JP1 RDF File Specification Version 4

Released 4 May 2009

Author: Mike England (mr_d_p_gumby)

http://www.hifi-remote.com/forums/

An RDF file defines the characteristics of a JP1 remote to the various JP1 software tools. The acronym "RDF" stands for Remote Definition File, and is also the file extension used for these files. Without an applicable RDF file, the JP1 tools would not know how to interact with a JP1 remote.

This document is intended to serve several purposes. First, it collects together and documents RDF file requirements from all of the various JP1 programs so that program authors can work from a common set of specifications. Second, it is intended to aid those creating and maintaining RDF files.

NOTE: Throughout this document, references to the JP1 program IR.exe should be assumed to apply equally to the RMIR program currently in development.

RDF file names

RDF files must be named according to a specific set of rules. JP1 programs like IR and RM use parts of the filename for special purposes, and violating the rules may prevent the RDF from being used at all.

The RDF filename must begin with the ASCII characters that represent the remote's signature. This is usually a string of eight characters, but some older remotes have only four-character signatures. In special cases the RDF name may start with less than the full signature, but in general all signature characters should be part of the name. For example, a 15-1994 remote has a signature of **"RSL6RSL0"**, so the filename for its RDF is:

RSL6RSL0 (RS 15-1994 6-in-1 Smart).rdf

The signature part of the filename is terminated with a space character.

- *The signature part of the RDF file name is very significant to the JP1 IR program. When downloading from the remote, IR identifies the signature currently contained in the remote (stored in the EEPROM of JP1 remotes starting at address $002; stored within the flash memory of JP1.x remotes). It then attempts to match the signature part of the RDF file names against the downloaded signature. If a single match is found, the RDF is automatically selected. If multiple matches are found, IR then compares items listed in the [Fixed Data] section of the RDF files. Again, if a single match is found, the RDF is automatically selected. If multiple RDF files match, then the user is prompted to select the desired RDF.*

The next part of the filename is the name of the remote, enclosed within parenthesis. In the example above, the remote's name is **"RS 15-1994 6-in-1 Smart"**. In naming the remote, do not use an underscore (_) character except as directed below, and do not use parenthesis. There should not be any characters between the closing parenthesis and the dot for the file extension, which must always be RDF.

RDF files often apply to more than one remote model. The underscore character is used as a delimiter within the remote name so that individual names may be displayed to the user. When the underscore character is used, it interacts with the dash (-) and space characters. Everything up to the last dash or space preceding the first underscore is used as the beginning of the name for all remotes. Everything after (and including) the first dash or space following the final underscore is used as the end of the name for all remotes. If a dash or space is not found, delimiting takes place at the appropriate parenthesis or underscore as necessary. Characters between the delimited beginning and ending parts of the name are used to construct the individual names.

Some examples will help to make this clearer.

EBV0EBV1 (URC-7550_7552_7560_7562 One for All).rdf

This RDF file applies to four different models, all of which have the same signature (**EBV0EBV1**).
The remote names displayed by RM for this RDF are:

URC-7550 One for All
URC-7552 One for All
URC-7560 One for All
URC-7562 One for All

CYC1 (Navigator URC-44000-B02_B04).rdf

This RDF file applies to two models having the signature **"CYC1"**. The remote names displayed by RM for this RDF are:

Navigator URC-44000-B02
Navigator URC-44000-B04

File format

The general format of an RDF file is basically the same as a Windows INI file. By convention, the layout of the RDF file sections will follow the order shown below. While this specification does not require this exact section order, there are some restrictions on which sections must come after other sections, so it is suggested that this order be used.

```
[General]
[Extender]
[SpecialProtocols]
[Settings]
[Checksums]
[FixedData]
[AutoSet]
[StaticUpgrades]
[DeviceTypes]
[DeviceTypeAliases]
[DeviceButtons]
[Buttons]
[MultiMacros]
[ButtonMaps]
[DigitMaps]
[Protocols]
[SetupCodes]
```

It should be noted that most entries in an RDF file are not arbitrary. With few exceptions, they are intended to communicate to programs exactly how the remote behaves, where and how it stores data, what complement of buttons it has, and so on. For example, if a remote stores a particular set of data at address $100, there is no point in changing the RDF entry to a different value. This would simply cause JP1 programs to store the data in the wrong place.

When adding or modifying a line in the RDF file, be careful not to have any extra trailing space or tab characters at the end of the line as this sometimes causes the line to be ignored. Also, when editing an RDF file in operating systems other than Windows, be sure each line is terminated with both the CR & LF characters.

Unless otherwise noted, throughout the RDF numeric values are considered as decimal values, unless prefixed with a dollar sign ($) to have the number be interpreted as hexadecimal. It is preferred that hexadecimal values be used for addresses, button codes, etc., but this is not mandatory.

Address ranges are specified in the form: **`StartAddr..EndAddr`**.

In general, string arguments and item names may not use any of the following characters, unless the section allows for quoted values:

`(  )  ,  ;  =`

There is no official method for adding comments to an RDF file. You should assume that every line in the RDF file will be processed. In most cases, trying to add a comment line will cause programs to display error messages. An unofficial method is to locate any comment lines at the very beginning of the RDF file, or at the end of the [FixedData] section, and to begin the line with a "#" character. This is not foolproof, as you must still make certain that the line is not accidentally recognized as a valid fixed data entry, and it should be tested with all programs that utilize RDF files. At present, this includes the following programs: IR, RM, RMIR, ECC and EXTINSTALL.

