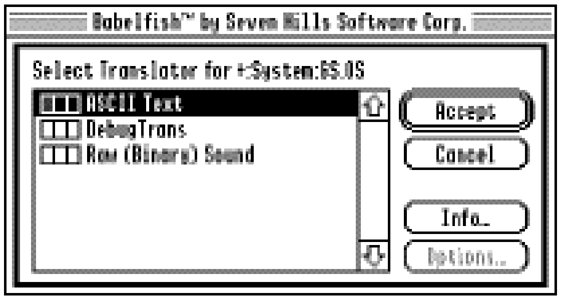
Description

BFChooseTrans is used when an application already knows which file to import or export, but it does not know which translator to use.

Importing

Babelfish first builds a list of non-busy translators capable of importing the given Kinds of data. If at least one translator is found, Babelfish gets information about the given file and calls each translator's TrFilter routine to get a ranking of that translators ability to import the file.

Based upon the ranking and the MiscFlags(MatchingBit) setting, Babelfish creates an alphabetical list of just those translators that can import the file. The translator with the highest ranking becomes the selected item, and is scrolled to if necessary. The following dialog is then presented (the prompt line can be specified by the application; the illustration shows the default prompt line):[2]


Babelfish™ by Seven Hills Software Corp.

Select Translator for *:System:GS.OS

ASCII Text
DebugTrans
Raw (Binary) Sound

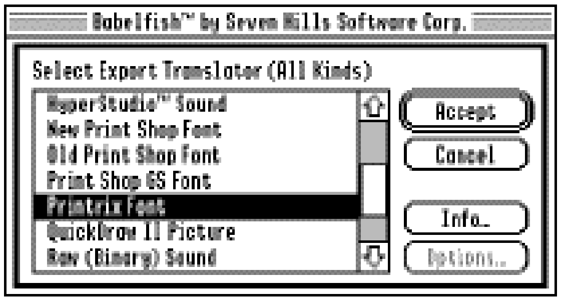
The Info button is always active; the Options button is active only if the highlighted translator has special import options that can be set.

Once the translator is chosen, the user clicks the Accept button and control returns to the application. Typically the application would then begin the import process (*see BFImportThis*).

[2] The "Accept" button is not named "Import" or "Open" because the user is not "Importing" or "Opening" a *translator*…he is simply making a translator choice then Accepting that choice.

Exporting

Babelfish builds an alphabetical list of non-busy translators capable of exporting the given Kind of data. If at least one translator is found, Babelfish selects the first item and presents the following dialog (the prompt line can be specified by the application; the illustration shows the default prompt line):



The Info button is always active; the Options button is active only if the highlighted translator has special export options that can be set.

Once the translator is chosen, the user clicks the Accept[3] button and control returns to the application. Typically the application would then begin the export process (*see BFExportThis*).

Parameters

dataIn — Pointer to this structure:

$00	– xferRecPtr –	Long-Pointer to Transfer Record
$04	– customPtr –	Long-Pointer to the custom record (NIL for defaults)

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Abort; undefined error occurred

[3] Again, the “Accept” button is not named “Export” or “Save” because the user is not “Exporting” or “Saving” a *translator*…he is simply making a translator choice then Accepting that choice.

Notes

NOTE: If customPtr is NIL the default prompt is used.

The customPtr record looks like this:

```
        |------------------------|
$00     |–       ParmCount     –|        Word–Number of Parameters (1)
        |------------------------|
$02     |–                     –|
        |–       promptPtr     –|   1    Long–Pointer to pString of prompt line
        |–                     –|
        |------------------------|
```

NOTE: If any field is NIL Babelfish's default wording is used.

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

MiscFlags–ShowErrorsBit (whichever you desire)

MiscFlags–MatchingBit (only required if importing; ignored if exporting)

MiscFlags–DirectionBit (whether importing or exporting)

DataKinds (if importing, several bits may be set; if exporting only one bit should be set)

FilePathPtr (only required if importing; ignored if exporting)

FileNamePtr (only required if importing; ignored if exporting)

Before calling TrImportOptions or TrExportOptions, Babelfish also makes these fields valid:
UserID

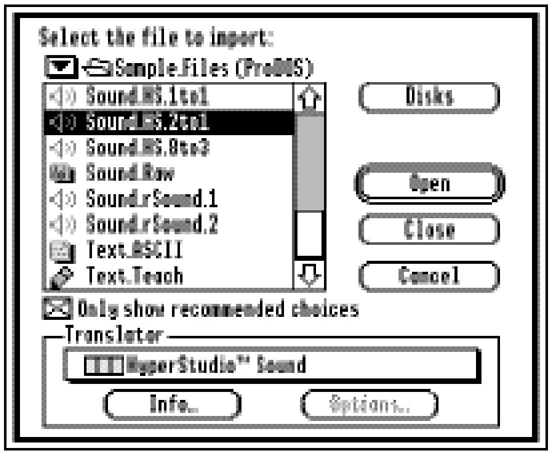
Before returning, Babelfish also makes these fields valid:
Status

TransNum (set to the translator the user selected)

BFGetFile ($9012)

Description

If there is at least one non-busy translator capable of importing one of the given Kinds of data, this routine presents a custom SFPGetFile2 dialog box. The prompt line and Open button name can be specified by the application.



Using this dialog the user selects the file he wishes to import, then the translator he wants to use. Optionally he may view an About box or change Options for any displayed translator. Once the file and translator are chosen, the user clicks the Open button and control returns to the application. Typically the application would then begin the import process (*see BFImportThis*).

NOTE: If you do not install the SFUtility INIT, two cosmetic problems are possible with the BFGetFile dialog. The problems are purely cosmetic and will not cause a crash or any other user error: (1) When the Volumes list is displayed, the popup may remain enabled. (2) When no files are displayed (empty directory or a directory in which no files are importable), the popup may remain enabled with the full list of translators that can import the given Kinds of Data shown, or, in some cases, the first translator in the list is shown and the popup looks enabled but it really is not (won't pop up).

Parameters

dataIn — Pointer to this structure:

```
     |————————————————————————|
$00  |–      xferRecPtr      –|  Long-Pointer to Transfer Record
     |————————————————————————|
$04  |–      customPtr       –|  Long-Pointer to the custom record (NIL for defaults)
     |————————————————————————|
$08  |–      replyPtr        –|  Long–Pointer to SF new-style reply record
     |————————————————————————|
```

dataOut — Pointer to this structure:

```
     |————————————————————————|
$00  |       recvCount        |  Word-Number of times the request was received
     |————————————————————————|
$02  |       BFresult         |  Word-Error code
     |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Notes

NOTE: If customPtr is NIL the default prompt and button name are used.

The customPtr record looks like this:

```
       |————————————————————|
  $00  |–     ParmCount    –|       Word–Number of Parameters (2)
       |————————————————————|
  $02  |–                  –|
       |–     promptPtr    –|  1    Long–Pointer to pString of prompt line
       |–                  –|
       |————————————————————|
  $06  |–                  –|
       |–  importButtonPtr –|  2    Long–Pointer to pString of “Open” button name
       |–                  –|
       |————————————————————|
```

NOTE: If any field is NIL Babelfish’s default wording is used.

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

MiscFlags–ShowErrorsBit (whichever you desire)

DataKinds (one or more bits should be set)

Before calling TrImportOptions, Babelfish also makes these fields valid:
UserID

Before returning, Babelfish also makes these fields valid:
Status

MiscFlags–MatchingBit (set to user’s choice)

MiscFlags–DirectionBit (set to importing)

TransNum (set to the translator the user selected)

BFImportThis ($9013)

Description

An application calls BFImportThis when it has a file it wants to import and when it knows which translator to use.

Babelfish loads the translator[4] and tells it to open the given file for importing. If the file is successfully opened and prepared for importing, control is returned to the application with a good result. Otherwise, an appropriate error is returned.

Upon a successful return, the application should examine the Kind field to discover what specific Kind of information will be returned, so it can prepare itself to receive that Kind of data.

Parameters

dataIn — Pointer to this structure:

```
        |————————————————————————|
  $00   |–       xferRecPtr     –|  Long-Pointer to Transfer Record
        |————————————————————————|
```

dataOut — Pointer to this structure:

```
        |————————————————————————|
  $00   |        recvCount       |  Word-Number of times the request was received
        |————————————————————————|
  $02   |        BFresult        |  Word-Error code
        |————————————————————————|
```

Errors

$0000	bfNoErr	No error; ready for BFRead calls
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available
$5304	bfTransBusy	The specified translator is busy

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8002	bfBadFileErr	Abort; during import it was discovered the file could not be imported by the translator (either bad structure or turned out not to be the translator’s file)
$8003	bfReadErr	Abort; media error reading from the disk
$80FF	bfAbortErr	Other error

Notes

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

MiscFlags–ShowProgressBit (whichever you desire)

MiscFlags–ShowErrorsBit (whichever you desire)

[4] BFImportThis only verifies that the given TransNum exists and is not busy—it does not double-check to make sure the specified translator can import the specified Kinds of data, and in fact doesn't check to see if the translator even supports importing.

DataKinds (one or more bits should be set)

TransNum (already set if BFGetFile or BFChooseTrans was called; otherwise the application must set this field)

FilePathPtr (application sets this to path of chosen file, as taken from reply record)

FileNamePtr (application sets this to name of chosen file, as taken from reply record)

Before calling TrStartup, Babelfish also makes these fields valid:
MiscFlags–DirectionBit (set to importing)

UserID

Before returning, the translator also makes these fields valid:
Status

DataKinds (sets only one bit to indicate the kind of data that will be returned)

ProgressAction

FullTherm

CurrentTherm (zero)

MsgPtr

DataRecordPtr (either here or during BFRead)

Before returning, Babelfish also makes these fields valid:
Status

BFRead ($9014)

Description

The application calls BFRead each time it is ready to accept more data.

Before calling BFRead the application fills in the appropriate fields of the XferRec.

If the application doesn't instruct Babelfish to abort, Babelfish calls the specific translator's TrRead routine.

If no error occurs, the application stores the received data in a temporary location and continues making BFRead calls.

If Babelfish discovers an error or "done" status, it shuts down the translator and returns the error to the application. If the application receives an error, any received data should be discarded as invalid; if the application receives a "done" status the received data is complete and ready to use. In either case, the application makes no further calls to Babelfish for this import.

Parameters

dataIn — Pointer to this structure:

```
      |—————————————————————|
$00   |–     xferRecPtr    –|  Long-Pointer to Transfer Record
      |—————————————————————|
```

dataOut — Pointer to this structure:

```
      |—————————————————————|
$00   |      recvCount      |  Word-Number of times the request was received
      |—————————————————————|
$02   |      BFresult       |  Word-Error code
      |—————————————————————|
```

Errors

$5301	bfBFBusy	Babelfish is busy

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8002	bfBadFileErr	Abort; during import it was discovered the file could not be imported by the translator (either bad structure or turned out not to be the translator's file)
$8003	bfReadErr	Abort; media error reading from the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Notes

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

DataRecordPtr (depending upon requirements of the specific data record format)

Before calling TrRead, Babelfish also makes these fields valid:

UserID

Status

Before returning, the translator also makes these fields valid:

Status

ProgressAction

CurrentTherm

MsgPtr

DataRecordPtr

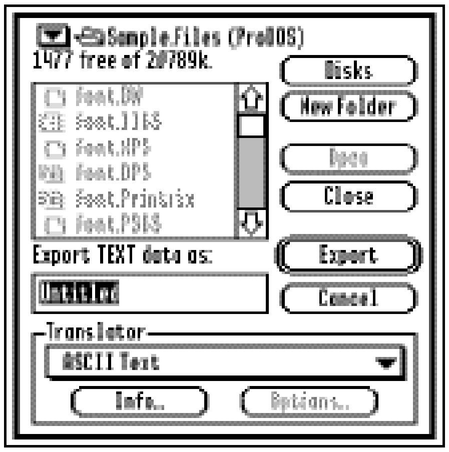
Before returning, Babelfish also makes these fields valid:

Status

BFPutFile ($9015)

Description

If there is at least one non-busy translator capable of exporting the given Kind of data, this routine presents a custom SFPutFile2 dialog box. The prompt line, default filename, and Export button name can be specified by the application (the illustration shows the default prompt and button name):



All applicable translators are displayed in alphabetical order. Optionally the user may view Info or change Options for any listed translator. Once the file location and name are specified, the user clicks the Export button and control returns to the application. Typically the application would then begin the export process (*see BFExportThis*).

Parameters

dataIn — Pointer to this structure:

$00	– xferRecPtr –	Long-Pointer to Transfer Record
$04	– customPtr –	Long-Pointer to the custom record (NIL for defaults)
$08	– replyPtr –	Long–Pointer to SF new-style reply record

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Notes

NOTE: If customPtr is NIL the default prompt, button name, and filename are used.

The customPtr record looks like this:

