Internet Configuration System

Programming Documentation

1.0

Quinn
3 Dec 1994

1 Introduction

This document describes the programming interface to the Internet Configuration System. You should read this document if you intend to write a program that records user preferences related to the Internet or you need a simple mechanism for mapping an extension to a file type and creator.

The document begins with an introduction to the system. Everyone should read this. It continues with a chapter describing how to use the system to access Internet preference. Everyone should read this too! The third chapter discusses the requirements of a functional Internet Configuration program. Finally there are two reference chapters that describe the application programmer interface (API) in detail.

1.1 Goals and Design

The goal of the Internet Configuration System is to simplify the Macintosh user's experience of the Internet. The primary focus will be to reduce the number of times that the user is required to enter information like their Email address.

Another important design goal was programmer simplicity. We recognised that this system would not be adopted if it was too complicated to use. Another aspect of this is that the system should be available in all major development environments.

The core of the system is a shared file, the Internet Preferences file, that contains this common preference information and an Internet Configuration application which the user runs to modify these preferences. This design was complicated by the requirement that it be capable of supporting Macs in labs. This requires that the system support search path for looking for the preference file.

One important design goal was to allow the system to be extended in the future to support new ideas such as application and user specific preferences. To achieve this we have complicated the system slightly by introducing an Internet Config Extension. This is a component that can be used to extend the system without relinking existing applications.

1.2 System Requirements

The Internet Configuration System relies on no modern system features and applications built using the system can be compatible with System 6 (and most probably System 4.1). The system will exploit the following advanced system features if they are available:

- Folder Manager will be used to find the Preferences folder. If Folder Manager is not present then the System Folder will be used instead.
- Alias Manager will be used to resolve aliases to preference files.
- Component Manager will be used to locate the component that implements the API. If Component Manager is unavailable, or the component is not registered,

then a glue implementation will be used.
- Gestalt Manager will be used to check for the presence of the previous two managers. If is it not present then MPW Gestalt glue will be used.

The system can be used from, and has been tested with, the following development environments:

- Think Pascal
- Think C
- MPW Pascal
- MPW C
- Metrowerks C (68K and PPC)

The system has limited support for PowerMac development environments. This includes a ‘universal’ C interface and source code for C glue to call the component. Because all PowerMac machines have Component Manager we assume that PowerMac applications will rely on the component and there is no glue implementation for the PowerMac environment.

1.3 Parts of the System

The system is made up of four major parts. The first is the API, as defined in the interface files ICTypes.[ph], ICAPI.[ph] and ICKeys.[ph]. These provide the declarations required to use the system.

The second part is a component, Internet Config Extension, that implements the functions of the API in an extensible and patchable manner. The Internet Configuration application contains a copy of this extension and installs and registers the component when it is run. Obviously this is only possible if the Component Manager is present and the system deals with this in the following way.

The third part is an MPW object file, ICGlue.o, that you link with your program. This glue performs two functions. When you use it it checks for the presence of the Component Manager and the Internet Configuration component. If they are both present then the glue routes all calls directly through to the component. If they are missing then the glue implements the calls itself.

The MPW object file will obviously be of no use in PowerMac development environments. There is a preliminary implementation of the C code to call the component from PowerMac C environments (ICPPCGlue.c).

Finally there is a preferences file, Internet Preferences, which is usually kept in the Preferences folder. This file holds the actual preference data and is modified by either the component or the glue depending on which system is in use at the time. This files is actually an implementation detail and its presence is only dictated by the current systems. Future systems may store preferences in a completely different manner.

1.4 User Interface

The primary user interface to the Internet Preferences file is the Internet Config application. Although this application can be replaced it is important that you do not attempt to duplicate its functionality within your own application. Otherwise the focus of your application will be lost; it will become unclear whether your program is a newsreader or a preference setting program.

If you dislike the Internet Config application's user interface then you should write a small focused application that replaces it. You can choose to replace it in its entirety or just some component of it. For example, it would be quite sensible to write a small focused application that replaces the Internet Config application's File Types dialog with something altogether less modal.

One of the problems with providing an interface to changing preferences is that your interface will have to be modified to keep up with any modifications made to the Internet Config system. For example, the Internet Config RandomSignature extension will obsolete any user interface you provide for changing the signature.

So, in general, we recommend that you do not provide the ability to change the Internet Preferences within your application. You might want to provide an easy mechanism for launching Internet Config so that the user can change these preferences more quickly. Internet Config may be extended to provide some mechanism for doing this. Please send us your thoughts on this issue.

If you do provide a mechanism to change preferences that you should make sure to pay attention to the locked attribute. Any extension that is not compatible with a simple preference changing user interface will set that attribute. You can use the Internet Config ReadOnly extension to test how well your application supports locked preferences.

1.5 Finding a Preference File

One of the most complicated aspects of the current Internet Configuration System is the method used to find a preferences file. Although, as stated in the previous sub-section, the preference file is an artifact of the implementation it is an important artifact and requires you to specify the search path for the preference file. You supply a search path by giving the program a list of folders to search.