[General] section

In the **[General]** section, all entries are in the form **Item=Value**, with no spaces on either side of the equals sign. All entries are optional and may appear in any order within the **[General]** section, though the **[General]** section itself should be the first section in the RDF file. Most entries can simply be omitted if they do not apply, and the programs will respond by supplying a default value internally. Omitting some entries can severely limit the functionality provided by programs that use the RDF file and/or result in incorrect data being uploaded to a remote.

The defined items are listed below, loosely grouped by their functionality.

- *This group of entries is concerned with naming and identifying the remote for the user. All entries are optional.*

Name

Display name of the remote. No special processing is performed on this text string, so in those cases where an RDF file applies to more than one remote, this entry should name them all.

Identification

A string displayed by programs (when multiple RDFs match a given signature) that should help the user to identify the remote supported by that RDF. The entire entry must be placed on a single line, as the entry is terminated when the first line end is encountered.

OldRemoteID

A number used to identify a specific remote in KM files saved with versions of KM prior to v6.06. If not applicable, this entry should be omitted.

Allows for backward compatibility with older upgrades when importing into RM or other programs. If the RDF applies to more than one remote, an ID number should be entered for each remote, separated by commas.

For example, the URC-881x/801x/601x RDF has the following entry:

OldRemoteID=24,24,24

- *This group of entries defines important address ranges and values that specify how the data storage is arranged in the remote. If one of the address range entries is omitted, it implies that the remote does not have a storage location for data of that type.*

SectionTerminator

Byte value used by the remote to mark the end of each memory section. When omitted, defaults to **$00**. Most more recent remotes use **$FF**, so they would need to have the following entry:

SectionTerminator=$FF

BaseAddr

This entry is only used for newer JP1.x remotes whose data storage area is contained within the main flash memory of the processor. When present, it specifies the physical flash memory address of the first byte of the data storage area. All other addresses and address ranges in the RDF are treated as offsets from this address. When omitted, defaults to **$0000**.

This example specifies that the data storage area begins at hex address **$600** within the main flash memory:

BaseAddr=$0600

EepromSize

Specifies the total size of the remote's data storage area, in bytes (not bits). The EEPROM designation refers to the size of the EEPROM chip used in the original JP1 remotes. For JP1.x remotes, this refers to the total size of the data storage area reserved in the main flash memory.

This example specifies the size as 2048 (800 hex) bytes:

EepromSize=$800

AdvCodeAddr

Specifies the address range of the advanced code section in the remote's data storage area. This is the area where keymoves and macros are stored. If this entry is omitted, it implies that the remote cannot store keymoves and macros.

This example specifies that keymoves and macros are stored in the area starting at hex address **$024** and ending at hex address **$0FF** (a total of 220 bytes):

AdvCodeAddr=$024..$0FF

UpgradeAddr

Specifies the address range of the upgrade section in the remote's data storage area. This is where device upgrades and protocol upgrades are stored. This entry refers to the address of the pointer to the device upgrade list, which is usually **$100** for S3C80-based and 6805-based JP1 remotes and **$104** for 740-based JP1 remotes. Newer JP1.2/JP1.3 remotes generally start the upgrade area around **$400**. If this entry is omitted, it implies that the remote cannot store upgrades.

This example specifies that device and protocol upgrades are stored in the area starting at hex address **$100** and ending at hex address **$3FF** (a total of 768 bytes):

UpgradeAddr=$100..$3FF

LearnedAddr

Specifies the address range of the learned codes section in the remote's data storage area. This is where the remote stores data for keys that have been learned from another remote. When this entry is present, it implies that the remote has learning capability, or at least the ability to play back learned data.

This example specifies that the learned key data is stored in the area starting at hex address **$400** and ending at hex address **$7FD** (a total of 1022 bytes):

LearnedAddr=$400..$7FD

DevUpgradeAddr

Specifies the address range of the device-specific upgrade section in the remote's data storage area. Applies only to those remotes that have device-specific upgrade areas in the data storage area, and should be omitted otherwise. (This excludes checksum bytes, if defined.)

This example specifies that the device-specific upgrade section data is stored in the area starting at hex address **$502** and ending at hex address **$6FF** (a total of 510 bytes):

DevUpgradeAddr=$502..$6FF

TimedMacroAddr

Specifies the address range of the timed macro section in the remote's data storage area. Applies only to those remotes that have a timed macro capability and store this type of macro in a separate area, and should be omitted otherwise. (This address range excludes checksum bytes, if defined.)

This example specifies that the timed macro data is stored in the area starting at hex address **$702** and ending at hex address **$7CE** (a total of 205 bytes):

TimedMacroAddr=$702..$7CE

Note: *The* **TimedMacroAddr** *entry will be ignored if* **MacroCodingType=2**. *See discussion in the* **MacroCodingType** *section of this document.*

Labels

Specifies the location and length of the button text labels section in the remote's data storage area. (These labels display on an LCD screen on the remote.) Applies only to those remotes that support this feature; omitted otherwise. When present, it implies that there is one text label for each device button, and the length parameter specifies the length of all entries.

The Labels entry takes the form:

Labels=UserLblAddr, Length [, [PadByte][, DfltLblAddr]]

where:

UserLblAddr is the start address of the section holding user-defined labels
Length is the length in bytes of each label
PadByte is the byte value used to pad short text in labels
DfltLblAddr is the start address of a second labels section holding default labels

Using the **PadByte** or **DfltLblAddr** parameters in this entry will make the RDF incompatible with programs that have not been updated to comply with RDF specification version 4. An alternate syntax is allowed in this case to allow for a transitional period while various JP1 programs are updated to comply with this version of the RDF specifications.
See Transition From RDF 3 to RDF 4 section of this document for further details

The **PadByte** parameter is optional, and may be omitted. If omitted, the default value is **$20** (ASCII space character).