```
      |————————————————————————|
$00   |–       ParmCount      –|      Word–Number of Parameters (3)
      |————————————————————————|
$02   |–                      –|
      |–       promptPtr      –|  1   Long–Pointer to pString of prompt line
      |–                      –|
      |————————————————————————|
$06   |–                      –|
      |–    exportButtonPtr   –|  2   Long–Pointer to pString of “Export” button name
      |–                      –|
      |————————————————————————|
$0A   |–                      –|
      |–      filenamePtr     –|  3   Long–Pointer to C1Input String of default filename
      |–                      –|
      |————————————————————————|
```

NOTE: If any field is NIL Babelfish’s default wording is used.

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

MiscFlags–ShowErrorsBit (whichever you desire)

DataKinds (only one bit should be set)

Before calling TrExportOptions, Babelfish also makes these fields valid:
UserID

Before returning, Babelfish also makes these fields valid:
Status

MiscFlags–DirectionBit (set to exporting)

TransNum (set to the translator the user selected)

BFExportThis ($9016)

Description

An application calls BFExportThis when it has a file it wants to export and when it knows which translator to use.

Babelfish loads the translator[5] and tells it to create a file for exporting. If this is successful, control is returned to the application with a good result. Otherwise, an appropriate error is returned.

[5] BFExportThis only verifies that the given TransNum exists and is not busy—it does not double-check to make sure the specified translator can export the specified Kinds of data, and in fact doesn't check to see if the translator even supports exporting.

Parameters

dataIn — Pointer to this structure:

```
       |———————————————————————|
  $00  |–      xferRecPtr     –|  Long-Pointer to Transfer Record
       |———————————————————————|
```

dataOut — Pointer to this structure:

```
       |———————————————————————|
  $00  |       recvCount       |  Word-Number of times the request was received
       |———————————————————————|
  $02  |       BFresult        |  Word-Error code
       |———————————————————————|
```

Errors

$0000	bfNoErr	No error; ready for BFWrite calls
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available; don’t attempt any BFWrite calls!
$5304	bfTransBusy	The specified translator is busy; don’t attempt any BFWrite calls!

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Notes

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

MiscFlags–ShowProgressBit (whichever you desire)

MiscFlags–ShowErrorsBit (whichever you desire)

DataKinds (only one bit should be set)

TransNum (already set if BFPutFile or BFChooseTrans was called; otherwise the application must set this field)

ProgressAction

FullTherm

CurrentTherm (zero)

MsgPtr

DataRecordPtr (either here or before BFWrite)

FilePathPtr (set to path of chosen file, as taken from reply record)

FileNamePtr (set to name of chosen file, as taken from reply record)

Before calling TrStartup, Babelfish also makes these fields valid:

MiscFlags–DirectionBit (set to exporting)

UserID

Before returning, the translator also makes these fields valid:
Status

Before returning, Babelfish also makes these fields valid:
Status

BFWrite ($9017)

Description

The application calls BFWrite each time it has more data to send.

Before calling BFWrite the application fills in the appropriate fields of the XferRec.

If the application doesn't instruct Babelfish to stop the process (abort, done, etc.), Babelfish calls the specific translator's TrWrite routine to write the data to disk.

If no error occurs, the application continues making BFWrite calls until all the data is exported.

If Babelfish discovers an error or "done" status, it shuts down the translator and returns to the application. If the translator receives an error, the export file should be deleted. When done, the application makes no further calls to Babelfish for this export.

Parameters

dataIn — Pointer to this structure:

```
      |——————————————————————|
$00   |–      xferRecPtr    –|  Long-Pointer to Transfer Record
      |——————————————————————|
```

dataOut — Pointer to this structure:

```
      |——————————————————————|
$00   |      recvCount       |  Word-Number of times the request was received
      |——————————————————————|
$02   |       BFresult       |  Word-Error code
      |——————————————————————|
```

Errors

$0000	bfNoErr	No error; ready for BFWrite calls
$5301	bfBFBusy	Babelfish is busy

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	User clicked Cancel
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Notes

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue or bfDone)

ProgressAction

CurrentTherm

MsgPtr

DataRecordPtr

Before calling TrWrite, Babelfish also makes these fields valid:
UserID

Status

Before returning, Babelfish also makes these fields valid:
Status

BFProgress ($9018)

Description

This supporting call may be made either by applications when exporting (just before or after calling BFWrite) or by translators when they are importing (in their TrRead routine). Babelfish uses information in the Transfer Record to update the progress window.

If the application has suppressed the progress window (presumably to show its own), Babelfish will rebroadcast this message from the translator, directing the request to the application’s ID number, with a `sendHow` value of `sendToUserID+stopAfterOne`. *NOTE: The application may ignore this message completely and need not return any code as Babelfish ignores all results, including errors—this is merely a courtesy relay for applications that want to draw their own progress window.*

In this case, the application must treat the transfer record exactly as Babelfish would: Use only the ProgressAction, FullTherm, CurrentTherm, and MsgPtr fields, then clear all the bits in ProgressAction before returning *(see The Transfer Record section for more information on using these fields)*. Make no other changes to the Transfer Record.

Parameters

dataIn — Pointer to this structure:

```
      |————————————————————————|
$00   |–       xferRecPtr     –|  Long-Pointer to Transfer Record
      |————————————————————————|
```

dataOut — Pointer to this structure:

```
      |————————————————————————|
$00   |        recvCount       |  Word-Number of times the request was received
      |————————————————————————|
$02   |        BFresult        |  Word-Error code
      |————————————————————————|
```

Errors
none

Status
no change

Notes

The only fields used in the transfer record are ProgressAction, FullTherm, CurrentTherm, and MsgPtr. All other fields are ignored and remain unchanged. For details on setting these fields, see the section on The Transfer Record.

Advanced Routines

This section describes Babelfish's advanced routines. With them, an application can perform translations without ever having a Babelfish window or dialog presented…the "stealth" mode. *NOTE: It is highly recommended that an application use the standard Babelfish calls in order to present consist interface to the user.*

BFMatchKinds ($9021)

Description

This call returns a handle to a list of translator numbers that can import or export the given Kinds of data. As usual, it is up to the caller to dispose of the handle when finished with it.

Parameters

dataIn — Pointer to this structure:

```
      |————————————————————————|
$00   |–      xferRecPtr      –|  Long-Pointer to Transfer Record
      |————————————————————————|
```

dataOut — Pointer to this structure:

```
      |————————————————————————|
$00   |       recvCount        |  Word-Number of times the request was received
      |————————————————————————|
$02   |       BFresult         |  Word-Error code
      |————————————————————————|
      |–      transListH      –|  Long–Handle to the list of translator numbers
      |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

The returned transListH is in the following format:

```
      |————————————————————————|
$00   |–      transCount      –|  Word–The number of items in transArray
      |————————————————————————|
$02   :       transArray       :  Array–Array of translators
      |————————————————————————|
```

Each transArray element is formatted as follows:

```
      |————————————————————————|
$00   |–       transNum       –|  Word–The matching translator number
      |————————————————————————|
```

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

MiscFlags–DirectionBit (whichever you desire)

DataKinds (one or more bits should be set)

BFMatchFile ($9022)

Description

This call returns a handle to a list of translator numbers and rankings of translators that can import the given file (this call does not apply for exporting). As usual, it is up to the caller to dispose of the handle when finished with it.

Parameters

dataIn — Pointer to this structure:

```
        |——————————————————|
  $00   |–    xferRecPtr  –|  Long-Pointer to Transfer Record
        |——————————————————|
```

dataOut — Pointer to this structure:

```
        |——————————————————|
  $00   |     recvCount    |  Word-Number of times the request was received
        |——————————————————|
  $02   |     BFresult     |  Word-Error code
        |——————————————————|
        |–    transListH  –|  Long–Handle to list of translator numbers and rankings
        |——————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No translators available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

The returned transListH is in the following format:

```
        |——————————————————|
  $00   |–    transCount  –|  Word–The number of items in transArray
        |——————————————————|
  $02   :     transArray   :  Array–Array of translators
        |——————————————————|
```

NOTE: The translator array is sorted from “best ranking” to “worst ranking”. If two rankings match then each translator is equally capable; the translator name is used as the tie-breaker.

Each transArray element is formatted as follows:

```
        |——————————————————|
  $00   |–    transNum    –|  Word–The matching translator number
        |——————————————————|
  $02   |–    ranking     –|  Word–The ability of this translator to interpret the file
        |——————————————————|
```

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

DataKinds (one or more bits should be set)

FilePathPtr (set to path of chosen file)

FileNamePtr (set to name of chosen file)

BFTransNum2Name ($9023)

This call returns a handle to a particular translator’s name. As usual, it is up to the caller to dispose of the handle when finished with it.

Parameters

dataIn — Pointer to this structure:

```
      |————————————————————|
$00   |–     xferRecPtr   –|  Long-Pointer to Transfer Record
      |————————————————————|
```

dataOut — Pointer to this structure:

```
      |————————————————————|
$00   |     recvCount      |  Word-Number of times the request was received
      |————————————————————|
$02   |      BFresult      |  Word-Error code
      |————————————————————|
      |–     trNameH      –|  Long–Handle to pString containing translator’s name
      |————————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No such translator available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

TransNum (must be a specific translator)

BFTransName2Num ($9024)

This call returns the TransNum of the first non-busy translator that exactly matches a given name (including case), matches the given Kinds, and supports import/export.

NOTE: The search is case sensitive and the name must match exactly.

Parameters

dataIn — Pointer to this structure:

```
     |———————————————————|
$00  |–    xferRecPtr   –|  Long-Pointer to Transfer Record
     |———————————————————|
$04  |–     namePtr     –|  Long-Pointer to the pString containing the translator name
     |———————————————————|
```

dataOut — Pointer to this structure:

```
     |———————————————————|
$00  |     recvCount     |  Word-Number of times the request was received
     |———————————————————|
$02  |     BFresult      |  Word-Error code
     |———————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No such translator available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

MiscFlags (DirectionBit)

DataKinds

BFGetTrData ($9025)

This call returns a handle to information about a particular translator. As usual, it is up to the caller to dispose of the handle when finished with it.

Parameters

dataIn — Pointer to this structure:

$00	xferRecPtr	Long-Pointer to Transfer Record

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code
	trDataH	Long–Handle to data about the translator

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No such translator available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

The returned information is in the following format: ***(see TrData for more details)***

$00	trRevision	Word–The revision of this structure ($0000)
$02	trGeneralFlags	Flag Word–Operating mode capabilities
$04	trImportFlags	Flag Word–Importing capabilities and options
$06	trImportKinds	8 Flag Bytes–Flags for the kind of importable data
$0E	trExportFlags	Flag Word–Exporting capabilities and options
$10	trExportKinds	8 Flag Bytes–Flags for the kind of exportable data

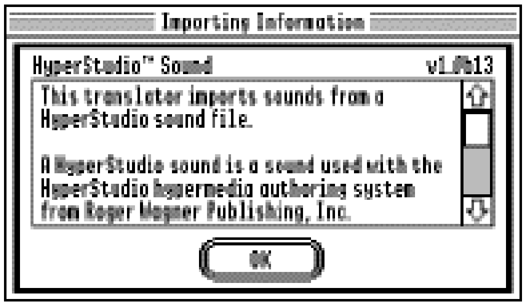
Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

TransNum (must be a specific translator)

BFShowInfo ($9026)

Description

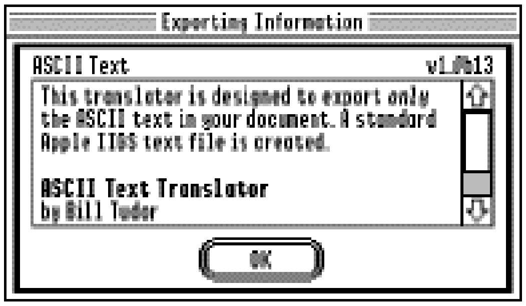
Given the translator number and whether the application is importing or exporting, BFShowInfo will display the “Importing Information” or “Exporting Information” for the given translator. The information is presented by Babelfish in a moveable, modal window. For example:



HyperStudio™ Sound v1.0b13

This translator imports sounds from a HyperStudio sound file.

A HyperStudio sound is a sound used with the HyperStudio hypermedia authoring system from Roger Wagner Publishing, Inc.



ASCII Text v1.0b13

This translator is designed to export only the ASCII text in your document. A standard Apple IIGS text file is created.

ASCII Text Translator
by Bill Tudor

Parameters

dataIn — Pointer to this structure:

```
        |————————————————————————|
  $00   |–       xferRecPtr     –|  Long-Pointer to Transfer Record
        |————————————————————————|
```

dataOut — Pointer to this structure:

```
        |————————————————————————|
  $00   |        recvCount       |  Word-Number of times the request was received
        |————————————————————————|
  $02   |        BFresult        |  Word-Error code
        |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No such translator available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

no change

Notes

Before calling this routine, the application must make these Transfer Record fields valid:
Status (must be bfContinue)

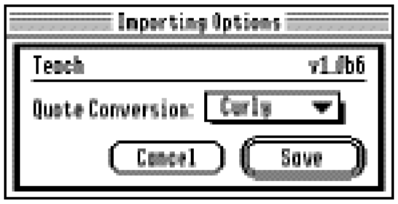
MiscFlags–DirectionBit (whichever you desire)

TransNum (must be a specific translator)

BFDoOptions ($9027)

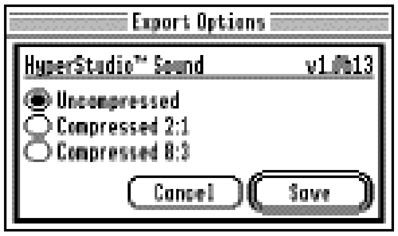
Description

Given the translator number and whether the application is importing or exporting, BFDoOptions will display the “Importing Options” or “Exporting Options” for the given translator. Babelfish calls TrImportOptions or TrExportOptions for the translator to present its dialog to the user. For example:



Teach v1.0b6

Quote Conversion: Curly



HyperStudio™ Sound v1.0b13

[✓ Uncompressed]
Compressed 2:1
Compressed 8:3

Parameters

dataIn — Pointer to this structure:

```
        |————————————————————————|
  $00   |–       xferRecPtr     –|  Long-Pointer to Transfer Record
        |————————————————————————|
```

dataOut — Pointer to this structure:

```
        |————————————————————————|
  $00   |        recvCount       |  Word-Number of times the request was received
        |————————————————————————|
  $02   |        BFresult        |  Word-Error code
        |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5301	bfBFBusy	Babelfish is busy
$5303	bfNoTransErr	No such translator available

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8001	bfUserAbort	Abort; user clicked Cancel
$8005	bfMemErr	Abort; memory error

Notes

Before calling this routine, the application must make these Transfer Record fields valid:

Status (must be bfContinue)

MiscFlags–DirectionBit (whichever you desire)

TransNum (must be a specific translator)

Before calling TrImportOptions or TrExportOptions, Babelfish also makes these fields valid:

UserID

BFLoadData ($9028)

Description

Important: This call can be made only be translators!

This call returns a handle containing preferences for the calling translator. If the handle is null (or if an error is returned) then no preferences were loaded and the translator should use its default settings.

Parameters

dataIn — Pointer to this structure:

```
      |————————————————————|
$00   |–      userID      –|  Word-Translator’s user ID
      |————————————————————|
```

dataOut — Pointer to this structure:

```
      |————————————————————|
$00   |     recvCount      |  Word-Number of times the request was received
      |————————————————————|
$02   |     BFresult       |  Word-Error code
      |————————————————————|
$04   |–    dataHandle    –|  Long-Handle containing the data
      |————————————————————|
```

Errors

$0000	bfNoErr	No error
$5307	bfBadUserID	You called a routine you weren’t supposed to

GS/OS errors returned unchanged

Notes

BFSaveData ($9029)

Description

Important: This call can be made only be translators!

This call saves preferences for the calling translator.

Parameters

dataIn — Pointer to this structure:

```
       |————————————————————|
  $00  |–      UserID      –|  Word-Translator’s user ID
       |————————————————————|
  $02  |–    dataHandle    –|  Long-Handle containing the data
       |————————————————————|
```

dataOut — Pointer to this structure:

```
       |————————————————————|
  $00  |     recvCount      |  Word-Number of times the request was received
       |————————————————————|
  $02  |      BFresult      |  Word-Error code
       |————————————————————|
```

Errors

$0000	bfNoErr	No error
$5307	bfBadUserID	You called a routine you weren’t supposed to

GS/OS errors returned unchanged

Notes

BFLoadSupport ($902A)

Description

Translators and applications that wish to use common support routines will first need to load them. This Babelfish call will simplify that task.

When Babelfish receives this call, it loads the named file from the Support folder, calls its SrInit function, and returns the memory ID of the file. If the file has already been loaded by another caller, Babelfish increments an internal counter, skips the load and initialization, and just returns the memory ID.

Translators and applications may then send requests to the support file using either `sendToName` or `sendToUserID` (using `sendToUserID` is recommended for efficiency). The first message sent to the support file must be to its SrStartup function and the last message must be to its SrShutdown function *(see the Translation Support Routines section for more information)*.

Unless this call is unsuccessful, it ***must*** *be balanced with a BFUnloadSupport call.*

Parameters

dataIn — Pointer to this structure:

```
       |————————————————————|
```

$00	namePtr	Long-Pointer to Pstring containing the file’s request name

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code
$04	SupportID	Word-Support file’s ID number

Errors

$0000	bfNoErr	No error
$5306	bfSupportNotFound	Support routine was not found

GS/OS errors returned unchanged

Notes

BFUnloadSupport ($902B)

Description

When the translator or application is finished with a support file (most likely as part of TrShutdown), it must release the support file. *Translators and applications must balance each successful BFLoadSupport call with a BFUnloadSupport call.*

Babelfish keeps an internal counter of the number of callers that requested the support file to be loaded. When the last caller requests the support file to be unloaded, Babelfish removes the support file’s message handler and disposes of its memory. *It is imperative that the support file user call the SrShutdown function prior to unloading it.*

Parameters

dataIn — Pointer to this structure:

$00	namePtr	Long-Pointer to Pstring containing the file’s request name

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	BFresult	Word-Error code

Errors

$0000	bfNoErr	No error

GS/OS errors returned unchanged

Notes

The Translator Format

Translators are located in the *:System:SHS.Babelfish folder. The filetype is $BE (Translator) and the auxtype is $4003 (Babelfish translator). *NOTE: Users can "inactivate" a translator by checking a box in the Finder, which sets the high bit of the auxiliary type. Babelfish respects the users wishes and ignores inactive files.*

When the translator receives control, the direct page and data bank are not set. The translator must set these registers as needed, but need not restore them upon return. Direct page space is not provided for the translator's use; the translator may use the stack or request and maintain its own space from the Memory Manager.

Babelfish loads a translator from disk, assigns a user ID, and calls Resource Startup at BFImportThis or BFExportThis time (before TrInit and TrStartUp are called by Babelfish). This makes the translators independant resource applications, which they remain throughout the Read/Write life cycle. When the Read/Write process is complete (or terminated due to an error), Babelfish calls Resource Shutdown on the translator and completely purges the translator from memory. *NOTE: This does **not** apply to the TrFilter routine; it **does** apply to the ExportOptions routine.*

Communication with the translator's main functions is by IPC. Babelfish will fill in `dataIn` and call SendRequest. The translator's message handler should route the call according to the request code to one of its function handlers. When the translator completes its action, it should ensure that the necessary fields in the Transfer Record are valid and that the TRresult code is appropriate (Babelfish initializes it to zero).

Babelfish communicates with translators by calling SendRequest with the proper request code, directing the request to the translator's ID number, with a `sendHow` value of `sendToUserID + stopAfterOne`.

The translator's UserID is provided in the XferRec and is to be used for all memory requests (aux IDs may also be used). You cannot call DisposeAll on the UserID, but you can on your Aux IDs.

When a translator function is called, the translator's resource fork is open "read only" (the only changes that a translator needs to save are user options, and they must be saved in a separate preference file).

The translator's resource fork is set as the current resource file, and the search path includes only it and System Resources (which means that the translator will not be able to interfere with other resources of Babelfish, the application, or any other translator). If necessary, a translator may load and manage its own resources, provided they are found in the current search path.

Translator Structure

Translators consist of several required and optional resources. The data fork is not used.

NOTE: The number in parentheses is the required resource ID number.

Translators must contain all of these resources:

Name	Type(ID)	Description
TrVersion	rVersion(1)	This resource is used by the Finder during Get Info. The version number and name are also used by Babelfish when showing the About dialog. *See GS Tech Note #76.*
TrData	$5472(1)	A custom resource collection of detailed info about the translator.

TrInit	rCodeResource(1)	Code that installs the message handler and function router.

All translators must contain the following functions:

Name	Type(ID)	Description
TrStartUp	required	Code that initializes buffers, variables, etc. in preparation to read or write a file.
TrShutDown	required	Code that reverses everything done in TrStartUp.

The following function is required if the translator supports importing:

Name	Type(ID)	Description
TrRead	optional	Code that performs importing.

The following function is required if the translator supports exporting:

Name	Type(ID)	Description
TrWrite	optional	Code that performs exporting.

The following resources are required if the translator supports importing:

Name	Type(ID)	Description
TrFilter	rCodeResource(4)	Code that is structured like a new-style SF filter procedure for use during SFPGetFile2. *See also GS Tech Note #86.*
TrImportOptions	rCodeResource(2)	Code that presents an “Options” dialog to set importing options.
TrImportInfo	rText(2)	Text that describes the importing options (if any) and contact information.

The following resources are required if the translator supports exporting:

Name	Type(ID)	Description
TrExportOptions	rCodeResource(3)	Code that presents an “Options” dialog to set exporting options.
TrExportInfo	rText(3)	Text that describes the exporting options (if any) and contact information.

The following resources are optional (but recommended):

Name	Type(ID)	Description
TrAbout	rComment(1)	This resource is used by the Finder during Get Info (on the Comment page). *See GS Tech Note #76.*
TrCantLaunch	rComment(2)	This resource is used by the System 6 Finder when it cannot launch a file. *See “Programmer’s Reference for System 6.0”, pp 364 & 429.*
TrImportInfoStyle	rStyleBlock(2)	Optional TE Style block for the TrImportInfo rText(2) resource. If this exists, TrImportInfo will be stylized; otherwise it appears in Shaston 8.
TrExportInfoStyle	rStyleBlock(3)	Optional TE Style block for the TrExportInfo rText(3) resource. If this exists, TrExportInfo will be stylized; otherwise it appears in Shaston 8.

TrVersion

Resource Type: rVersion ($8029) Resource ID: $00000001 Attributes: none

This resource contains a long word version number which is used internally by Babelfish.

It also contains the translator name which is displayed in the “select translator” window. *NOTE: The list which displays the translator names is limited in width so you should test to be sure the entire name can be displayed without truncation.*

TrData

Resource Type: rData ($5472) Resource ID: $00000001 Attributes: none

This resource contains detailed information about the translator and its capabilities.

The following information is stored in the resource:

Revision	Word	Used by Babelfish to verify future compatibility. This revision is $0000. If we add new fields to this structure in the future, Babelfish will use this field to determine whether the translator supports the new features.
GeneralFlags	FlagWord	Bit0: Can operate in 320 mode Bit1: Can operate in 640 mode Bits2-15: Reserved; must be zero
ImportFlags	Flag Word	Bit0: Translator can import Bit1: Has importing options Bits2-15: Reserved; must be zero
ImportKinds	8 Flag Bytes	The Kinds of data this translator can import
ExportFlags	Flag Word	Bit0: Translator can export Bit1: Has exporting options Bits2-15: Reserved; must be zero
ExportKinds	8 Flag Bytes	The Kinds of data this translator can export

TrAbout

Resource Type: rComment ($802A) Resource ID: $00000001 Attributes: none

This “Finder Info” resource contains general information about the translator (what it converts, whether or not it supports importing, exporting, or both, and so on).

For example, the text might be something like this:

```
This translator reads and writes ASCII text files.  It can also be used to extract text from non-
text files.