The algorithm for finding the preference file is as follows:

- search each of the folders in the search path
- then search the preferences folder
- when searching a folder, first looking for a file of the right name, then looking for any file of the right type
- follow all aliases to files of the appropriate type

This mechanism supports the use of Macs in laboratory situations. Programs like Eudora and NewsWatcher can specify the folder containing the launched preferences file in the preference search path. This way the Internet Config System will find the user's

preference file rather than the one in the Preferences folder. Preferences can be shared between applications using aliases.

1.6 Preferences and Their Attributes

A program gets a **preference** using the call ICGetPref. The program specifies the preference using a **key** which is simply a Str255 that uniquely identifies the preference. Keys are not case sensitive and all high-bit set characters are either reserved or have a special meaning. The current list of keys is defined in Appendix A.

The system responds to a ICGetPref request by returning a chunk of data that is the value of the preference. Some keys will return data of a fixed size; other data may be variably lengthed. There is no practical limit to the size of key data.

Each preference also has an **attribute** which is a long word that contains flags that provide additional information about the preference. The currently defined attributes include a locked bit and a volatile bit. The locked bit defines whether a request to modify the key's data will succeed. The volatile bit is discussed in the section on caching.

1.7 Future Extension (he he he!)

One of the most important designed goals of the system is that it can be extended easily. This is achieved in two different ways.

Firstly the key space (remember keys are defined by Str255) is huge and more keys can be added as more common preferences are requires. Secondly the system can be patched by replacing the core component with a later, and hopefully improved, version. Replacement components do not even have to implement the entire API; they can partially override an existing component using the Component Manager's capturing facility.

An example of this behaviour is the Random Signature extension, which ships as part of the system.

2 Using Internet Config Services

This chapter describes how a normal Internet aware application would access, and even modify, the common Internet preferences. This chapter is important for anyone developing Internet application.

2.1 Starting Up and Shutting Down

When you start your application you should call ICStart and give it your application creator. If the system starts correctly it returns you an ICInstance. This is a private type whose only use is to supply back to the system.

When your application shuts down it is important that you call ICStop with the ICInstance returned by ICStart. You should not call ICStop if ICStart fails.

```
program Main;
    var
        inst : ICInstance;
        err, junk : ICError;
begin
    err := ICStart(inst, my_creator);

    if err = noErr then begin
        err := DoMyApplication;
        junk := ICStop(inst);
    end; (* if *)
end; (* Main *)
```

The creator is not used in the current implementation but may be used in the future to support application specific preferences.

2.2 Specifying the Search Path

Once you have started the system you should then specify which folders the system should search to locate the preference file. You do this by calling ICFindConfigFile, passing it an array of folders to search. The system searches these folders in order, from first to last, and then the Preference folder. The search algorithm was given in the previous chapter.

Note that if you have no special search requirements then you should just call ICFindConfigFile with a count of 0 and a folders pointer of nil. The system will then just look for the preferences in the Preferences folder.

The following code fragment shows how to call ICFindConfigFile in the more complicated case.

```
function DoSetupSearchPaths : ICError;
    var
        folder_spec: ICDirSpecArray;
begin
    folder_spec[0].vRefNum := -1;              (* search for prefs in root of the system *)
    folder_spec[0].dirID := 2;                 (* volume, obviously you'd use other things *)
    DoSetupSearchPaths := ICFindConfigFile(inst, 1, @folder_spec) : 1);
```

```
end; (* DoSetupSearchPaths *)
```

Notice how the routine initialises the folder_spec array to contain the list of folders it wants the system to search. It does not have to specify the Preferences folder because the system always searches that folder last.

2.3 Getting Preference Information

Once you have told the system where to find a preferences file you can then proceed to get data out of the file. To do this you must first call ICBegin, access the required preferences using ICGetPref and ICSetPref and then terminate this access by calling ICEnd.

The following routine demonstrates this process.

```
function GetEmailAddress : Str255;
    var
        err, junk : ICError;
        size : longint;
        result : Str255;
        attr : ICAttr;
begin
    err := ICBegin(inst, icReadOnlyPerm);
    if err = noErr then begin
        size := sizeof(result);   (* max size for returned data *)
        err := ICGetPref(inst, kICEmail, attr, @result, size);
        junk := ICEnd(inst);
    end; (* if *)
    if err <> noErr then begin
        result := '';
    end; (* if *)
    GetEmailAddress := result;
end; (* GetEmailAddress *)
```

This routine only accesses one preference but normally you would access all the preferences you're interested in between the ICBegin and ICEnd.

It is important that you call ICEnd if and only if ICBegin does not return an error. It is also important that you do not allow other applications to run (for example, by calling WaitNextEvent) between the ICBegin and ICEnd.

2.4 Preference Coherency

The Internet Preferences file is a shared data structure that can be accessed by multiple programs. This obviously causes consistency problems if an application reads a preference and it is then modified by some other application. Internet Config provides three mechanisms to deal with this problem. These are discussed in order of correctness.

The **On Demand** approach requires that the application read its preferences when it actually needs them. Because the application does not hold copies of the preference, it can safely ignore the coherence problem. The primary problem with this approach is that it requires that you track down all references to specific preferences and change them to calls to Internet Config. This may or may not be easy depending on how your code base

is structured.