The **DfltLblAddr** parameter is only used if the remote supports a set of default labels, and should be omitted otherwise.

At present, there are two different approaches to labels that are supported by this entry.

The first approach is as used in the 15-100 remote. The remote displays a 10-character text label on the LCD display for each device button. The labels are populated by default from built-in default values (in English or Spanish), but the user can set each label as desired. The entry for the 15-100 is:

Labels=$200, 10

This specifies that the user text labels are stored starting at hex address **$200**, and that each label is 10 characters long. Text shorter than 10 characters will be padded on the right with space characters. The number of entries is indirectly defined by the number of buttons defined in the **[DeviceButtons]** section.

The second approach is as used in the URC-7780/7781. This remote does not have any device buttons; it has a number of device "slots". The user chooses a device from the LCD display by scrolling through all the slots that have a setup code assigned. The remote will display a 4-character text label on the LCD display for each active device. The entry for the URC-7780 is:

Labels=$38, 4, $FF, $6C

This specifies that the user text labels are stored starting at hex address **$38**, that each label is 4 characters long, and that the default labels are stored at hex address **$6C**. Text shorter than 4 characters in both the user and default labels will be padded on the right with **$FF** bytes. The number of entries is indirectly defined by the number of buttons defined in the [DeviceButtons] section. (Even though this remote has no device buttons, the device "slots" are treated as buttons in that section of the RDF file.)

- *This group of entries is concerned with how the remote works with keymoves (AKA advanced codes or EFCs). All entries are optional, and will default to the values indicated when the entry is omitted.*

AdvCodeFormat

(Related to **AdvCodeBindFormat**) Specifies the format of keymove data stored in the advanced code section. The **AdvCodeFormat** value can be either **HEX** or **EFC**. The default value assumed when this entry is omitted is **HEX**.

Both **AdvCodeFormat** and **AdvCodeBindFormat** combine to reflect how the remote stores command data within a keymove. The four possible setting combinations s are each discussed below.

Note that **AdvCodeBindFormat** also affects other aspects of the keymove structure as discussed under that entry in this document.

- **AdvCodeFormat=HEX** and **AdvCodeBindFormat=NORMAL** (one or both entries may be omitted)
  Example: RS 15-1994 (and all JP1 remotes prior to the URC-6131)

This is the original keymove format used in JP1 remotes. (Omitting both entries defaults to this format.) One or more hex command bytes are included within the keymove structure, and all of them are loaded into the protocol buffer unmodified. While the remote itself cannot create keymoves longer than one byte via EFC entry, JP1 programs are able to do so. Some remotes can create multiple-byte keymoves using the copy-from-another key method.

- **AdvCodeFormat=EFC** and **AdvCodeBindFormat=NORMAL** (**AdvCodeBindFormat** may be omitted)
  Example: URC-6131, Atlas URC-1054 (unextended)

This is the first modification of the keymove format, and is limited to relatively few remotes. It pre-dates the introduction of 5-digit EFC numbers, so only one-byte keymove commands may be created via EFC entry on the remote. There are two basic types of keymoves possible, based on the length of the keymove structure.

Keymoves that contain only one command data byte interpret that byte as a button number (key code), and effectively behave as if that button had been pressed, using the setup code referenced in the keymove. The number of command bytes loaded into the protocol buffer depends only on the protocol executor used by the setup code. Note that in order for anything to happen, the button number must be defined to do something within the referenced setup code.

Keymoves that contain two command data bytes always have the first byte set to zero. The second byte contains a 3-digit EFC number (in hex). The remote pre-processes the second byte before loading it into the protocol buffer by calling a routine to do EFC-to-HEX conversion. This type of keymove is limited to one-byte commands (3-digit EFCs).

Keymoves that contain more than two bytes of command data are treated as if they contained only one byte, i.e., the first byte is considered a button number, and the rest are discarded.

- **AdvCodeFormat=EFC** and **AdvCodeBindFormat=LONG**
  Example: URC-9960B01, Comcast URC-1067 (and some early JP1.2 remotes)

This is the second modification of the keymove format, and is rarely encountered. It is basically an expansion of the first modification to allow for use of 5-digit EFC numbers.

Processing of keymoves is exactly the same as discussed above, with the exception that keymoves with two command bytes may contain a 5-digit EFC (in hex) when the first byte is not zero. In this case, the remote pre-processes both bytes before loading them into the protocol buffer by calling a routine to do EFC-to-HEX conversion. This type of keymove is limited to one-byte or two-byte commands (3-digit or 5-digit EFCs).

There is some variation in how these remotes process a keymove with more than two command bytes. Some remotes respond as if there were only one command byte, while others behave as if there were two command bytes.

- **AdvCodeFormat=HEX** (**AdvCodeFormat** may be omitted) and **AdvCodeBindFormat=LONG**
  Example: URC-8820 (and most other JP1.2 or JP1.3 remotes)

This is the most recent modification of the keymove format, and is typically found in most JP1.2 and JP1.3 remotes. It is nearly identical to the second modification, except that hex commands are stored directly in the keymove command data bytes, rather than EFC values, when the keymove contains two bytes.

Processing of keymoves works the same way. One-byte keymoves are interpreted as button codes as discussed above.

When the remote processes a two-byte keymove, it simply copies both bytes to the protocol buffer unmodified. If the protocol executor only expects one byte, the second byte will be ignored; if it expects two bytes, then both are used. Despite the fact that more than two byte commands could be handled readily, because of the logic UEI chose to use, this type of keymove is limited to one-byte or two-byte commands because only the first two bytes are copied to the protocol buffer even if the keymove structure includes more than two command bytes.