Copyright 1991-1998 by Seven Hills Software Corporation
```

This text is shown by the Finder (on the Comments page) when the user selects “Icon Info” for the translator file. It should be formatted to fit the Comments page space.

TrCantLaunch

Resource Type: rComment ($802A) Resource ID: $00000002 Attributes: none

This resource contains a standard message that is displayed if the user tries to open a translator from the Finder. The suggested text is:

```
This is a translator and can’t be launched.  It is a Seven Hills Software “Babelfish” extension
that reads/writes a special data format.  To use it, put it in the “*:System:SHS.Babelfish”
folder and open an application that uses Babelfish.
```

TrImportOptions

Resource Type: rCodeResource ($8017) Resource ID: $00000002 Attributes: locked, convert

Description

This code resource presents a dialog box in which the user sets importing options. If there are no import options this resource does not exist (and bit 1 of ImportFlags is clear).

For consistency with Babelfish windows and all translators from Seven Hills, the translator should present its options in a moveable modal window. The title “Importing Options” should be used as the window’s title, and the translator name and version number (as specified in the TrVersion resource) should be displayed. See “Overview—Using Babelfish” for examples.

At the bottom of the dialog should be a “Cancel” and a default “Accept” button. If the user hits “Cancel” then no changes are made; if the user hits “Accept” then the options are saved by Babelfish. *NOTE: The Accept button should not be named “Import” or “Open” because the user is not “importing” or “opening” the translator or its options…the user is simply “accepting” the displayed options.*

Parameters

Stack before call

```
        |      Previous contents       |
        |——————————————————————————————|
        |–        xferRecPtr          –|  Long–Pointer to Transfer Record
        |——————————————————————————————|
        |                              |  <—SP
```

Stack after call

```
        |      Previous contents       |
        |——————————————————————————————|
        |                              |  <—SP
```

Errors

$0000 bfNoErr	No error

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000 bfContinue	No error
$8005 bfMemErr	Abort; memory error
$80FF bfAbortErr	Other error

Logic

```
Prior to calling displaying the dialog, the translator should call BFLoadData to request any
    previously-saved import options from a preference file.  If there is any error in loading the
    file, the translator should set up the dialog using default choices.
The preference file contains data previously stored by the translator, and that data should be used
    to set up the options dialog.  The format of this data is entirely up to the translator.  It is
    recommended that the translator include some signature or revision indicator that it can use to
    validate the data (if the validation fails, use the default settings).
If the user changes any settings and clicks Accept, the translator should make the necessary changes
    in the data and call BFSaveData to save a new preference file to disk.
```

Helpful Illustrations/Examples

TrImportInfo

Resource Type: rText ($8016) Resource ID: $00000002 Attributes: none
Optional Resource: rStyleBlock ($8012) Resource ID: $00000002 Attributes: none

This text is displayed when the user selects the Info button in any translator-selection dialog. It gives general information about the translator, and specifically describes the importing options (if any). *NOTE: The translator's name and version are displayed by Babelfish; to avoid potential conflicts the version number should not be included in the text itself.*

The text is displayed in a TextEdit control and is scrollable. If the corresponding rStyleBlock resource exists the text is styled; otherwise the text appears in Shaston 8.

One example of a translator that does **not** have any importing options…

```
This translator supports importing, but no importing options are required.

Technical support and information about new versions of this translator can be obtained by
contacting:

Seven Hills Solutions Specialists, 1254 Ocala Road, Tallahassee, FL  32304.
Technical help: support@sevenhills.com
Everything else: sales@sevenhills.com

Copyright 1991-1998 by Seven Hills Solutions Specialists
```

One example of a translator that **does** have importing options…

```
You can specify the following options before importing:

CONVERT LINES TO PARAGRAPHS
Blah blah blah blah blah

REPLACE CR/LF WITH JUST CR
Blah blah blah blah blah

...and so on...

• • • • • • • • • •

Technical support and information about new versions of this translator can be obtained by
contacting:

Seven Hills Solutions Specialists, 1254 Ocala Road, Tallahassee, FL  32304.
Technical help: support@sevenhills.com
Everything else: sales@sevenhills.com

Copyright 1991-1998 by Seven Hills Solutions Specialists
```

TrFilter

Resource Type: rCodeResource ($8017) Resource ID: $00000004 Attributes: locked, convert

NOTE: This code resource should be as small as possible because is loaded into memory when Babelfish is started and is not released until Babelfish is shut down.

Description

When Babelfish must decide which translators can handle importing a particular file, it calls the filter procedure for each of the non-busy translators that support importing the given Kind of data. The translator's filter procedure determines whether it can deal with the file and returns a rating of its ability. *TIP: If the translator supports HFS or AppleShare files, the Type and Creator may be found in the OptionsList.*

There are three basic responses from the filter procedure:

- **No($00)** means the translator cannot import this file.
- **GenericYes($01 or $02)** means the translator can import the file, but it's not a specific translator (e.g. a text HexDump or graphic Histogram translator would fall into this "generic" category). An $01 match is very generic (histogram, hexdump, ascii characters from a BIN file); an $02 match is slightly more useful (e.g. ascii characters from a TXT file).
- **SpecificYes($03 or $04)** means you're a translator designed to import a specific kind of file (not generic), and the file you're looking at appears to be one of yours. An $03 match means the file is probably yours, but not positive (e.g. single hires translator looking at a properly-sized BIN file). An $04 match means you are reasonably positive the file is yours (e.g. an SHR translator looking at an Apple Preferred file/auxtype).

Two-Stage Filters

The filter code must be very quick because it is called frequently by Babelfish. That's easy if your file has a specific file/auxtype, but what about "fuzzy" files that have no specific file/auxtype? In order to optimize the filter, yet still provide a high degree of confidence in your answer, Babelfish supports an ***optional*** "two-stage" filter.

When TrFilter is called you should quickly examine information *immediately* available to you (file, auxtype, size, etc.). If you can determine that you definitely cannot import this file then return $00. If you determine that you definitely can import this file then return $04.

If the file is a "maybe" then you would normally return an $01, $02 or $03 ranking and be done with it (as described above). However, if closer examination of the file would tell you for sure whether the file is supported, then you can implement a two-stage filter. When returning an $01, $02 or $03 ranking, also set bit 15 of the ranking word. *Important: Never return an $00 or $04 ranking with bit 15 set!*

By setting bit 15 you are telling Babelfish that you can take extra steps to determine if the file really is yours. If Babelfish runs a particular file through all the available filters and finds one that can definitely handle the file then your filter will not be called again. But if no other filters can positively handle the file, Babelfish will call your TrFilter routine a second time.

The second time TrFilter is called, bit 31 of the Directory Entry Pointer will be set. This is your cue to perform a closer examination of the file. That examination should then result in an $00 or $04 ranking (and bit 15 **must** be clear). *NOTE: It won't harm Babelfish if you return an $01, $02 or $03 ranking at this second stage, but if the second stage cannot determine between $00 and $04 then you should not use a two-stage filter!*

Important points:

If your filter is *not* a two-stage filter you must never return with bit 15 of the ranking word set! Your filter can quickly examine the information in memory (file/auxtype/filesize, etc.) and return a ranking.

If your filter *is* a two-stage filter you should first check bit 31 of the Directory Entry Pointer. If it is *not* set then you're being called the first time; quickly examine the information in memory and return a ranking (with bit 15 set if you think you can be more precise for this file). If bit 31 of the Directory Entry Pointer *is* set then you are being called the second time; skip the initial checks (you know this file already passed them) and immediately perform the detailed checks.

The "raw sound" data format is "any binary file that contains no byte values larger than $7F". While it is *possible* to write a two-stage filter with the second stage scanning through the entire file to verify there were no values larger than $7F, that should not be done because it would be *painfully* slow for the user!

The "Applied Engineering Sound" sound data format is also "any binary file that contains no byte values larger than $7F"…but it also has a header in the file that contains 4 bytes that are required to be certain values. In this case a two-stage filter makes sense; the second stage would load those 4 bytes into memory and compare them against the values they should be. No match returns $0000; a match returns $0004 (a binary file with those 4 exact bytes is highly likely to be an AE Sound).

Parameters

Stack before call

```
|      Previous contents       |
|------------------------------|
|            Space             |  Word–Space for result
|------------------------------|
|–       GetDirEntryRec      – |  Long–Pointer to GetDirEntry record
|------------------------------|
|                              |  <—SP
```

Stack after call

```
|      Previous contents       |
|------------------------------|
|            Rating            |  Word–Rating between $00-$04 of import ability
|------------------------------|
|                              |  <—SP
```

Errors

None

Logic

Helpful Illustrations/Examples

One-Stage Filters

An “AppleWorks Classic” filter can immediately respond $00 or $04 based on the file/auxtype.

An “ASCII Text” filter can immediately respond $03 for TEXT files and $02 for other files.

Two-Stage Filters

On the first pass, a “Sound Resource” filter can:

- immediately respond $0000 if the file does not have a resource fork
- immediately respond $0004 if the file/auxtype indicates it is a “sound resource” file
- immediately respond $8002 if the file has a resource fork but is not a “sound resource” file

On the second pass, a “Sound Resource” filter can:

- immediately open the resource fork of the file (it knows it has one because it passed the first stage filter) and ask the Resource Manager if there are any “rSound” resources in this file. Based on that answer the filter can return $0000 or $0004.

TrExportOptions

Resource Type: rCodeResource ($8017) Resource ID: $00000003 Attributes: locked, convert

Description

This code resource presents a dialog box in which the user sets exporting options. If there are no export options this resource does not exist (and bit 1 of ExportFlags is clear).

For consistency with Babelfish windows and all translators from Seven Hills, the translator should present its options in a moveable modal window. The title “Exporting Options” should be used as the window’s title, and the translator name and version number (as specified in the TrVersion resource) should be displayed. See “Overview-Using Babelfish” for examples.

At the bottom of the dialog should be a “Cancel” and a default “Accept” button. If the user hits “Cancel” then no changes are made; if the user hits “Accept” then the options are saved by Babelfish. *NOTE: The Accept button should not be named “Export” or “Save” because the user is not “exporting” or “saving” the translator or its options…the user is simply “accepting” the displayed options.*

Parameters

Stack before call

```
|       Previous contents       |
|———————————————————————————————|
|–          xferRecPtr         –|  Long–Pointer to Transfer Record
|———————————————————————————————|
|                               |  <—SP
```

Stack after call

```
|       Previous contents       |
|———————————————————————————————|
|                               |  <—SP
```

Errors

$0000 bfNoErr	No error

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000 bfContinue	No error
$8005 bfMemErr	Abort; memory error
$80FF bfAbortErr	Other error

Logic

```
Prior to calling displaying the dialog, the translator should call BFLoadData to request any
   previously-saved export options from a preference file.  If there is any error in loading the
   file, the translator should set up the dialog using default choices.
The preference file contains data previously stored by the translator, and that data should be used
   to set up the options dialog.  The format of this data is entirely up to the translator.  It is
   recommended that the translator include some signature or revision indicator that it can use to
   validate the data (if the validation fails, use the default settings).
If the user changes any settings and clicks Accept, the translator should make the necessary changes
   in the data and call BFSaveData to save a new preference file to disk.
```

Helpful Illustrations/Examples

TrExportInfo

Resource Type: rText ($8016) Resource ID: $00000003 Attributes: none
Optional Resource: rStyleBlock ($8012) Resource ID: $00000003 Attributes: none

This text is displayed when the user selects the Info button in any translator-selection dialog. It gives general information about the translator, and specifically describes the exporting options (if any). *NOTE: The translator's name and version are displayed by Babelfish; to avoid potential conflicts the version number should not be included in the text itself.*

The text is displayed in a TextEdit control and is scrollable. If the corresponding rStyleBlock resource exists the text is styled; otherwise the text appears in Shaston 8.

One example of a translator that does **not** have any exporting options…

```
This translator supports exporting, but no exporting options are required.

Technical support and information about new versions of this translator can be obtained by
contacting:

Seven Hills Solutions Specialists, 1254 Ocala Road, Tallahassee, FL  32304.
Technical help: support@sevenhills.com
Everything else: sales@sevenhills.com

Copyright 1991-1998 by Seven Hills Solutions Specialists
```

One example of a translator that **does** have exporting options…

```
You can specify the following options before exporting:

CONVERT PARAGRAPHS TO LINES OF __ CHARACTERS
Blah blah blah blah blah

REPLACE CR/ WITH CR/LR
Blah blah blah blah blah

...and so on...