The **Cache Watching** approach allows applications to get preferences and then look for modifications to those preference. It is centred around the preference seed, which is a number that monotonically increases whenever the preference data changes. You should get this seed and remember it immediately after reading your preference information. You should then get it at regular intervals and flush any cached preferences if the seed changes. For example, the pseudocode for your application might look like the following.

```
program Frog;
begin
    start system
    get my preferences
    err := ICGetSeed(inst, seed);
    while not quit do begin
        process events
        err := ICGetSeed(inst, new_seed);
        if new_seed <> seed then begin
            reread my preferences
            seed := new_seed;
        end; (* if *)
    end; (* while *)
end; (* Frog *)
```

The cache watching approach is further complicated by the volatile attribute. If you get a preference and it has this attribute set then you should not cache that preference. This feature allows certain preferences to change dynamically without affecting the seed.

The final approach is the **Ostrich** approach. In this mechanism you just get your preferences and ignore caching issues entirely. This approach has the advantage of being the easiest to implement although it does mean that your application will not work properly at all times.

The approach you choose is up to you. We highly recommend that applications take the on demand approach but recognise that this may be difficult to do with an existing code base.

Regardless of which approach that you take you must flush any cached preferences when you launch. The seed value is not valid across reboots.

2.5 Indexing All Preferences

In some cases you might want to index through all of the preferences. You can do this using the ICCountPref and ICGetPref routines. The following routine shows how this is done.

```
procedure DumpKeys;
    var
        err : ICError;
        ndx : longint;
        count : longint;
        key : Str255;
```

```
begin
    err := ICCountPref(count);
    if err = noErr then begin
        for ndx := 1 to count do begin
            err := ICGetIndPref(inst, ndx, key);
            if err = noErr then begin
                writeln(key);
            end; (* if *)
        end; (* for *)
    end; (* if *)
end; (* DumpKeys *)
```

3 Internet Config Application Requirements

This chapter describes some of the technical details for writing the Internet Configuration application. Most readers will not be interested in the details contained in this chapter.

3.1 Basic Operations

The Internet Configuration application has the basic ability to create and edit Internet preferences files, which the application just views as documents. When it is launched without a specific document the application opens the Internet Preferences file in the Preferences folder. When it is launched with a document it edits that document. The documents can then be edited, closed, saved, etc as per the usual Mac interface.

The actual user interface used to edit preferences is beyond the scope of this discussion.

3.2 Component Installation

When it is launched the application first checks the version of the Internet Config Extension in the Extensions folder. If it is not present it should create it; if it out of date it should update it. This operation involves creating the file and copying a bunch of resources from the application's resource fork into the file. The application should does this regardless of whether the Component Manager is present in the hope that it will eventually become useful.

Because it is in the Extensions folder the component will be automatically be registered the next time the system starts up. However, in order to make the component available immediately the application registers the component if the Component Manager is present.

Note that from now on the Internet Configuration application access the Internet Preferences file using exactly the same API as every other program. Well, more or less. There are some API calls that are designed for use specifically by the Internet Configuration application and are not recommended for use by normal applications.

3.3 Setting Up Default Values

The application can now start a session using ICStart. When it opens a preference file the application makes sure that every preference that has a meaningful default value is initialised to that value.

Note that download folder is tricky one in that it is not a static value; the application must create the 'alias' to the desktop folder on the fly.

In future the application might set up the ArchiePreferred, InfoMacPreferred and UMichPreferred based on the machines reverse DNS name.

3.4 Editing Configuration Documents

The API provides one extra routine for the configuration application, namely ICSpecifyConfigFile. This allows you to open a config file in any folder directly, without having to mess around with the search path.

4 API Reference

This chapter is divided into two sections. The first section describes the types and constants provided by the interface. The second section describes the routines provided by the interface.

4.1 Types and Constants

This section describes the types and constants provided by the Internet Configuration System. These are provided in ICTypes.[ph].

The following error codes can be returned by the system.

```
icPrefNotFoundErr = -666;                   (* preference not found (duh!) *)
icPermErr = -667;          (* cannot set preference *)
icPrefDataErr = -668;      (* problem with preference data *)
icInternalErr = -669;                       (* hmm, this is not good *)
icTruncatedErr = -670;                      (* more data was present than was returned *)
icNoMoreWritersErr = -671;                  (* you cannot begin a write session because someone
                                             else is already doing it *)
```

The ICAttr type is simply a longint containing flags that describe the attributes of a key and its data.

```
ICAttr = longint;                           (* type for preference attributes *)
```

The ICattr_no_change constant is used when you call ICSetPref and do not want to mess around with attributes. You can supply this value and the system will not change the attribute of the preference.

```
ICattr_no_change = -1;                      (* supply this to ICSetPref to tell it not to change
                                            the attributes *)
```

The following bits are defined in the ICAttr type:

```
ICattr_locked_bit = 0;                      (* bits in the preference attributes *)
ICattr_locked_mask = $00000001;             (* masks for the above *)
ICattr_volatile_bit = 1;
ICattr_ volatile _mask = $00000002;
```

If the locked bit is set then any attempt to set the preference will result in an error. If the volatile bit is set then you should not cache the value of this preference because it is subject to non-seed changing changes. See the section on caching preferences for more information about this issue.