AdvCodeBindFormat

(Related to **AdvCodeFormat**) The **AdvCodeBindFormat** value can be either **NORMAL** (the default) or **LONG**.

Note that this setting will impact other aspects of the keymove structure as discussed under the **AdvCodeFormat** entry in this document.

In some remotes, advanced code entries are encoded such that both the device button and the length of the entry are stored in a single byte. In other remotes, the length of the entry is stored in a byte all by itself. (This causes the overall length of the entry in these remotes to be 1 byte larger than in the other remotes.)

For remotes that use the extra byte, specify the following:

AdvCodeBindFormat=LONG

EFCDigits

The **EFCDigits** value can be either **3** (the default) or **5**.

Some remotes use 3-digit EFCs and some use 5-digit EFCs. This is independent of the **AdvCodeFormat** parameter described above. Remotes supporting 5-digit EFCs should specify the following:

EFCDigits=5

- *This group of entries is concerned with how the remote works with macros. All entries are optional, and will default to the values indicated when the entry is omitted.*

MacroCodingType

Specifies the format of macro data stored in the advanced code section. Set to one of the values discussed below. If this entry is omitted, the default value is **1** with no additional parameters.

The MacroCodingType entry is in the form:

MacroCodingType=Type[, TimedMacros[, TMCountAddr]]

where:

Type is one of the values discussed below.
TimedMacros specifies if timed macros are supported.
TMCountAddr is the address of a byte that stores a count of timed macros.

MacroCodingType=1 is the default, and indicates that the remote stores macros in the original format supported by the JP1 tools. This is the format used by the vast majority of JP1 remotes. A few remotes of this type supported timed macros, but stored them in a separate area (see **TimedMacroAddr**).

MacroCodingType=2 is used with some newer remotes that can store different types of macros in the same area. Macros in these remotes can be encoded to differentiate their nature. At present, this is used to encode timed macros. When this format is designated, the **TimedMacroAddr** entry, if present, will be ignored. (At the present time, the only remotes known to use this format are the URC-7780 & URC-7781.)

Values other than **1** or **2** are not currently supported, and should not be used.

The **TimedMacros** parameter is optional, and is ignored if **MacroCodingType=1**. It should be set to **Y**, **Yes**, **T**, **True** or **1** to indicate that the remote has timed macro capabilities. If the remote does not support timed macros then it should be omitted, or set to **N**, **No**, **F**, **False**, or **0**.

The **TMCountAddr** parameter is only valid if **TimedMacros** is true (and not being ignored), and is otherwise ignored. It should only be present if the remote stores a count of timed macros.

For example, the URC-7780's entry looks like this:

MacroCodingType=2,Y,$2D

This entry specifies that the remote can store timed macros in the same area as other macros, that the remote supports timed macros, and that a count of the number of stored timed macros is maintained in a single byte at hex address **$2D**. If a **TimedMacroAddr** entry is present in the RDF, it is ignored.

- *This group of entries specifies whether a remote supports certain functions or not, and may specify parameters for those functions when required. All entries are optional, and will default as indicated when the entry is omitted.*

KeyMoveSupport

Set to **N**, **No**, **F**, **False**, or **0** for remotes that do not support key moves. If the remote supports keymoves, this entry should be omitted, or set to **Y**, **Yes**, **T**, **True** or **1**.

MacroSupport

Set to **N**, **No**, **F**, **False**, or **0** for remotes that do not support macros. If the remote supports macros, this entry should be omitted, or set to **Y**, **Yes**, **T**, **True** or **1**.

FavKey

Some remotes perform special “Fav/Scan” processing for a “favorite” button (typically labeled FAV or SCAN). This entry specifies parameters for this button, and should only be present if the remote has this feature.

The **FavKey** entry is in the form:

FavKey=KeyCode, DevBtnAddr, MaxEntries, EntrySize[, Segregated]

where:

KeyCode is the button number assigned to the Fav/Scan button
DevBtnAddr is the address of the byte that specifies the device to use
MaxEntries is the maximum number of allowable entries.

The optional argument **Segregated** specifies whether Fav Key macros are stored within the Key Move section or segregated off by itself. If present, **Segregated** should specify the address where Fav Key macros are to be stored (must be a non-zero value).

For example, the 15-1994's entry looks like this:

FavKey=$14, $01A, 15

This entry specifies that the Fav Key has a button number of **$14**, that the device type currently assigned to the Fav Key is stored at hex address **$01A**, and that up to 15 Fav Key macros are supported. Since a **Segregated** option is not present, Fav Key macros will be stored in the same area as other macros.

PowerButtons

Some remotes utilize the device buttons as power buttons. This entry specifies which device buttons perform this dual function, and should only be present if the remote operates this way.

The PowerButtons entry is a list of button numbers separated by commas. The button numbers are those belonging to the device/power buttons, and are listed in ButtonMap order. There must be an entry for each ButtonMap. A zero should be entered if a given map does not have a power button.

The PowerButtons entry is in the form:

PowerButtons=Button#ForMap0, Button#ForMap1, etc.

For example, the Scientific-Atlanta ER1 has three ButtonMaps, and the entry looks like this:

PowerButtons=$3E,$39,$3B

TimedMacroWarning

Applies only to remotes that support timed macros. Set to non-zero for remotes that cannot delete individual timed macro entries, and thus must delete all entries following one that is deleted. This entry should normally be omitted (or set to **0**) unless it is specifically required.

WavUpgrade

Set to **Y**, **Yes**, **T**, **True**, or **1** for remotes that allow data to be uploaded via phone or computer audio. If the remote does not support phone/computer audio upgrades, this entry should be omitted, or set to **N**, **No**, **F**, **False**, or **0**.