• • • • • • • • • • 

Technical support and information about new versions of this translator can be obtained by
contacting:

Seven Hills Solutions Specialists, 1254 Ocala Road, Tallahassee, FL  32304.
Technical help: support@sevenhills.com
Everything else: sales@sevenhills.com

Copyright 1991-1998 by Seven Hills Solutions Specialists
```

TrInit

Resource Type: rCodeResource ($8017) Resource ID: $00000001 Attributes: locked, convert

Description

When Babelfish prepares to use a translator, it calls the translator's TrInit function. The purpose of TrInit is to afford the translator an opportunity to install its message handler.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

none

Notes

The format of the translator's request name is "Babelfish~Your Company Name~Translator Name~`MemoryID`" (where `MemoryID` is the memory ID returned by _MMStartup; this lets your translator be duplicated numerous times and have each copy be recognized as a unique translator).

Translators may structure the handler any way they see fit; however, the recommended method would be to include the required and optional functions in this code resource. Babelfish leaves this code locked and fixed throughout the entire translation process.

Logic

TrStartUp ($9101)

Description

This code initializes buffers, variables, etc. in preparation to read or write the given file. If TrStartup opens a resource file, the file will remain open until after TrShutdown is called. If the translator does not close the resource file, _ResourceShutdown will close it when the translator is “closed”.

NOTE: If you return an error from TrStartUp you will ***not*** *be called again! This means if you going to return an error you must dispose of any memory you allocated, etc., before returning from TrStartUp. If TrStartUp is successful, TrShutDown will always get called (even if an error occurs during TrRead or TrWrite).*

Parameters

dataIn — Pointer to this structure:

$00	– xferRecPtr –	Long-Pointer to Transfer Record

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	TRresult	Word-Error code

Errors

$0000	bfNoErr	No error

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8002	bfBadFileErr	Abort; structure not as expected
$8003	bfReadErr	Abort; media error reading from the disk
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Logic

```
By examining the DirectionBit of the BFMiscFlags you can determine whether you’re supposed to
   prepare for importing or exporting.
If you need/want a direct page you must acquire and maintain it yourself (and release it in
   TrShutDown).
```

IF Importing

```
Open the given file for importing (read-only recommended)
Determine the one specific Kind of data you will be returning and store that information in the
   Kinds field.  This step is easy if you only support one kind of data, but what if more than one
   Kind is specified?  In that case, use the best format for the particular file that’s being
   imported.  For example, if the Kinds record says “640 screen graphic” “320 screen graphic” and
   “QuickDraw Picture” then the application said it could import any of those kinds of graphics (AND
   your ImportKinds said that you could import at least one of those Kinds).  At this point you
   should examine the file to determine which is best…if it’s a 320 screen graphic then return that
   as the Kind; if it’s a QuickDraw Picture file, then return that as the Kind.
Determine the value that will represent a full thermometer and store it in the “FullTherm” field of
   the XferRec (see the “Standard Data Format Definitions” section for further guidelines).
Establish and initialize the data record for this Kind of data.
```

```
You could also establish your data buffer (or release and allocate it during each TrRead call).
   Whether you establish your data buffer now or during TrRead, you must remember to release it in
   the TrShutDown routine!
```

IF Exporting

```
Create and Open the given file for exporting (write-only recommended).  By examining the Kinds
   record you will know the specific Kind of data you will be receiving.
```

Helpful Illustrations/Examples

TrShutDown ($9102)

Description

This code reverses everything done in TrStartUp (e.g. closes the file, releases buffers, etc.). All memory acquired by the translator must be disposed of during (or before) TrShutDown. *NOTE: If the TrStartUp call was successful, Babelfish* ***will*** *balance with a call to TrShutDown, even if an error occurs during TrRead or TrWrite.*

NOTE: Babelfish will remove the translator's message handler and close its resource fork immediately after this call completes, ***regardless*** *of the error or status returned.*

Parameters

dataIn — Pointer to this structure:

```
      |————————————————————————|
$00   |–       xferRecPtr     –|  Long-Pointer to Transfer Record
      |————————————————————————|
```

dataOut — Pointer to this structure:

```
      |————————————————————————|
$00   |       recvCount        |  Word-Number of times the request was received
      |————————————————————————|
$02   |       TRresult         |  Word-Error code
      |————————————————————————|
```

Errors

None

Status

$8000	bfDone	Successful import/export completion
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Logic

```
If you acquired a direct page you must release it.
```

IF Importing

```
Close the file
Release your data buffer
Release the data record
```

IF Exporting

```
Close the file
If Status is NOT "bfDone" then delete the destination file (because the transfer was aborted due to
   an error, or because the user requested the export be cancelled).
```

Helpful Illustrations/Examples

TrRead ($9103)

Description

This code reads some or all of the given file, converts the contents into one of the standard data formats, then returns that information to Babelfish.

Instead of importing an entire file at once, data is often divided into smaller, more manageable sizes (see “Standard Data Format Definitions” for further information). To import an entire file an application repeatedly calls the BFRead routine (which in turn calls TrRead) until “bfDone” is returned.

When TrRead is called the translator reads some or all of the file into memory. It then fills in the XferRec fields appropriately (including the DataRecordPtr) and returns. *NOTE: If the Status is “bfDone” then the DataRecordPtr is not valid when the application gets control back! Therefore, if this is the last chunk of data you should leave the Status as “bfContinue” (with a full thermometer) and on the next TrRead call you can immediately return a “bfDone” Status to shut down everything.*

Translators may use additional support routines after calling BFLoadSupport and SrStartup. When they are finished with the support routine, they must call SrShutdown and BFUnloadSupport.

Translators should call BFProgress to update the progress window often. For import translators, user messages can be used to provide more information to the user about the importing progress. For example, a “Super Hi-Res Graphics” translator that can import Unpacked Screens, Packed Screens, Apple Preferred Files, etc., might want to state the exact kind of file being imported. And it could even update the progress during the importation:

```
Apple Preferred File Format
   Reading header…
Apple Preferred File Format
   Calculating palette information…
Apple Preferred File Format
   Importing…
```

Parameters

dataIn — Pointer to this structure:

```
       |————————————————————————|
  $00  |–       xferRecPtr     –|  Long-Pointer to Transfer Record
       |————————————————————————|
```

dataOut — Pointer to this structure:

```
       |————————————————————————|
  $00  |        recvCount       |  Word-Number of times the request was received
       |————————————————————————|
  $02  |        TRresult        |  Word-Error code
       |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5305	bfNotSupported	Translator function not supported

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8002	bfBadFileErr	Abort; structure not as expected
$8003	bfReadErr	Abort; media error reading from the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Logic

Helpful Illustrations/Examples

TrWrite ($9104)

Description

This code accepts data in one of our standard data formats, converts it into its specific format, then saves it to disk.

Instead of exporting an entire file at once, data is often divided into smaller, more manageable sizes (see “Standard Data Format Definitions” for further information). To export an entire file an application repeatedly calls the BFWrite routine (which in turn calls TrWrite).

Translators may use additional support routines after calling BFLoadSupport and SrStartup. When they are finished with the support routine, they must call SrShutdown and BFUnloadSupport.

When the application is finished sending data, the translator’s TrShutDown routine will be called.

Parameters

dataIn — Pointer to this structure:

```
     |————————————————————————|
$00  |–       xferRecPtr     –|  Long-Pointer to Transfer Record
     |————————————————————————|
```

dataOut — Pointer to this structure:

```
     |————————————————————————|
$00  |        recvCount       |  Word-Number of times the request was received
     |————————————————————————|
$02  |        TRresult        |  Word-Error code
     |————————————————————————|
```

Errors

$0000	bfNoErr	No error
$5305	bfNotSupported	Translator function not supported

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Status

$0000	bfContinue	No error
$8004	bfWriteErr	Abort; media error writing to the disk
$8005	bfMemErr	Abort; memory error
$80FF	bfAbortErr	Other error

Logic

Helpful Illustrations/Examples

Babelfish Support Routines

Support Routines can be anything, but they are intended to be routines that many translators and/or applications will find useful. For example, instead of twenty translators having pages of code to load a Macintosh resource, that the code can exist *once* as a support routine. And, like system tools, if a support call is ever rewritten to be better, stronger, or faster, *all* the translators and applications that use the support routine instantly get better with no recoding!

A translator or application follows these steps in order to use a particular support routine:

1) Call BFLoadSupport

2) Call SrStartUp (only if the support routine was loaded)

3) Call SrShutDown (don’t forget! but only if SrStartUp did not return an error!)

4) Call BFUnloadSupport (don’t forget! but only if BFLoadSupport did not return an error!)

Support Routine File Format

Support routines are located in the *:System:SHS.Babelfish folder. The filetype is $BC (Generic load file) and the auxtype is $4014 (Babelfish support routine). *NOTE: Users can “inactivate” a support routine by checking a box in the Finder, which sets the high bit of the auxiliary type. Babelfish respects the users wishes and ignores inactive files.*

When the support routine receives control, the direct page and data bank are not set. The support routine must set these registers as needed, and must restore them upon return. Direct page space is not provided for the support routine’s use; the support routine may use the stack or request and maintain its own space from the Memory Manager.

Communication with the support routine’s functions is by IPC. An application calls SendRequest with the proper request code, directing the request to the support routine’s ID number, with a `sendHow` value of `sendToUserID + stopAfterOne`. The support routine’s message handler should route the call according to the request code to one of its function handlers.

Support File Structure

Support files are structured similar to translators. They consist of several required and optional resources. The data fork may be used at the author’s discretion.

Specific message numbers are at the discretion of the author, but shall be in the range of $B000-BFFF.

NOTE: The number in parentheses is the required resource ID number.

Support files must contain all of these resources:

Name	Type(ID)	Description
SrVersion	rVersion(1)	This resource is used by the Finder during Get Info. *See GS Tech Note #76.*

SrInit	rCodeResource(1)	Code that installs the message handler and function router.
SrRequestName	rPString(1)	The request name used to send messages.

All support files must contain the following functions:

Name	Type(ID)	Description
SrStartUp		Code that initializes the support routines.
SrShutDown		Code that terminates and releases memory.
SrGetVersion		Code that returns the version number.

The following resources are optional (but recommended):

Name	Type(ID)	Description
SrAbout	rComment(1)	This resource is used by the Finder during Get Info (on the Comment page). *See GS Tech Note #76.*
SrCantLaunch	rComment(2)	This resource is used by the System 6 Finder when it cannot launch a file. *See "Programmer's Reference for System 6.0", pp 364 & 429.*

SrVersion

Resource Type: rVersion ($8029) Resource ID: $00000001 Attributes: none

This resource contains a long word version number.

SrAbout

Resource Type: rComment ($802A) Resource ID: $00000001 Attributes: none

This "Finder Info" resource contains general information about the support file.

SrCantLaunch

Resource Type: rComment ($802A) Resource ID: $00000002 Attributes: none

This resource contains a standard message that is displayed if the user tries to open a support file from the Finder. The suggested text is:

```
This is a support file and can't be launched.  It is a Seven Hills Solutions Specialists
"Babelfish" extension.  To use it, put it in the "*:System:SHS.Babelfish" folder and open an
application that uses Babelfish.
```

SrRequestName

Resource Type: rPString ($8006) Resource ID: $00000001 Attributes: none

This is the request name of the support file as it may be used with `SendRequest`.

SrInit

Resource Type: rCodeResource ($8017) Resource ID: $00000001 Attributes: locked, convert

Description

The sole function of SrInit should be to install the support routine's IPC message handler. SrInit is only be called once (if an additional translator calls BFLoadSupport and the support routine is already loaded, SrInit is *not* called a second time).

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

Tool Locator errors returned unchanged.

Notes

The format of the support file's request name is "Babelfish Support~Your Company Name~Support File Name~".

Support files may structure the handler any way they see fit; however, the recommended method would be to include the required functions in this code resource. Babelfish leaves this code locked and fixed throughout the entire translation process (until BFUnloadSupport). If you have large data tables or lengthy code, you are encouraged to load it separately as it is needed.

Babelfish will call SrInit with a JSL in full native mode. SrInit must save and restore the data bank and direct page registers. SrInit must return via RTL with the accumulator and the carry affected by the `_AcceptRequests` call. If SrInit returns an error, Babelfish will unload and dispose of the SrInit memory.

Logic

SrStartUp ($Bxxx)

Description

This code initializes the support routines for use. Its actions, parameters, and errors are up to the author. If the optional functions are not already in memory, they should be loaded now.

Additional code or data may be stored in the support file's resource or data fork. When loading resources, care must be taken to leave the environment unchanged. In other words, you may not leave your resource fork open. The steps for loading a resource are:

1. GetCurResourceApp:origapp
2. ResourceStartup(MMStartup)
3. OpenResourceFile(readEnable,0,LGetPathName2(MMStartup,1)):fileID
4. LoadResource
5. If you need to keep it around, DetachResource
6. ResourceShutdown
7. SetCurResourceApp(origapp)

SrShutdown ($Bxxx)

Description

This code shuts down the support routines. Its actions, parameters, and errors are up to the author, but as a minimum it should release any acquired memory and close opened files. This is the last call made to the support file's code; the next thing that happens will be Babelfish disconnecting the message handler and removing the code from memory (unless there are other users).