The following values define the file type, creator and default name of the Internet Preferences file.

```
ICfiletype = 'ICAp';
ICcreator = 'ICAp';
ICdefault_file_name = 'Internet Preferences';  (* default file name, for internal use,
                                            overridden by a component resource *)
ICdefault_file_name_ID = 1024;              (* ID of resource in component file *)
```

The ICDirSpec record is used to hold the vRefNum and dirID of a directory. An array of these is supplied to ICFindConfigFile to specify the search path. This array is defined to contain just 4 elements but is in fact arbitrarily extensible.

```
ICDirSpec =
    record                                  (* a record that specifies a folder *)
        vRefNum: integer;
        dirID: longint;
    end;
ICDirSpecArray = array [0..3] of ICDirSpec;    (* an array of the above *)
ICDirSpecArrayPtr = ^ICDirSpecArray;           (* a pointer to that array *)
```

The ICError type is used for all error results from the system. A longint is used because we make lots of calls to Component Manager which uses longints for error codes.

```
ICError = longint;                          (* type for error codes *)
```

The ICInstance type is an opaque type that is used to hold a reference to a session with the Internet Configuration System. Applications can create instances by calling ICStart, used them with any of the API routines and destroy them by calling ICStop.

```
ICInstance = Ptr;                           (* opaque type for preference reference *)
```

The ICPerm type is used to specify whether you wish to access the preferences for read-only or read-write.

```
ICPerm = (ioNoPerm, icReadOnlyPerm, icReadWritePerm);
```

4.2 Routines

The following routines are available in ICAPI.[ph].

```
function ICStart (var inst: ICInstance; creator: OSType): ICError;
```

You should call this routine at application initialisation time, passing it your creator type. If it returns noErr then you are all set to use the system. If it returns an error then ICInstance will be nil and you should not call ICStop.

```
function ICStop (inst: ICInstance): ICError;
```

You should call this when your application terminates, passing it the instance you got from ICStart.

```
function ICFindConfigFile (inst: ICInstance; count: integer; folders: ICDirSpecArrayPtr): ICError;
```

This routine tells the system where to look for a configuration file. You must call this before calling ICBegin. You should pass in a pointer to an array of ICDirSpecs that defines in which folders the system should look for the preference file. You should pass in count as the number of valid entries in this array. You do not have to supply the Preferences folder; the system will search there automatically if it can’t find anything in

the specified folders. You can set folders to nil if and only if count is 0.

You must specify a config file, using either this routine or the next, before you can access any preferences.

```
function ICSpecifyConfigFile (inst: ICInstance; config: FSSpec): ICError;
```

This routine is intended for use by the Internet Configuration application only. It tell the system to use a specific configuration file, bypassing the search mechanism used by ICFindConfigFile.

```
function ICGetSeed (inst: ICInstance; var seed: longint): ICError;
```

This routine returns the seed for the current preferences. The seed is a value that monotonically increases when any preferences are changed. You can repeatedly call this routine to determine whether any preference information you are storing is out of date. The value returned by ICGetSeed is only valid until the machine reboots, which basically means that you should not use this values across repeated launches of your application. The seed value is not valid inside a pair of ICBegin and ICEnd calls. You should sample the seed after calling ICEnd.

```
function ICGetPerm (inst: ICInstance; var perm:ICPerm): ICError;
```

This routine returns the current permissions for this instance. This routine is not very useful for applications. It was included so that overriding components could obtain this information easily.

```
function ICBegin (inst: ICInstance; perm: ICPerm): ICError;
```

This routine prepares the system to read (set perm to icReadOnlyPerm) or read and write (set perm to icReadWritePerm) preferences. You should call this routine before calling ICGetPref or ICSetPref. If this routine returns an error than you cannot access the preferences. If it returns noErr then you should proceed to access your preferences and eventually call ICEnd. You must not let any other application run, by calling WaitNextEvent or any other routine that gives time, between these calls.

Except where otherwise noted, any attempt to read, delete or write preferences without calling ICBegin will result in a paramErr.

```
function ICGetPref (inst: ICInstance; key: Str255;
            var attr: ICAttr; buf: Ptr; var size: longint): ICError;
```

This routine gets a preference’s data given its key. It puts the data into a buffer that you supply. It also returns the attributes in attr. You should point buf to the beginning of your buffer and set size to its size. You can also use this routine to just get information about the preference by setting buf to nil.

You do not need to call ICBegin before calling this routine. If you do no do so then this routine will automatically called ICBegin(inst, icReadOnlyPerm) on entry and ICEnd(inst) on exit.

Key must not be the empty string. If buf is nil then no data is returned and the value of size is ignored; otherwise the value of size must not be negative and is the size of the buffer pointed at by buf. If the preference is present then the call sets attr to be the preference's attributes, size to be the preference's true size and returns noErr.

The routine may return icTruncatedErr if the buffer's size is too small to hold the data. In this case attr is valid, size contains the total size of the preference and the system has placed as many bytes of the preferences as will fit in the buffer. You may want to increase the size of the buffer and refetch the preference to recover the lost data.