- *This group of entries specifies the processor (CPU) used within the remote. All entries are optional, and will default as indicated when the entry is omitted.*

Processor

Specifies the basic processor family, and (indirectly) the JP1 interface type. Only the values listed below are allowed. If omitted, this entry defaults to **S3C80**.

Processor=S3C80 .... used for JP1 remotes with a Samsung S3C80 series processor.
Processor=S3F80 .... used for JP1.3 remotes with a Samsung S3F80 series processor.
Processor=HCS08 .... used for JP1.2 remotes with a Freescale HCS08 series processor.
Processor=SST ...... used for JP1.1 remotes with an SST SST65 series processor.
Processor=6805 ..... used for JP1 remotes with a Motorola 68HC05 series processor.
Processor=740 ...... used for JP1 remotes with a Mitsubishi P8/740 series processor.

Two different Samsung S3C80 processors are used in JP1 remotes. They differ in the address where RAM is located, in execution speed, the location of protocol vectors, and several other details. When an RDF file specifies **Processor=S3C80**, the **RamAddr** entry (see below) is used by programs to differentiate between the two types.

Two different Motorola 68HC05 processors are used in JP1 remotes. They differ considerably in capabilities and data formats. When an RDF file specifies **Processor=6805**, the **ProcessorVersion** entry (see below) is used to differentiate between the two types.

ProcessorVersion

For remotes where **Processor=6805**, set to one of the values listed below. If omitted, the default value is **C9**. This entry is ignored for processors other than the **6805**, and should be omitted in that case.

ProcessorVersion=C9.....specifies a 68HC05C9 processor
ProcessorVersion=RC16/18.....specifies a 68HC05RC16/18 processor

The older **C9** version uses a software-only IR engine, and its capabilities most closely resemble remotes using a **740** processor. The newer **RC16/18** version has IR generation hardware, and its capabilities most closely resemble the **S3C80** based remotes. With the exception of the JP1.1 interface, the **SST** based remotes are nearly identical to the **RC16/18**.

RamAddr

This entry is used **only** when **Processor=S3C80** or **Processor=S3F80**; it should be omitted otherwise. It specifies the location of RAM within the processor, but is used indirectly to determine the type of **S3C80** processor used in the remote. JP1 terms for the two types are:

S3C8 ..... refers to the older version where RAM is located at **$8000**.
S3C8+ ..... refers to the newer version where RAM is located at **$FF00**.

When **Processor=S3C80**, this entry must be one of the following:

RamAddr=$8000 specifies an S3C8 (“old”) type processor. (Default if omitted.)
RamAddr=$FF00 specifies an S3C8+ (“new”) type processor.

When **Processor=S3F80**, this entry must be:

RamAddr=$FF00

- *This group of entries is concerned with specifying how the remote works with device upgrades. All entries are optional, and will default as indicated when the entry is omitted.*

2BytePID

Specifies the format used to store the protocol ID (PID) within a device upgrade. Set to **Y**, **Yes**, **T**, **True**, or **1** for remotes that store the PID as a two-byte value. Omit the entry, or set to **N**, **No**, **F**, **False**, or **0** for remotes that store the PID as a one-byte value, and use a single bit flag in the upgrade header to flag if the PID is greater than **$FF**.

DevCodeOffset

Offset value added to the setup code. If this entry is omitted, it defaults to zero. Certain remotes (for example, the RCU810) have an offset value added to the internal setup code number when the setup codes are entered or displayed on the remote itself.

For example, take the case where the remote uses an offset of 14 for its Setup Codes, and you have set the TV device to setup code 0170 on the remote. Without this entry, when viewed in the IR program, the setup code assigned to the TV button would show as 0156. With **DevCodeOffset=14**, IR will also display the code as 0170, which matches what you would enter on the remote itself.

MaxUpgradeLength

Specifies the maximum size for a device upgrade, in bytes. If omitted, no limit will be placed on the size of a device upgrade.

OEMDevice

Specifies parameters that define an OEM device type. If omitted, it will be assumed that the remote does not have an OEM device type.

The **OEMDevice** entry is in the form:

```
OEMDevice=DeviceNo, DeviceAddr
```

where:

DeviceNo is the device index of the OEM device type
DeviceAddr is the address of the OEM bitmap.

(Bit 0 of the OEM bitmap corresponds to the first device button; bit 1 corresponds to the second device button, etc.)

For example, the Replay RDF has the following entry:

OEMDevice=4, $015

OmitDigitMapByte

Set to **Y**, **Yes**, **T**, **True**, or **1** for remotes that do not contain digit map bytes within their device upgrades. This entry should normally be omitted (or set to **N**, **No**, **F**, **False**, or **0**) unless it is specifically required.

UpgradeBug

Set to non-zero for remotes that must use the device-specific upgrade area for all upgrades that include a protocol upgrade. This entry should normally be omitted (or set to **0**) unless it is specifically required.

- *This group of entries is concerned with specifying how the remote works with protocol upgrades. All entries are optional, and will default as indicated when the entry is omitted.*

MaxProtocolLength

Specifies the maximum size for a protocol upgrade, in bytes. If omitted, no limit will be placed on the size of a protocol upgrade.

ProtocolVectorOffset

Offset value to be added to or subtracted from the subroutine calls a protocol makes to the remote. Signed value; can be preceded by + or – character. Default is zero.

Example:

ProtocolVectorOffset=-$100

ProtocolDataOffset

Offset value to be added to or subtracted from the data references a protocol makes to the remote. Signed value; can be preceded by + or – character. Default is zero.