Note that the support routine must **never** remove its own request procedure, even when SrShutdown is called! The reason is that there may be other translator's still using the support routine. For example, someone may call BFLoadSupport but *not* immediately call SrStartUp, planning to do it a little later. In the meantime someone else can call BFLoadSupport, SrStartUp, SrShutdown, and BFUnloadSupport. As far as the support routine knows, everyone is "done" with it, but in reality the first translator expects to be able to call SrStartUp. So a support routine must never remove its own request procedure; Babelfish will do it at the appropriate time.

SrGetVersion ($Bxxx)

Description

This code's actions, parameters, and errors are up to the author; but as a minimum it should return the support file's version number to the caller.

Suggested Support Routines

See the separate “MiniMacRezMgr” file to examine an actual support routine that is already written!

Some support routines have been proposed and are documented here for discussion.

Remap PixelMap

Remap PixelMap contains routines to help deal with remapping PixelMaps.

Remap PixelMap to 640 Mode

This action code modifies a PixelMap so that, on the 640 mode screen with a standard color palette, the PixelMap will look as much like the original as possible. *NOTE: This routine is guaranteed to return quickly if the pixelmap is already in 640/standard format, so there’s no need to check for that condition before making this call.*

dataIn — Pointer to this structure:

$00	LocInfoPtr	Long–Pointer to PixelMap’s LocInfo record
$04	SCBsArrayPtr	Long–Pointer to array of SCBs for the PixelMap
$08	PalettesPtr	Long–Pointer to an array of palettes for the PixelMap

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	SRresult	Word-Error code

The LocInfo’s BoundsRect is assumed to start at 0,0. If the SCBsArrayPtr is NULL then all SCBs are assumed to be equal to the mode byte in the LocInfo. Only palettes referenced by an SCB will be accessed.

On return, the PixelMap referenced by the LocInfo has been modified as needed, and the LocInfo itself has been modified if a change of mode has occurred. The LocInfo is now suitable for drawing (via PPToPort, etc.) in 640 mode (SCBs = $80) with the standard color palette. Note that the SCB and palette data passed to this call is *not* modified.

Errors

$0000	srNoErr	No error
$80FF	srError	Error; call failed

Memory Mgr, GS/OS, Resource Mgr errors returned unchanged

Get Remap Tables

This action code returns a pointer to the translate tables used by action code $03 to convert arbitrary colors to 640 mode, standard color palette. This can be used by applications or translators that want to do their own remapping.

dataIn — NIL

dataOut — Pointer to this structure:

```
|————————————————————————|
```

$00	recvCount	Word-Number of times the request was received
$02	SRresult	Word-Error code
	– TablePtr –	Long–Pointer to an even-scanline translate table

Errors
$0000 srNoErr No error

To get the pointer to the odd-scanline table, add 4096 ($1000) to the returned pointer. Each table is indexed by standard IIGS color palette entries in the range $0000-$0FFF, interpreted as $0RGB.

To use the table, look up a pixel's color entry from the appropriate palette, use that to look up a byte in the appropriate translate table (the even/odd designations aren't really important, as long as you alternate between the two for each line), then mask out as many bits as needed from that byte (2 bits for a 640-sized pixel, 4 for 320). *Do not ever shift bits from these tables before using them:* each bit must remain in its own position for the dither patterns to form correctly!

Convert Quotes

Convert Quotes contains a routine to convert quotes in a block of text to or from "curly" quotes.

Convert Quotes

This action code modifies a block of text so it contains curly quotes, or so it contains only straight quotes.

dataIn — Pointer to this structure:

– TextStreamLength –	Long–Number of characters in text stream
– TextStreamPtr –	Long–Pointer to to text stream
Which	Word–1 means straight to curly; 2 means curly to straight

dataOut — Pointer to this structure:

$00	recvCount	Word-Number of times the request was received
$02	SRresult	Word-Error code

Errors
$0000 srNoErr No error

Standard Data Format Definitions

Several data formats have already been defined. The formats are not currently listed in numerical order. Instead, we start with a simple format that is described in-depth, then move on to other formats, focusing on different points (so even if you're not interested in an earlier format you should read the information anyway, to enhance your understanding). Further formats may be proposed by anyone at any time.

For simplicity, any "Implementation Notes" assume no error occurs. Also, the notes are from the standpoint of an application using a translator to import data. However, the data formats themselves are defined using the generic terms "Sender" and "Receiver," which apply equally to a translator or an application:

- The "Sender" is the one who converts a private data format into the generic data format and sends it along. When an application is importing, the translator is the Sender; when an application is exporting, the application is the Sender.
- The "Receiver" is the one who receives the generic data format and converts it into a private data format. When an application is importing, the application is the Receiver; when an application is exporting, the translator is the Receiver.

Therefore, if you are writing a translator that imports (sends) data to an application, you must be aware of the Sender's responsibilities. If you also support exporting (receiving) data from an application, you must be aware of the Receiver's responsibilities.

Likewise, if you are writing an application that imports (receives) data from a translator, you must be aware of the Receiver's responsibilities. If you also support exporting (sending) data to a translator, you must be aware of the Sender's responsibilities.

For future compatibility, each data record begins with a parameter count. Either the application or the translator (whoever doesn't create the data record, but relies on the data in that record) must check to make sure it doesn't attempt to store/retrieve values past the end of the data record that was created.

General Rules

- Whenever a pointer to data, or actual data, is passed, the Receiver is expected to copy the information (if desired) into his own private area.
- Whenever a handle is passed, that handle becomes the sole responsibility of the Receiver. **The Receiver must use _SetHandleID so he becomes the owner of the handle!**
- If your application does not make use of a particular piece of information that is being supplied, just ignore it (if data or a pointer to data) or dispose of it (if handle).
- No data is exchanged in the data record during a TrStartUp or TrShutDown call. The only time a Sender or Receiver can expect the data record to be valid is during/after the BFRead, TrRead, BFWrite, and TrWrite calls.
- The main goal of Babelfish is to make life easier for the application, not the translator author. Therefore, the following rules apply whenever a data format has duplicate fields, such as "XyzPointer" *and* "XyzHandle":

- <u>translators that send data to an application (an import translator)</u> **must** set the pointers to NULL and make the handles valid (thus applications *can* rely on always being given handles).
- <u>translators that receive data from an application (an export translator)</u> **must** check the handle field ***first***; if the handle is *not* null then the export code must use only the handle information (own it, use it, dispose of it). If the handle field was null then the export code obviously uses the pointer field.

Format 0: No data returned

This format doesn't have any data structure. It is designed for those occasions when the application does not want to go completely through the importation process. Some files contain data that might be suited to "doing" something with it rather than transforming it. For example, sounds could be played and pictures could be shown. A sound (or graphic) translator could report that it supports this "play-and-don't-import" mode as well as normal importing. If the app does not wish to support code to accept the data of this kind, it may simply ask for this "null" kind instead.

"Importing" this kind would still require a TrStartup and TrShutdown, but the TrRead would "do it" rather than fill in a data record and return trDone.

Notes

There doesn't seem to be any useful "exporting" for this kind.

This suggests having another button on the TMGetFile dialog named "Sample…" which would be visible only if the application asked for this kind.

Format 5: Font

Our font data record looks like this:

```
      |————————————————————|
$00   |–     ParmCount    –|      Word–Number of Parameters (2)
      |————————————————————|
$02   |–                  –|
      |–     FamilyName   –|  1   Long–Pointer to pString of font family name
      |–                  –|
      |————————————————————|
$06   |–                  –|
      |–  FontDataHandle  –|  2   Long–Handle to a standard IIGS font
      |–                  –|
      |————————————————————|
```

FamilyName

The name can be up to 25 characters long. *See TB1, page 8-3.*

FontData

The standard IIGS font format as described in *Apple IIGS Toolbox Reference: Volume 2, page 16-25 and Volume 3, page 43-5.*

Implementation Notes

The application has previously started Babelfish. Now it wants to import a file, so it creates an XferRec of the appropriate size, fills in the appropriate fields, then calls BFGetFile. After the user selects a file and translator, the application calls BFImportThis.

Babelfish calls TrStartUp, which sees we're importing a file, so it opens the specified file (FilePathPtr+FileNamePtr) for read only.

Because we're returning a font, TrStartUp sets the DataKinds field to $40 (Font Kind).

Because a font can be translated with two "TrRead" calls (the first time with the data being passed, the second time with a "done" status), TrStartUp sets the "FullTherm" field to 2, and initializes the CurrentTherm to 0.

TrStartUp creates space for the data record ($0A bytes) and stores a 2 in the ParmCount field. It stores a pointer to this data record in the DataRecordPtr field of the XferRec.

TrStartUp returns control to Babelfish.

The application prepares itself to receive a font, then calls BFRead, which in turn calls TrRead.

TrRead reads the file and formulates the two pieces of data we need (the font name and the IIGS font data). It sets the fields in the data record appropriately, and updates the DataRecordPtr field if necessary.

Because there aren't any useful messages that this particular font translator can display, the StandardMsg bit in ProgressAction is set to display a default "Working…" message.

The CurrentTherm value is incremented (becomes 1) and control returns to Babelfish.

Babelfish updates the progress window (if it's open for this import the thermometer will now be half full) and returns to the application.

The application stores the font name and font data, then calls BFRead again, which in turns calls TrRead again.

TrRead knows there isn’t any more data to pass, so it sets the Status field to “bfDone” and increments CurrentTherm (becomes 2). The current message (“Working…”) is still valid, so it clears the NewMessage bit in ProgressAction to indicate “no change.” Then, control returns to Babelfish.

Babelfish updates the progress window (if it’s open for this import the thermometer will now be completely full). Because the Status field says “bfDone,” Babelfish calls TrShutDown.

TrShutDown closes the file it was working with, and it releases the space it acquired for the data record. Then it returns to Babelfish.

Babelfish closes the progress window (if it’s open for this import), then returns the “bfDone” Status to the application.

The application sees the “bfDone” status, releases the memory it acquired for the XferRec, then uses the font data that it received earlier.

TIP: If an application is requesting only “font,” it might want to specify “no progress window” because the conversion happens in only two brief steps. However, if the application sometimes requests fonts and sometimes requests other kinds of data, the progress window should probably be shown all the time for consistency.

Format 2: Graphic—PixelMap

A single call returns/writes an entire QuickDraw II pixelmap. *NOTE: A translator or application might wish to use Babelfish support routines to remap the image for best display quality on a standard 640 mode screen.*

Our Graphic—PixelMap data record looks like this:

$00	ParmCount		Word–Number of Parameters (8)
$02	LocInfoRecPtr	1	Long–Pointer to LocInfo record
$06	SCBsArrayPtr	2	Long–Pointer to SCB array
$0A	PalettesPtr	3	Long–Pointer to an array of color palettes
$0E	ImageHnd	4	Long–Handle to image data
$12	SCBsArrayHnd	5	Long–Handle SCB array
$16	PalettesHnd	6	Long–Handle to an array of color palettes
$1A	ResolutionH	7	Long–Horizontal (left/right) resolution of the picture
$1C	ResolutionV	8	Long–Vertical (up/down) resolution of the picture

LocInfoRecPtr

Pointer to a LocInfo record.

BoundsRect must always start at 0,0.

SCBsArrayPtr

Pointer to an array of SCBs corresponding to this pixelmap, 1 byte each. If NULL, all SCBs are equal to the mode byte in the LocInfo.

PalettesPtr

Pointer to an array of color palettes referenced by this pixelmap. All 16 palettes may be present, or only those up to the highest one used.

ImageHnd

Handle to image data. If NULL use the pointer in the LocInfo record (PaletteData field) and do not dispose of the data. If not NULL, dispose of the handle when you're through.

SCBsArryHnd

Handle to an array of SCBs (see SCBsArrayPtr). If the handle is NULL use the SCBsArrayPtr and do not dispose of the data. If not NULL, dispose of the handle when you're through.

PalettesHnd

Handle to an array of color palettes (see PalettesPtr). If the handle is NULL use the PalettesPtr and do not dispose of the data. If not NULL, dispose of the handle when you're through.

ResolutionH

Horizontal (left/right) resolution of the picture, in 640-mode-sized pixels (80dpi). $0000 if unknown. *See notes under Graphic—QuickDraw II Picture format.*

ResolutionV

Vertical (up/down) resolution of the picture, in 640-mode-sized pixels (36dpi). $0000 if unknown. *See notes under Graphic—QuickDraw II Picture format.*

Notes

When importing, a translator must always put valid handles in the ImageHnd, SCBarrayHnd, and PalettesHnd fields.