On other errors the system returns attr as ICattr_no_change and size as 0. The most common error, icPrefNotFoundErr, implies that the preference associated with key is not available.

```
function ICSetPref (inst: ICInstance; key: Str255; attr: ICAttr; buf: Ptr; size: longint): ICError;
```

The routine sets a preference given its key, attributes and a buffer containing the preference data. You can not set the attributes by specifying an attribute of ICattr_no_change. You can not set the data by setting buf to nil. Not setting both values has no effect if the preference already exists but creates an empty preference with the default attributes otherwise.

You do not need to call ICBegin before calling this routine. If you do no do so then this routine will automatically called ICBegin(inst, icReadWritePerm) on entry and ICEnd(inst) on exit.

Key must not be the empty string. If buf is nil then the value of size is ignored; otherwise it must be the non-negative size of the data to store. If the preference is successfully modified then the routine returns noErr.

The routine returns icPermErr if the perm parameter to ICBegin was icReadOnlyPerm.

The routine also returns icPermErr if the current attr is locked, the new attr is locked and buf is not nil.

```
function ICCountPref (inst: ICInstance; var count: longint): ICError;
```

This routine returns the total number of preferences available. If it returns an error then count will be 0.

```
function ICGetIndPref (inst: ICInstance; n: longint; var key: Str255): ICError;
```

This routine returns the key associated with the Nth preference. The value of n must be positive. The routine returns icPrefNotFoundErr if n is beyond the last preference.

```
function ICDeleteKey (inst: ICInstance; key: Str255): ICError;
```

This routine deletes a preference given its key. The routine returns icPrefNotFoundErr if

the preference does not exist.

```
function ICEnd (inst: ICInstance): ICError;
```

This routine tells the system that you have finished accessing preference information. You must have successfully called ICBegin to call this.

```
function ICDefaultFileName (inst: ICInstance; var name: Str63): ICError;
```

This routine returns in name the name of the default file name of the Internet Preferences file. This name is used during the preference search and is also the name used to create a preference file if none is found.

This value is hardwired into the glue implementation but is set by a resource in the component version. The component calls itself to set up the default name so a capturing component can override this action.

```
function ICGetComponentInstance (inst: ICInstance; var component_inst: univ Ptr): ICError;
```

This routine returns the component instance being addressed by the glue. It returns an error and nil if the glue is using its built in routines.

Note that the type of component_inst in univ Ptr rather than Component Instance so that applications using ICAPI.[ph] do not need access to Components.[ph] which is not always available.

5 Component API Reference

This section documents the component interface to the Internet Configuration System. If you are sure that your application will be used only on systems that support the Component Manager you can talk to the component directly using this interface. About the only advantage of this is that you avoid having to link with a pile of glue that you're never going to use.

5.1 Routines

```
function ICCStart (var inst: ComponentInstance; creator: OSType): ICError;
```

If routine is glue that checks for the presence of the Component Manager and the Internet Configuration component. If it finds them it creates an instance, initialises it and returns it. If it can't find them or the initialisation fails then it returns badComponentInstance and inst is nil.

```
function ICCStop (inst: ComponentInstance): ICError;
```

This routine is glue that shuts down the instance and closes it.

The following routines correspond exactly with their equivalents in ICAPI.[ph].

```
function ICCFindConfigFile (inst: ComponentInstance;
            count: integer; folders: ICDirSpecArrayPtr): ICError;
function ICCSpecifyConfigFile (inst: ComponentInstance; config: FSSpec): ICError;
function ICCGetSeed (inst: ComponentInstance; var seed: longint): ICError;
function ICCGetPerm (inst: ComponentInstance; var perm:ICPerm): ICError;
function ICCBegin (inst: ComponentInstance; perm: ICPerm): ICError;
function ICCGetPref (inst: ComponentInstance; key: Str255;
            var attr: ICAttr; buf: Ptr; var size: longint): ICError;
function ICCSetPref (inst: ComponentInstance; key: Str255;
            attr: ICAttr; buf: Ptr; size: longint): ICError;
function ICCCountPref (inst: ComponentInstance; var count: longint): ICError;
function ICCGetIndPref (inst: ComponentInstance; n: longint; var key: Str255): ICError;
function ICCDeleteKey (inst: ComponentInstance; key: Str255): ICError;
function ICCEnd (inst: ComponentInstance): ICError;
function ICCDefaultFileName (inst: ComponentInstance; var name: Str63): ICError;
```

6 Overriding Components

Internet Config provides a powerful mechanism for programmers to override the default implementation of the preference code in a large number of applications. Anyone wielding this power should be careful that they only use it for the forces of niceness and good! Remember that any Internet Config component you write is dynamically linked into the application and the applications will expect you to behave certain rules. Some of these rules are given in this chapter but this chapter can never be exhaustive. Please use some common sense.

6.1 Sample Code

The current distribution provides two sample component that capture and extend the behaviour of the default Internet Config component. These sample programs are not final code and are only a guideline for implementors. If you are interested in writing overriding component then you should be aware of their limitations. Please talk to us before you use these as the basis for your new, Way Cool™ overriding component.