- *This group of entries is concerned with specifying how the remote works with learned IR commands. All entries are optional, and will default as indicated when the entry is omitted.*

LearnedDevBtnSwapped

Set to **Y**, **Yes**, **T**, **True**, or **1** for remotes that store the **DevBtn** index in the second nibble of the second byte (rather than in the first nibble) of each learned command entry. If this entry is omitted (or set to **N**, **No**, **F**, **False**, or **0**), then the first nibble will be used.

- *This group of entries is used to supply parameters for special programming options that are unique to the JP1 tools. All entries are optional, and will default as indicated when the entry is omitted. (Creating entries in this group will typically require the assistance of a JP1 expert.)*

DevComb

This entry is used by the Device Combiner protocol and Fav/Scan patch to adjust internal values to match the characteristics of the remote. If this entry is omitted, the Device Combiner protocol and Fav/Scan patch will not be available. (Creating this entry will typically require the assistance of a JP1 expert.)

The **DevComb** entry is a series of numbers separated by commas. The numbers may be expressed in decimal or hex. The entry is in the form:

DevComb=[fav][, [comb1][, [comb2][, [comb3][, [comb4][, [comb5][, comb6]]]]]]

where each of the items in the list is optional, and the items are those as identified by the current version of KM (v9.xx). If an item is to be omitted, the item should be left blank. Trailing blank items may be omitted from the list.

For example, the 15-1994's entry looks like this (note comb3 & comb6 are omitted.):

DevComb=$7C, $683, $11FC, , $8AEF, $1259

TimeAddr

Only applicable to remotes with real-time clocks; should be omitted otherwise. Specifies the address where IR will store the current (system) time when uploading to the remote. Must be supported by special reset code in the remote. If omitted, the current time will not be loaded into the remote.

The **TimeAddr** entry is in the form:

TimeAddr=Addr[, Format]

where

Addr is the address where three bytes of time data are to be stored.
Format is optional and determines the format of the time data.

Format, if present, must be set to one of the following values:

Hex binary format (primarily for older remotes)
BCD12 12-hour BCD with AM/PM (for remotes with a 12-hour clock)
BCD24 24-hour BCD (for remotes with a 24-hour clock)

If **Format** is omitted, the default setting will be **Hex**.

Using the **Format** setting in this entry will make the RDF incompatible with programs that have not been updated to comply with RDF specification version 4. An alternate syntax is allowed in this case to allow for a transitional period while various JP1 programs are updated to comply with this version of the RDF specifications.
See Transition From RDF 3 to RDF 4 section of this document for further details

SetupValidation

This entry should only be present when the RDF file contains a **[SetupCodes]** section, and should be omitted otherwise. This entry is in the form:

SetupValidation=Option

where:

Option is one of three possible settings: **OFF**, **WARN** or **ENFORCE**.

SetupValidation=OFF is the default, and is equivalent to omitting the entry entirely. JP1 programs will not expect a **[SetupCodes]** section, but if one is present, it may be used for convenience purposes.

SetupValidation=WARN will cause the program to issue a warning to the user in the event an invalid setup code is present, but will not require the user to change the setup code.

SetupValidation=ENFORCE will cause the program to require that the user correct all invalid setup codes. This is useful when a remote reacts badly to invalid setup codes.

Setup code validation only takes place when a user attempts to upload to the remote or create a WAV file. If a **[SetupCodes]** section is not present in the RDF file, then this entry will be ignored.

RDFVersionAddr

If present, this entry will cause a single byte RDF version number to be written into the remote image at the address specified. The version number will be the highest RDF version number supported by the program. This entry takes precedence over any conflicting entry that might exist in the **[AutoSet]** section.

Example:

RDFVersionAddr=$3FE

ExtenderVersionAddr

This entry is used only in RDF files that support an extender, and should be omitted otherwise. This entry is in the form:

ExtenderVersionAddr=Addr1, Format1[, Addr2]

where

Addr1 specifies the address where the major version data is located.
Format1 defines the format of the major version data.
Addr2 is optional and specifies the address where the minor version data is located.

Format1 determines both how the major version is stored, and how the version information is displayed within a program. It must be set to one of the following values:

Hex the major version is stored as a number
Asc the major version is stored as an ASCII character

When the **Hex** setting is used, programs will display the value of the major version byte in decimal as the version number. If a minor version is defined, it will be separated from the major version number by a dot, and expressed as the decimal value of the minor version byte. If the minor version is less than 10, a leading zero will be inserted. For example, if the major version byte is **$02** and the minor version byte is **$03**, the version will be displayed as "2.03".

When the **Asc** setting is used, programs will display the ASCII character corresponding to the major version byte. If a minor version is defined, it will be expressed as the decimal value of the minor version byte. For example, if the major version byte is **$42** and the minor version byte is **$0E**, the version will be displayed as "B14".

- *This group of entries is specific to remotes that do not have device buttons. Devices are selected by scrolling through a list. These entries are only applicable to such remotes, and should be omitted otherwise.*

SoftDev

Specifies parameters for "soft" device selection.

The SoftDev entry takes the form:

SoftDev=Use[, [EmptyButtons][, [CountAddr][, [SeqAddr]]]]

where:

Use specifies whether or not the remote uses soft device selection.
EmptyButtonSettings determines if device settings are allowed to be empty.
CountAddr is the address of a byte that holds the number of filled slots.
SeqAddr is the address of the data area that maps the LCD display order to the slot positions.

Use is set to **Y**, **Yes**, **T**, **True**, or **1** for remotes that use soft device selection. If the remote does not use soft device selection, the entire **SoftDev** entry should be omitted (or **Use** should be set to **N**, **No**, **F**, **False**, or **0,** and the remaining parameters omitted).