When exporting, an application can put valid handles in 0, 1, 2, or 3 of the ImageHnd, SCBarrayHnd, and PalettesHnd fields (putting NULL into the unused fields). If a pointer is passed the translator will copy the data (if necessary) and save it to disk. If a handle is passed the translator will use the data and dispose of the handle. *NOTE: If a handle is passed the corresponding pointer fields must still be correct!*

Although passing a pointer to some data and possibly passing a handle to the same data may seem repetetive at first, it makes sense when you consider what happens during an import or export.

If only a pointer was passed during import, a translator would load an image into memory and pass a pointer to the image. The application would then have to make a *copy* of the data (which requires twice as much memory and takes time). As soon as the translator gets control again it would release the image from memory. Instead, the translator just "hands off" the picture to the application by passing a handle to the data (making it the application's responsibility).

When exporting, an application might have pictures that are in use. To prevent having to duplicate a picture just to export it, the application can pass a pointer to the picture. If an application *did* want to dispose of the picture it would pass a handle. Note that only the application has the choice; translators importing to an application must always use handles.

Format 4: Graphic—QuickDraw II Picture

Our Graphic—QuickDraw II Picture data record looks like this:

$00	ParmCount		Word–Number of Parameters (3)
$02	PicHandle	1	Long–Handle to QuickDraw II picture data structure
$06	ResolutionH	2	Long–Horizontal (left/right) resolution of the picture
$08	ResolutionV	3	Long–Vertical (up/down) resolution of the picture

PicHandle

A standard QuickDraw II Picture (as created by OpenPicture/…commands…/ClosePicture calls, and as documented in Apple IIGS Tech Note #46).

ResolutionH

Horizontal (left/right) resolution of the picture, in 640-mode-sized pixels (80dpi). $0000 if unknown.

ResolutionV

Vertical (up/down) resolution of the picture, in 640-mode-sized pixels (36dpi). $0000 if unknown.

Notes

As with any PicHandle, to draw the picture an application pushes the PicHandle, pushes a pointer to a RECT that defines the destination rectangle, then calls DrawPicture. To avoid scaling the picture, the destination rect should be set to the picture's boundary rectangle (see PicFrame in Apple IIGS Tech Note #46).

No scaling should be done if the picture resolution is unknown or if it is standard screen resolution. Other resolutions indicate the drawing rectangle should be scaled so that the picture will be displayed at the proper size. For example:

```
PictureWidth is the width of the picture's rectangle (as specified in the PicHandle)
PictureHeight is the height of the picture's rectangle (as specified in the PicHandle)
DesiredHres := the desired horizontal (left/right) resolution of the picture (80 is standard IIGS 640
    mode screen)
DesiredVres := the desired vertical (up/down) resolution of the picture (36 is standard IIGS 640 mode
    screen)
ActualHres := the actual horizontal (left/right) resolution of the picture
ActualVres := the actual vertical (up/down) resolution of the picture
If ((ActualHres=0) OR (ActualHres=DesiredHres)) THEN DrawWidth:=PictureWidth ELSE
    DrawWidth := (DesiredHres/ActualHres)*PictureWidth
If ((ActualVres=0) OR (ActualVres=DesiredVres)) THEN DrawHeight:=PictureHeight ELSE
    DrawHeight := (DesiredVres/ActualVres)*PictureHeight
Now draw the picture using a rectangle that's DrawWidth wide by DrawHeight high and the picture will
    be displayed at the proper size for the desired resolution.
```

For example, a MacPaint picture (approx. 80x72) being displayed on the IIGS 640 mode screen (80x36) should be scaled. If the MacPaint rectangle is 200 pixels tall, the rectangle used for drawing on the screen will be (36/72)*200, or 100 pixels tall.

On the other hand, a MacPaint picture (approx. 80x72) being displayed in GraphicWriter III (80x72) will not be scaled. But a standard IIGS screen (80x36) will be scaled to (80x72).

If a 300x300dpi TIFF graphic that is 1000 pixels wide by 1000 pixels tall (3.33 inches square) is being displayed on the 80x36 IIGS screen, the drawing rectangle will be (80/300)*1000 by (36/300)*1000, or 267 by 120 (3.213x3.333 inches).

Implementation Notes

The QuickDraw picture format is so easy to load, why wouldn't an application just read a QD II Pict directly off the disk? Well, that's the whole point of Babelfish: By using Babelfish an application could actually support more than a single data format.

Imagine this: An application directly loads files of type $C1/$0001 (Apple IIGS QuickDraw II Picture File). That's a single file format that is supported.

Now imagine this: An application uses Babelfish to load "Graphic—QuickDraw II Picture". The application is still getting the same data returned (a PicHandle to pass to the DrawPicture routine), but by going through Babelfish, a wealth of other files can be imported:

A translator could support importing $C0/$0003 (Packed QDIIPict). With no extra effort by the application, it now supports two file formats.

A translator that's capable of reading the different "bitmap" formats (packed screen, unpacked screen, Apple Preferred, PaintWorks Packed, etc.) could very easily return a handle to a QDIIPict. With no extra effort by the application, it now can import *any* bitmap image.

A translator that reads a font file off the disk could be modified to create a bitmap font sample of that font, and return it as a QDIIPict. Again, with no extra effort by the application, it now can import a font as a QDIIPict.

And so on…

Format 3: Graphic—True Color Image

A true color image is a format for pixelmap pictures whose color range or resolution is beyond the standard IIGS screen capabilities. *NOTE: A translator or application might wish to use Babelfish support routines to remap the image to a standard pixelmap, and from there they may wish to remap the image for best display quality on a standard 640 mode screen.*

Our Graphic—True Color Image data record looks like this:

$00	ParmCount		Word–Number of Parameters (7)
$02	Width	1	Word–The width of the image
$04	Height	2	Word–The height of the image
$06	Lines	3	Word–The number of image lines being passed on this call
$08	PixelDataPtr	4	Long–Pointer to the pixel data
$0C	PixelDataHnd	5	Long–Handle to the pixel data
$10	ResolutionH	6	Long–Horizontal (left/right) resolution of the picture
$12	ResolutionV	7	Long–Vertical (up/down) resolution of the picture

Width

The width of the image. This value must be filled in by the Sender before the first call and must not be changed later.

Height

The height of the image. This value must be filled in by the Sender before the first call and must not be changed later.

Lines

The number of image lines being passed this call. *NOTE: Lines of pixels are sent in top-down order.*

PixelDataPtr

Pointer to pixel data (Width * Lines * 3 bytes). Each pixel takes three consecutive bytes, in the order of Red, Green, and Blue. Each byte's color range is from 0 to 255.

The image data is assumed to be gamma corrected for display on a device with the NTSC standard gamma of 2.2, such as the IIGS' video system.

PixelDataHnd

Handle to the pixel data. If NULL use the PixelDataPtr and do not dispose of the data. If not NULL, dispose of the handle when you're through.

ResolutionH

Horizontal (left/right) resolution of the picture, in 640-mode-sized pixels (80dpi). $0000 if unknown. *See notes under Graphic—QuickDraw II Picture format.*

ResolutionV

Vertical (up/down) resolution of the picture, in 640-mode-sized pixels (36dpi). $0000 if unknown. *See notes under Graphic—QuickDraw II Picture format.*

Implementation Notes

To convert from a true color pixel to a IIGS color value, drop the low four bits of each byte and pack the rest together in a word. To convert the other way, duplicate the four bits of each color component: color levels $0, $1, $2, ..., $E, $F translate to $00, $11, $22, ..., $EE, $FF.

True color images tend to be huge, easily larger than available memory, so translators should read them in chunks, with generally less than 64K of pixel data per chunk. If the application is just accumulating all of the lines of pixels it will still run out of memory, however the application normally would be reducing the data to a more compact form or immediately writing it to disk to allow handling an image larger than available memory.

Format 6: Sound

Our Sound data record looks like this:

$00	ParmCount		Word–Number of Parameters (7, 8, 9, or 10)
$02	ActionCode	1	Word–Sender gives order to the Receiver
$04	ResponseCode	2	Word–Receiver's response
$06	SoundDataLength	3	Long–Number of bytes of sound data being passed
$0A	SoundDataPointer	4	Long–Pointer to the sound data
$0E	SoundDataHandle	5	Long–Handle to the sound data (or NIL)
$12	channelLength	6	Long–Total length of this channel
$16	frequency	7	Word–Frequency in Hz (0-65535Hz)
$18	volume	8	Word–Volume (1-15 or 0 for max)
$1A	relPitch	9	Word–relative pitch (from rSound) or 0 if "none/not provided"
$1C	SoundName	10	34 Bytes–pString of sound name (NIL if no name)

ActionCode

ActionCode is given by the Sender to tell the Receiver what to expect next. The Receiver acts on the ActionCode and returns a ResponseCode.

Valid ActionCodes:

$0000	Continuing	No special action needs to be performed by the Receiver (beyond accepting the incoming data and doing whatever he needs to with it).
$01xx	NewChannel	This tells the Receiver to "Prepare to receieve channel xx" (for example, $0101 tells the Receiver to prepare to receive channel 1).

ResponseCode

ResponseCode is given by the Receiver in response to an ActionCode. The Sender should check the ResponseCode only immediately after issuing an ActionCode to which a response is expected (at other times the ResponseCode is undefined).

Valid ResponseCodes:

$00	OK	The general response to mean "OK; proceed; send that…"
$80	SkipThat	This response can be given after one of the "Prepare to receive…" action codes is received. It tells the Sender not to bother sending that piece of information.

Implementation Notes

You MUST send channels in order, e.g., channel 1 then channel 2, etc.

Future action codes may be defined. The receiever should *always* respond “Skip That” to unknown codes.

You must have at least the first 7 parameters.

The volume parameter can be left at zero. This means “maximum”.

The “relPitch” field is the “relative pitch” parameter of an rSound. You can leave this value as zero, meaning “I don’t know what it is” (it can usually be calculated by the items that need to know it, but for some pitched sounds it is more accurate to have the parameter present rather than calculated)

If the string length of the name is zero, no name is used.

A Translator must always set the SoundDataHandle and pass the handle to the application. Otherwise, the SoundDataHandle may be zero, meaning none. (Just like Text Data). *NOTE: You can include a name without including a volume or relative pitch parameter by simply leaving these two as zeros. To omit all three, use a ParmCount of 7.*

Data for a channel may be sent in 1 pass or in multiple passes. The reciever keeps track of the current channel by watching for the “NewChannel” Action Codes. Only 1 channel at a time may be passed during a single Read/Write call.

Except for the Response Code, the receiver must never change any parameters in the Sound Data record.

The name, relPitch, frequency, and channel length paramaters must remain valid for the entire channel’s transfer, i.e., when the Action Code is bfContinuing they are valid. They are not required to be valid during the “NewChannel” message.

Channels may be different lengths and frequencies (although we don’t know why they ever would be).

Format 1: Text

Our Text data record looks like this:

$00	ParmCount		Word–Number of Parameters (5-28)
$02	ActionCode	1	Word–Sender gives order to the Receiver
$04	ResponseCode	2	Word–Receiver's response
$06	TextStreamLength	3	Long–Number of characters in text stream
$0A	TextStreamPtr	4	Long–Pointer to the text stream
$0E	TextStreamHnd	5	Long–Handle to the text stream
$12	FamilyID	6	Word–Family ID number for this text
$14	FontSize	7	Word–Point size for this text
$16	FontStyle	8	Word–Style for this text
$18	ForeColor	9	Word–Color for this text
$1A	BackColor	10	Word–Color for this text's background
$1C	Position	11	Word–Vertical position of this text
$1E	CharSpacing	12	Word–chExtra value for this text
$20	LineSpacing	13	Word–Line Spacing for current paragraph
$22	SpaceBefore	14	Word–Space Before this paragraph
$24	SpaceAfter	15	Word–Space After this paragraph
$26	FirstIndent	16	Word–First line indent of this paragraph
$28	LeftIndent	17	Word–Left indent of this paragraph
$2A	RightIndent	18	Word–Right indent of this paragraph
$2C	Justification	19	Word–Justification of this paragraph
$2E	TabArray	20	Array–Array of tabs (64 bytes)
$6E	Options	21	Flag Word–Paragraph-oriented options
$70	Border	22	Long (4 Bytes)–Border indicators
$74	Page Length	23	Word–Total length of the page in pixels
$76	Page Width	24	Word–Total width of the page in pixels

```
      |————————————————————|
$78   |–                  –|
      |–                  –|
      |–                  –|
      |–   Section Rect   –|   25  Rect–Dimensions of the current section
      |–                  –|
      |–                  –|
      |–                  –|
      |————————————————————|
$80   |–     Columns      –|   26  Word–Number of columns
      |————————————————————|
$82   |–      Gutter      –|   27  Word–Width of gutter between columns in pixels
      |————————————————————|
$84   |–                  –|
      |–    PicHandle     –|   28  Long–Handle to a QuickDraw II Picture
      |–                  –|
      |————————————————————|
```

<u>ActionCode</u>

ActionCode is initiated by the Sender to tell the Receiver what to expect next, and in some cases it tells the Receiver to perform some action. The Receiver acts on the ActionCode and returns a ResponseCode.