6.2 Preference Consistency

It is critical that your component maintains the following invariant: if any preferences that are not marked as volatile change then the seed must increase. If you do not maintain this invariant then you will break applications that are using the Cache Watching approach to preference consistency.

6.3 Seeds and ICBegin/ICEnd

It is critical that your application does not clash with programs using the Cache Watching approach to maintain preference consistency. This is surprisingly difficult to do! The important things to remember are that the seed in not valid within ICBegin/End pairs and that applications sample the seed after calling ICEnd. Conceivably it would be sensible to cache preference modifications inside ICBegin/ICEnd pairs and only write those preferences on the call to ICEnd. This would obviously modify the seed, which is perfectly sensible and would not cause any problems because the application shouldn’t have sampled the seed yet. There is a possible problem if you want to change the preferences at ICEnd time and you want the application to notice these changes. The solution is to make the next two seeds return different values. Note that you shouldn’t always return different values otherwise applications will be continuously refetching their cached preferences.

6.4 Readers and Writers

The Internet Config system uses a single writer or multiple readers approach to preference consistency. This means that either one writer or multiple readers can be accessing the preferences at any given point in time. Your component should enforce this restriction. You can determine whether a program is a reader or writer by watching the ICBegins.

At the moment, we're not entirely sure whether the current implementation actually *does* enforce this restriction. This is entirely besides the point!

6.5 Override Problems

The current implementation of the component supports component targeting, but not consistently. For example it calls the target component for default file name but not for the implicit ICBegin/ICEnd calls around a ICGetPref or ICSetPref and not for the ICGetPerm call. Your overriding component will have to deal with this.

6.6 Locking Preferences

If you override a preference in such a way that a standard user interface is no longer appropriate for changing the preference then you should make sure to mark that preference as locked. This will prevent applications that provide their own user interface for changing preferences from attempting to change the preference. A good example of this is the RandomSignature component which locks the signature so that programs do not allow the user to edit the signature using their own user interface. Obviously if you do this then you need to provide an alternative interface for editing the preference.

A Current Keys

Keys must are Str255s that are case insensitive. All high bit set characters are either reserved or defined to be special. The special characters are described later in this appendix.

A.1 Key Types

ICKeys.[ph] defines a number of data types that are the type of various preferences. The following types are defined:

PString
Pascal formatted string. The data is of minimal length.

STR#
The is same format as a STR# resource, ie count word followed by packed strings.

TEXT
This is the same format a TEXT resource, ie straight characters.

ICFontRecord
This is used to specify a font, size and face and is defined as:

```
ICFontRecord = record
        size: integer;
        face: Style;
        font: Str255;
    end;
```

ICAppSpec
This is used to specify an application and is defined as:

```
ICAppSpec = record
        fCreator: OSType;
        name: Str63;
    end;
```

ICFileInfo
This is used to specify a file type and creator and is defined as:

```
ICFileInfo = record
        fType: OSType;
        fCreator: OSType;
        name: Str63;
    end;
```

ICFileSpec
This is used to specify a file or folder and is defined as:

```
ICFileSpec = record
        vol_name: Str31;                (* volume that file is on *)
        vol_creation_date: longint;     (* creation date of said volume (poor man's alias) *)
        fss: FSSpec;                    (* vRefNum field contains nothing of value *)
        alias: AliasRecord;             (* plus extra data, aliasSize 0 means no Alias Manager
    end;                                present when ICFileSpec was created *)
```

ICCharData

This is used to specify a mapping from Macintosh ASCII to net ASCII and vice versa. If is defined as:

```
ICCharTable = record
        net_to_mac: packed array[char] of char;
        mac_to_net: packed array[char] of char;
    end;
```

ICMapData

This is used to specify extension and MIME mappings. It is discussed in detail in Appendix B.

ICServices

This is used to the mapping between TCP service names and their ports. The data returned is an ICService record, which contains a count followed by an unbounded array of ICServiceEntries.

```
ICServices =
    record
        count: integer;
        services: array[1..1] of ICServiceEntry;
        (* this array is packed, so you can't index it directly *)
    end;
ICServicesPtr = ^ICServices;
ICServicesHandle = ^ICServicesPtr;
```

Note that each element in the array is tightly packed, which means you can't index the array directly. The format of an ICServiceEntry is as follows:

```
ICServiceEntry =
    record
        name: Str255;     (* this strings is tightly packed *)
        port: integer;                          (* which means, these fields might have an *)
        flags: integer;                         (* odd address *)                    end;
ICServiceEntryPtr = ^ICServiceEntry;
ICServiceEntryHandle = ICServiceEntryPtr;
```

The bits in the flags field are defined as:

```
ICservices_tcp_bit = 0;                 (* this is a TCP service *)
ICservices_tcp_mask = $00000001;
ICservices_udp_bit = 1;                 (* this is a UDP service *)
ICservices_udp_mask = $00000002;
```

It is possible for both the UDP and TCP bits to be set, which means that the service is available via both protocols.

A.2 Key Space

The is currently only one special character in key strings, namely the bullet "•". This is used in two places. Firstly it allows applications to store application private preferences using the mechanism described below. Secondly it is used as a field separator for indexed entries, such as the "Helpers•" entry. Indexed entries rely on the fact that the bullet character is not valid within normal keys, so all the keys beginning with "Helper•" must be helper mapping entries.