EmptyButtonSettings affects all entries in the **[Settings]** section that specify the **DeviceButtons** section name as a list of choices, and determines if a user will be allowed to make or leave such settings blank ("empty"). If this parameter is set to **Y**, **Yes**, **T**, **True**, or **1**, the user will be allowed to have blank entries; if omitted (or set to **N**, **No**, **F**, **False**, or **0**), then, all entries must be filled with a device button selection.

CountAddr is the address of a byte that holds the number of filled device slots. If omitted, then it is assumed that the remote does not store this data.

SeqAddr is the start address of a series of index bytes that map the LCD display order to the device slot positions. There is one byte per device slot; the length of this sequence is indirectly given by the number of buttons defined in the **[DeviceButtons]** section. If omitted, then it is assumed that the remote does not store this information.

As an example, the entry for the URC-7780 is:

SoftDev=Y,Y,$2C,$A0

This signifies that the remote uses soft device selection, it permits the user to leave empty those settings that reference device buttons, the number of filled device slots is stored at $2C and the mapping of the LCD display order to slot positions starts at $A0.

SoftHT

Specifies how Home Theater is handled by remotes that use soft device selection. This entry is only valid for such remotes, and should be omitted otherwise (if present it will be ignored).

The **SoftHT** entry is in the form:

SoftHT=Use[, DevIndex, DevCode]

where:

Use specifies if the remote allows Home Theater to occupy a device slot.
DevIndex is an index referring to the Home Theater device in the **[DeviceTypes]** section.
DevCode is the setup code that the remote uses internally to identify the Home Theater device.

Use is set to **Y**, **Yes**, **T**, **True**, or **1** if it is valid for the Home Theater device to be assigned to a soft device slot. In this case, the remaining two parameters are required. If set to **N**, **No**, **F**, **False**, or **0,** then Home Theater is not a valid device selection, and the remaining parameters should be omitted.

DevIndex and **DevCode** are both required when **Use** is true. There must be an entry in the **[DeviceTypes]** section that defines the Home Theater device type. The **DevIndex** value here is a (0-based) index into the list of device buttons defined in the **[DeviceTypes]** section, and refers to the entry defining the Home Theater device. For example, if the Home Theater device is defined in the fourth entry within the **[DeviceTypes]** section, then the **DevIndex** value that refers to it is **3**.

Note that there is a subtle difference between the Home Theater device type and all others in that the Home Theater device can only be assigned to one device slot. The user interface on the remote keeps one extra device slot available for the Home Theater device, so to the user it will not appear to occupy a device slot. These restrictions are also enforced by IR.exe.

As an example, the entry for the URC-7780 is:

SoftHT=Y,15,$3F3

This specifies that the remote allows Home Theater to occupy a soft device slot, that Home Theater is the 16th entry in the **[DeviceTypes]** section (0-based **DevIndex** is 15), and that the setup code 1011 (**$3F3** in hex) is the Home Theater device.

- *This group of entries defines characteristics of buttons on the remote. All entries are optional, and will default as indicated when the entry is omitted.*

Shift

Specifies the binary bit(s) that represent the primary shift button in other button key codes, and optionally, the label programs should use when referring to the primary shift button.

The Shift entry is in the form:

Shift=ShiftMask[,ShiftLabel]

where:

ShiftMask specifies the bit or bits used to indicate that a key is in a shifted state.
ShiftLabel determines the text that programs will use to identify the shifted state.

A shifted state is normally achieved by pressing the Setup (Magic, P, etc.) button before pressing the button in question. The **ShiftMask** value is a single byte, and may not be **0**. If there is no **Shift** entry in the RDF, the following will be the default:

Shift=$80, Shift

XShift

Specifies the binary bit(s) that represent the secondary shift button (XShift) in other button key codes, and optionally, the label programs should use when referring to the secondary shift button.

The XShift entry is in the form:

XShift=XShiftMask[,XShiftLabel]

where:

XShiftMask specifies the bit or bits used to indicate that a key is in an XShifted state.
XShiftLabel determines the text that programs will use to identify the XShifted state.

The **XShiftMask** value is a single byte. An XShifted state can be achieved by pressing the button in question after pressing a special Shift key or entering a special shift mode. XShift functionality is disabled if the **XShiftMask** value is 0. If there is no **XShift** entry in the RDF, the following will be the default:

XShift=$00, XShift

DefaultRestriction

Defines the button restriction that should be applied to any key for which a restriction is not explicitly defined, and takes the following form:

DefaultRestriction= ButtonRestriction[{+,-}ButtonRestriction...]

where **ButtonRestriction** can be any one of the constants listed in the **[Buttons]** section of this document (see details on Button Restrictions in the **[Buttons]** section below).

- *This group of entries controls program behavior and/or specifies auxiliary files used by JP1 programs. All entries are optional, and will default as indicated when the entry is omitted.*

ImageMap

Specifies the filename of an image-map file for the remote. If omitted, no image of the remote will be available. If the RDF applies to more than one remote, a filename for each remote may be entered, separated by commas. Multiple filenames should be entered in the same order indicated by the name of the RDF. (It is implied that a JPEG image of the remote will also exist; the name of the JPEG file is specified within the image map file.)

Example:

ImageMap=URC-6820.map,URC-8820.map,URC-10820.map

PauseParams

Specifies how a pause duration is calculated and formatted for Pause special protocols defined in the **[SpecialProtocols]** section.

The PauseParams entry is in the form:

PauseParams=UserName, DataBytes, Multiplier

where:

UserName is the name specified in the [SpecialProtocols] section.
DataBytes specifies the format of the data as discussed below.
Multiplier is the conversion factor from seconds to data value.