Valid ActionCodes:

$00	Continuing	No special action needs to be performed by the Receiver (beyond accepting the incoming data and doing whatever he needs to with it).
$01	GetSettings	This tells the Receiver to fill in the data record with its current settings. *This action is required; it should be performed before any data is sent.* For future compatibility, before issuing this action the Sender should completely initialize the data record to default values it can use (just in case the Receiver doesn't fill in all the fields of the data record). Then the Sender should issue this action so the Receiver can initialize the data record with its current information. By this "double initialization" both Sender and Receiver can be sure the data record is filled with information that each can use.

Each "section" of a document is divided into the following parts. Only the parts that actually exist must be passed, but if a part is going to be passed, it must be passed in the order listed here. *NOTE: If the Receiver receives a document part out of sequence, that indicates the previous section is complete, and a new section is beginning.*

The following action codes inform the Receiver to "prepare to receive…" a particular part of a document section:

$02	FirstHeader	"Prepare to receive the header for the first page of this section."
$03	OddHeader	"Prepare to receive the header for the odd pages of this section."
$04	EvenHeader	"Prepare to receive the header for the even pages of this section."
$05	FirstFooter	"Prepare to receive the footer for the first page of this section."
$06	OddFooter	"Prepare to receive the footer for the odd pages of this section."
$07	EvenFooter	"Prepare to receive the footer for the even pages of this section."
$08	BodyText	"Prepare to receive the body text for this section."
$09	Footnotes	"Prepare to receive the footnotes for this section."

ResponseCode

ResponseCode is given by the Receiver in response to an ActionCode. The Sender should check the ResponseCode only immediately after issuing an ActionCode to which a response is expected (at other times the ResponseCode is undefined).

Valid ResponseCodes:

$00	OK	The general response to mean "OK; proceed; send that…"
$80	SkipThat	This response can be given after one of the "Prepare to receive…" action codes is received. It tells the Sender not to bother sending that piece of information.

TextStreamLength

The length of the text being passed (the number of relevant characters found at TextPtr).

TextStreamPtr

A pointer to the text being passed. The other fields of the text data record contain the formatting information for *this* text stream. Any time there is a change in formatting, the current text stream ends and a new text stream begins. For example, the sentence "**The quick brown fox** jumps over the *lazy dog*" would be sent as five separate text streams:

Helvetica, 10, plain	The
bold	quick
plain	brown fox
Times, 12	jumps over the
italic	lazy dog

The text can contain any ASCII character in the range $20-$FF, and select characters in the range $00-$1F. Characters in the range $00-$1F have special meaning; the currently defined characters are listed below (all undefined characters are reserved):

$06	Footnote Ref	This character signals that the following character is to be interpreted as the footnote reference (typically displayed superscripted).
$09	Tab	Tabs to the next tab stop.
$0A	Line Feed	
$0B	Column Break	Denotes the end of a column; following text should start at the top of the next column.
$0C	Page Break	Denotes the end of the page.
$0D	Return	Denotes the end of a paragraph. *NOTE: There can be multiple Return characters in a single text stream; a different text stream needs to be generated only for changes in format, not for "text lines" or "paragraphs."*
$0E	Soft Hyphen	Use hyphen if at right margin, otherwise ignore.
$0F	New Line	Indicates a new line of the same paragraph.
$14	Page Number	Wherever this character appears, the current page number should appear in its place (if the text appears on page 1, "1" should appear in place of $14; if the text is moved to page 2, "2" should appear in place of $14).
$15	Date	Wherever this character appears, the current date should appear in its place.
$18	Time	Wherever this character appears, the current time should appear in its place.
$1B	Picture	A picture goes here. This code just indicates where a picture should be inserted; check the PicHandle field to determine if there is actually any picture being passed. *NOTE: Only one Picture code should appear in a single text stream.*

TextStreamHnd

A handle to the text being passed. If the handle is NULL use the TextStreamPtr and do not dispose of the data If not NULL, dispose of the handle when you're through. *NOTE: Translators must also pass a handle to an application; Applications may pass either a handle or a pointer.*

FamilyID

The font family number that the Font Manager expects to see for InstallFont.

FontSize

The point size of this text (from 1-255).

FontStyle

Contains the same bit settings as used for the QuickDraw II SetTextFace call, with some additions. The complete definition:

Bit0	Bold	Bit8	Underline (word)
Bit1	Italic	Bit9	Underline (double)
Bit2	Underline (continuous)	Bit10	Underline (dotted)
Bit3	Outline	Bit11	Strikethru
Bit4	Shadow	Bit12	Small Caps
Bit5	<reserved>	Bit13	All Caps
Bit6	Superscript	Bit14	Inverted (fore/back swap)
Bit7	Subscript	Bit15	Hidden

NOTE: Absence of styles indicates "Plain" style.

ForeColor

Indicates the color for text in this stream in IIgs format 0-15 (Page 16-35 Toolbox Ref Vol 2).

BackColor

Indicates the background color for text in this stream in IIgs format 0-15 (Page 16-35 Toolbox Ref Vol 2).

Position

The number of pixels to shift this text up or down from the normal baseline. Positive numbers shift character position down; negative number shift character position up. 0 displays "normally."
NOTE: Actual superscripting/subscripting should be done via the FontStyle field.

CharSpacing

The number of pixels to add or subtract after each character in this text stream ("kerning"). Positive numbers increase space; negative numbers decrease space. *NOTE: This effect is usually accomplished by setting chExtra, so the limitations of using chExtra apply.*

LineSpacing

The line height (from baseline to baseline), expressed as a point size. 0 means "automatic" line spacing. To achieve double or triple spacing, set LineSpacing to double or triple the current FontSize value.

SpaceBefore

The number of pixels to leave blank before a paragraph.

SpaceAfter

The number of pixels to leave blank after a paragraph.

FirstIndent

The offset (in pixels) from the leftmost text position for the left side of the first line of a paragraph.
NOTE: The FirstIndent and LeftIndent are not relative to each other; they are both given as an offset from the leftmost text position.

LeftIndent

The offset (in pixels) from the leftmost text position for the left side of lines 2-X of a paragraph.
NOTE: The FirstIndent and LeftIndent are not relative to each other; they are both given as an offset from the leftmost text position.

RightIndent

The offset (in pixels) from the rightmost text position for the right side of all lines of a paragraph.

Justification

The paragraph justification: 0=Left, 1=Center, 2=Right, 3=Full

TabArray

The tab array defines 16 tab stops. Each TabArray element is formatted as follows:

$00	TabLeader	Byte–ASCII character to use for a tab leader (or filler) character
$01	TabType	Byte–0=No Tab Stop; 1=Left; 2=Center; 3=Right; 4=Decimal
$02	– TabPosition –	Word–The offset (in pixels) from the leftmost text position for the tab.

Tab stops can be defined in any order. While there must be 16 tab stops defined, TabType entries of 0 (no tab) can appear anywhere, even between actual tab stops.

TabLeader entries of 0 mean “no leader.”

Options

A flag word indicating various paragraph options:

Bit0	Keep lines of this paragraph together
Bit1	Keep this paragraph and the next paragraph together
Bits2-15	Reserved (must be clear)

Border

This long is treated as 4 separate flag bytes (one each for the Top, Bottom, Left and Right of the paragraph). Each flag byte looks like this:

Bit0-Bit2	Thickness (in pixels) of line 1 (closest to paragraph)
Bit3-Bit5	Thickness (in pixels) of line 2 (further away from paragraph)
Bit6-Bit7	Space (in pixels) between the two lines (00=no space; 01=space the thickness of line 1; 10=space the thickness of line 2; 11=space the absolute difference between line 1 and line 2).

Page Length

The length of the page in pixels. This is from the absolute top edge to the bottom edge, including top and bottom margins and the main body.

Page Width

The width of the page in pixels. This is from the absolute left edge to the right edge, including any side margins.

Section Rect

The rectangle of the current section of text. This rectangle is relative to the whole page (where the top and left are 0,0). This should change when moving between headers, footers, and main bodies, etc. The “Indent” values are based from this rectangle.

Columns

The number of columns in this section.

Gutter

The space between adjacent columns, in pixels.

PicHandle

Data for a standard QuickDraw II Picture (see Apple IIGS Tech Note #46). NULL if no picture is being passed.

Implementation Notes

IMPORTANT: Beware of the handles! By definition, any handle you are passed belongs to you! If you are passed a PicHandle you must be sure to dispose of it, even if your application or translator does nothing with pictures embedded in text!

Imagine we are importing into an application that only supports body text. We have a Microsoft Word document that has two “sections” (a section is like a “mini-document”…each section can have its own headers, footers, body text, and footnotes).

In the first section there’s a header on every page (no footer) and the single-spaced body text is “*This is* a test”.
In the second section there’s a footer on every page (no header) and the double-spaced body text is “**of the** Word importer.”

Importing this file would generate these actions:

The translator sends “GetSettings” so the application will initialize the data record with all its current settings.

The translator reads a hunk of the file and finds the odd and even header, so it says “OddHeader.” The application responds “SkipThat.” The translator says “EvenHeader.” The application responds “SkipThat.”

The translator finds no footers but does encounter the “*This is* a test” body text, so it sends “BodyText.” The application responds “OK” because it wants the body text.

The translator fills in the data record:
TextLength=8; Text=“This is ”; Action=Continuing; line height=single; style=italic

App stores text and calls BFRead

The translator fills in the data record:
TextLength=6; Text=“a test”; Action=Continuing; style=plain

App stores text and calls BFRead

The translator is done with the first section so it begins working on the new section. It reads a hunk of the file and finds the odd and even footer, so it says “OddFooter.” The application sees the action is no longer “Continuing”…at this point it has the complete BodyText from the first section. Because sections don’t matter to the application, the application continues the import in case there is more “body text” in another section. The application responds “SkipThat” (in response to “OddFooter”). The translator says “EvenFooter.” The application responds “SkipThat.”

The translator says “BodyText” and the application responds “OK” because it wants body text.

The translator fills in the data record:
TextLength=7; Text=“of the ”; Action=Continuing; line height=double; style=bold

App stores text and calls BFRead

The translator fills in the data record:
TextLength=14; Text=“Word importer.”; Action=Continuing; style=plain

App stores text and calls BFRead

The translator sees that there is nothing more to import, so it disposes of the data record and sets the XferRec’s Status to bfDone (which causes Babelfish to call TrShutDown). When the app gets bfDone it disposes of the XferRec and uses the imported text.

Questions and Answers

When an application says it can import more than a single kind of data, how does my TrStartup(ForImport) code determine which ONE of those formats to return?

You just determine the best format to return. *NOTE: After determining the format that you'll be using, remember to change the DataKinds field to indicate just that single kind!*

If you're writing a *font* translator that also can return a picture of the font, your best format is a *font*. If an application says it can take either a font or a picture, then you would return the *font* because that is your *native and best format*. Consider this: if an application says it can take a font *and* a picture (granted, a very odd combination) then it is probably able to take the returned font and make a picture of it (but it could not easily take a picture and make a font of it!).

If you're writing a translator that loads a *QuickDraw II Picture*, but it can also return a PixelMap, your best format is a *QuickDraw II Picture*, so that is your first and best choice.

What Next?

Data Formats

We have only defined a few data formats so far. And the ones we defined were ones that we had some knowledge of. As time goes on we hope that more formats will be defined. Data formats will then be added to this document.

Translators

Without translators, Babelfish has nothing to manage! If you want to work on a particular translator, let Dave Hecker (support@sevenhills.com) know; he'll let you know if someone else is already working on it (no sense duplicating effort).

Applications

Without applications that use Babelfish there's no point!

Seven Hills plans to convert all its products to require Babelfish for import/export. Steve and Joe at GS+ Magazine have already modified EGOed to support it. We hope you'll consider supporting or requiring Babelfish in your products.

Currently, SuperConvert and Spectrum both support Babelfish, more applications will be added as time goes on.