If an applications wishes to store a private preference then it should prepend its key with the hexadecimal representation of its creator type and a bullet.

For example, the Internet Config application stores its window positions with the following key: 49434170•WindowPositions.

You can register more keys with the keeper of the Internet Configuration application.

A.3 Currently Defined Keys

The following keys are defined in the initial implementation. These keys are documented in ICKeys.[ph] and you should look in that file for the most up-to-date list of keys.

RealName PString user's real name
Email PString user's Email address
MailAccount PString user's mail account (user@host)
MailPassword PString password for above (scrambled)
NewsAuthUsername PString user name for news authorisation
NewsAuthPassword PString password for above (scrambled)
ArchiePreferred PString preferred Archie server (formatted)
ArchieAll STR# list of Archie servers (formatted)
UMichPreferred PString preferred UMich mirrors (formatted)
UMichAll STR# list of UMich mirrors (formatted)
InfoMacPreferred PString preferred InfoMac mirrors (formatted)
InfoMacAll STR# list of InfoMac mirrors (formatted)
PhHost PString Ph server (host)
WhoisHost PString whois server (host)
FingerHost PString finger server (host)
FTPHost PString default FTP server (host)
TelnetHost PString default Telnet host (host)
SMTPHost PString selected SMTP server (host)
NNTPHost PString selected NNTP server (host)
GopherHost PString selected default Gopher server (host)
WWWHomePage PString selected WWW home page
WAISGateway PString selected WAIS Gateway
ScreenFont ICFontRecord preferred font for screen displays
PrinterFont ICFontRecord preferred font for printing
DownloadFolder ICFileSpec location of folder to store downloads
Signature TEXT user's signature
Organization PString user's organisation
Plan TEXT user's plan
QuotingString PString preferred quoting string
MailHeaders TEXT extra mail headers
NewsHeaders TEXT extra news headers
Mapping ICMapData extension and MIME mappings
CharacterSet ICCharTable default character translation
Helper•*url_prefix* ICAppSpec helper application for URL prefix
Services ICServices port number to name mappings

A.4 Scrambled PStrings

The scrambling algorithm for PStrings is as followings:

```
for i in 1 .. length(str)
    str[i] := str[i] xor ($55 + i);
```

A.5 Formatted PString and STR# Preferences

Both PStrings and each entry in STR# preferences can be **formatted** to contain all the required information about a service. Formatted strings contain 3 fields, each separated by colons ":". The first field contains the user displayable name for a specific service. The second field contains the machine's DNS name. The final field contains the path, which is empty for Archie servers. A typical formatted string would be "Australia:archie.au:/micros/mac/info-mac".

A.5 Host PStrings

A large number of **host** PStrings are used to denote default services. Internet Config itself does not interpret these strings but applications are required to. The format is as follows:

```
[ whitespace ] ( DNS_name | IP_number ) [ whitespace port ] [ whitespace anything ]
```

B Comments on Mappings

The Mapping key returns an array of the ICMapEntry type.

```
ICMapEntry = record
        total_length: integer;  (* from beginning of record *)
        fixed_length: integer; (* from beginning of record *)
        version: integer;
        file_type: OSType;
        file_creator: OSType;
        post_creator: OSType;
        flags: longint;          (* variable part starts here *)
        extension: Str255;    (* these strings are tightly packed *)
        creator_app_name: Str255;               (* which means, these ones might have an *)
        post_app_name: Str255;                  (* odd address *)
        MIME_type: Str255;
        entry_name: Str255;
    end;
```

Note that this is not literally an array and that you cannot index it directly. You must step through each entry using the total_length field of each record to decide how many bytes to skip.

Programs that wish to do simple extension mapping can just walk the Mapping data looking for a matching extension, as described in the next section. Programs that want to get involved with MIME types etc have a harder job. We suggest that the various MIME people get together to figure out how to fly this data structure.

There is Pascal sample code for parsing and modifying this data structure supplied in the distribution (ICMappings.p).

B.1 Mapping Flags

The bits in the flags entry are defined in ICKeys.[ph]. They include

```
ICmap_binary_bit = 0;        (* file should be transfered in binary as opposed *)
ICmap_binary_mask = $00000001;              (* to text mode *)

ICmap_resource_fork_bit = 1;                (* the resource fork of the file is significant *)
ICmap_resource_fork_mask = $00000002;

ICmap_data_fork_bit = 2;    (* the data fork of the file is significant *)
ICmap_data_fork_mask = $00000004;

ICmap_post_bit = 3;          (* post process using post fields *)
ICmap_post_mask = $00000008;

ICmap_not_incoming_bit = 4;                 (* ignore this mapping for incoming files *)
ICmap_not_incoming_mask = $00000010;

ICmap_not_outgoing_bit = 5;                 (* ignore this mapping for outgoing files *)
ICmap_not_outgoing_mask = $00000020;
```

The first three flags can be used to determine how to upload or download a file, specifically for protocols such as FTP. The post-processing flag is set up by the user and indicates whether applications should post-process this type after a download. The last

two flags can be used to make entries asymmetric, they allow the user to use different settings depending on whether the files are being moved to or from the Macintosh.

B.2 Simple Extension Mappings

Most applications fall into two categories, those that are moving files from a non-Mac to the Mac and those that are moving files in the other direction. Programs that are downloading files to the Mac should use something like the following algorithm:

```
for each entry in the Mapping preference do
    if the extension matches the file's extension then
        if the entry is not marked "not for incoming" then
            use the type and creator for the Mac file
```

If all you need to do is get the creator type for a text file then you should read section C.1.

Programs that are moving files from the Mac to a non-Mac should use something like the following algorithm:

```
rank each entry according to the following table
    1. type, creator and extension match
    2. type and extension match
    3. extension and creator match
    4. extension matches
    5. type and creator match
    6. type matches
    7. anything else
throw away any entries that are marked "not for outgoing"
use the entry with the lowest ranking
```

B.3 Post-Processing

When moving files to the Macintosh the application can choose whether to support post-processing, that is feeding the file on to some other application which hopefully renders it into a more useful form. This is controlled by three fields in the ICMapEntry. The first is the post-processing bit in the flags. If this bit is set then the application should post-process this file (if it supports post-processing). The post-processing application is determined by the post_creator and post_app_name fields. If the post_creator is OSType(0) then no post-processing application has ever been specified by the user.

B.4 Application Specific Data

The ICMapEntry data structure allows for an arbitrary amount of data between the end of the strings and the end of the entry as determined by the total_length field. This information is available for application specific use. The Internet Config application guarantees to preserve this information when it edits the entry.

If you insert data into this area then you must conform to the following convention. Your data must start with a creator type (4 bytes) (usual your application's creator type) followed by the length of the data you've added (4 bytes), which includes the creator and length. You can then add length - 8 bytes of data after that. Your application can

modify and remove data as long as it maintain the consistency of the data structure.

This protocol allows any number of applications to add data without getting in each others way.

Because the minimal length of application specific data is 8 bytes, the Internet Config application will remove any data that is less than 8 bytes long.

C Technical Notes

This appendix contains a number of technical notes about how to use Internet Config correctly.

C.1 Text Files and the Editor Helper

If you are creating a text file and the file does not have an ICMapEntry (either because you don’t have a foreign file name (such as saving an article file in NewsWatcher) or because the file’s extension doesn’t exist in the Mapping preference) then you should use the editor helper as the file’s creator.

If the file has a valid foreign name and that name’s extension is found in the Mapping preference then you should use the entry’s type and creator regardless.

The rationale for this is that we want to provide an easy mechanism for applications such a NewsWatcher to determine the text file creator without having to mess with the Mapping preference. Hopefully we will avoid user confusion because the Mapping preference only affects files with the appropriate extension and the editor helper is used otherwise.

C.2 Binary Stamp Identification with Internet Config

The following is a draft proposal. Please ask the author for a final version before writing code that relies on this.

Version 1 Dec 94
Obsoletes previous BINA proposal.

PURPOSE
Retyping files by means of extensions is often inadequate, especially when working with non-Mac systems that do not require the use of extensions. The BINA entry in the user field of the IC mappings data structure provides a "binary stamp" that can be used to identify files based on their contents rather than their extensions.

IMPLEMENTATION
This information is stored in the user data field of the IC mappings structure with the signature BINA. See the IC programmer’s reference manual for information regarding this mechanism. The format of the entry is as follows:

Bytes	Contents
00-03	'BINA'
04-07	length of entire data structure plus 8 (yy)
08-09	number of signatures
0A-xx	binary signatures stored as PStrings (byte aligned)
xx-yy	(any trailing space is reserved and should be preserved)

The binary signature is the first several bytes shared by every file of the given type. If there are multiple file formats with distinct signatures (which are supported by programs such as Binary Pump) the signatures should be stored sequentially in the BINA entry.

USE
Since many file formats do not include a binary stamp, this mechanism should be used in addition to extension checking, not as a replacement for it. To match by stamp, scan the mappings list and compare each stamp to the file in question. Longer entries take precedence over shorter ones.

Binary stamps should not be applied to files marked as a text type in the main mappings record.

CONTACT
Eric Kidd
eric.kidd@dartmouth.edu

D Credits

The official support address for Internet Config is <internet-config@share.com>. If you find a bug in IC then please forward details to that address. If you want to discuss IC in general then I suggest you host that discussion on the comp.sys.mac.comm newsgroup.

The Internet Configuration System was written by Quinn “The Eskimo!” <quinn@cs.uwa.edu.au> and Peter N Lewis <peter.lewis@info.curtin.edu.au> over a period of too many late nights and weekends. Certain important chunks of code were contributed by Marcus Jager <jager@netcom.com>. Craig Richmond <craig@ecel.uwa.edu.au> provided a lot of help sorting out the default MIME mappings.

We would like to thank all of those on the Internet Config mailing list and all of the developers who we hope will adopt the system.

The Internet Config application uses the PopupCDEF by Ari Halberstadt. This code is Copyright 1994 Ari Halberstadt and is not placed in the public domain. The rest of the Internet Config system is public domain and can be redistributed without restriction.