The possible entries for **DataBytes** are:

1 — one data byte
2/1 — two data bytes, only the first is used
2/2 — two data bytes, only the second is used
2/B — two data bytes, big-endian (high byte first)
2/L — two data bytes, little-endian (low byte first)

When **DataBytes** is set to **2/1** or **2/2** the byte not used by the protocol is set so that the second byte is always the (hex) EFC of the first byte, regardless of which of them is the actual data value.

Multiplier is expressed as a real number (including decimal point if needed). This value is a conversion factor, and will be multiplied by the number of seconds entered by the user, and rounded to the nearest integer value. This result then formatted per the **DataBytes** entry.

PauseParams entries are optional, but if present then all three parameters are required. There can be multiple **PauseParams** entries provided they have different values for **UserName**. If an entry with a **UserName** is not assigned in the **[SpecialProtocols]** section then the entry is ignored.

If a **UserName** assigned in the **[SpecialProtocols]** section does not have a corresponding **PauseParams** entry, the parameters used by the stand-alone Pause special protocol (as supported by RM and KM) will be assumed. This default is equivalent to:

PauseParams=(name),2/1,10.66 (S3C8 remotes only)
or
PauseParams=(name),2/1,16 (all other remotes, including S3C8+)

As an example, assuming both are properly defined in the **[SpecialProtocols]** section, the **[General]** section could contain the entries:

```
PauseParams=PauseA,2/1,10.66
PauseParams=PauseB,2/B,1000
```

The first entry applies to the stand-alone RM/KM Pause special protocol, while the second is for a Pause special protocol included in an extender. To be activated, all the required elements of a Pause protocol as defined in the **[SpecialProtocols]** section must be present in the remote. Once activated, the user may choose to use it. So, in this example, it would be possible that one or both Pause protocols are activated. Regardless of which one the user chooses, the matching **PauseParams** entry will determine how the timing values are calculated and formatted.

Please refer to the **[SpecialProtocols]** section of this document for further examples and a more detailed discussion of how to use the **PauseParams** entry.

RDFSync

Used to keep RDFs in sync with programs. Should not be omitted. For RDF files compliant with this specification, the entry should be:

RDFSync=4

Programs compatible with this RDF specification will also allow use of RDF files with **RDFSync=3**. A special syntax is allowed in this case to allow for a transitional period while various JP1 programs are updated to comply with this version of the RDF specifications.

See Transition From RDF 3 to RDF 4 section of this document for further details.

StartReadOnlySettings

Specifies the position in the list of entries in the **[Settings]** section of the first setting to be read-only. If omitted then no entries are read-only. Note that "read-only" as used here means that the user cannot edit the value displayed in the Other Settings panel of the IR.exe General page.

As an example, an entry of **StartReadOnlySettings=12** causes the 12th and subsequent entries in the **[Settings]** section to be read-only.

The **InitValue** and **Inverted** parameters of an entry in the **[Settings]** section still determine the value assigned when memory is cleared.

[Extender] section

The [Extender] section is used to define some of the features of an extender, and is only present in RDF files that are used with extenders. The presence of this section (even if empty) will indicate to a program that the RDF is for use with an extender only.

Currently, only one entry type is defined for this section, but more may be added as needed.

DeviceSetup

Defines a setup code, type and name for a device that is normally resident in the extender.

- *NOTE: This entry is currently used only by the Extender Code Calc (ECC) spreadsheet, and will probably be considered obsolete at some point, but should be maintained until the ECC spreadsheet can either be retired, or modified to comply with the [SpecialProtocols] section used by other JP1 programs.*

The DeviceSetup entry is in the form:

DeviceSetup=DevType/SUCode, SPName

where:

DevType is one of the DeviceTypes defined for the remote in the **[DeviceTypes]** section.
SUCode is a 4-digit decimal setup code.
SPName is the name given to the special protocol used by the device.

There can be any number of **DeviceSetup** entries in this section, though each should be unique. The names of special protocols are standardized across all RDF files.

For example, the **[Extender]** section looks like this for the 15-2133 Extender 1:

```
[Extender]
DeviceSetup=VCR/1800, ToadTog
DeviceSetup=TV/1106, LDKP
DeviceSetup=TV/1104, Pause
DeviceSetup=TV/1103, DSM
DeviceSetup=TV/1101, Multiplexer
```

[SpecialProtocols] section

- *NOTE: The Special Protocols tab in the IR program will not be displayed if this section is empty or missing.*

This section contains entries that define special protocols such that a JP1 program is able to provide a friendly user interface for the special protocol. The entries are in one of two forms:

SpecialProtocolName=[DevName/SUCode:][-]PID|XPID[(UserNameList)]

or

SpecialProtocolName=Internal:IDNum[(UserNameList)]

where:

SpecialProtocolName must be one of the recognized standard names of special protocols.
DevName/SUCode is an optional device type and setup code (see below).
PID is the 4-digit hexadecimal Protocol ID number for the special protocol (without a "$").
XPID is a value that includes the PID and data to uniquely identify a specific protocol version.
UserNameList is a comma-separated list of names enclosed within parenthesis.
Internal: is a literal value.
IDNum is a serial ID number that identifies a special protocol built into an extender.

The first syntax form is the one that is in predominant use. The second syntax form is only used in special circumstances (see discussion later in this section). The order in which the entries appear in this section is not important. The minimum required syntax is:

SpecialProtocolName=PID

This form will be discussed first.

Using any other form of this entry will make the RDF incompatible with programs that have not been updated to comply with RDF specification version 4. An alternate syntax is allowed in this case to allow for a transitional period while various JP1 programs are updated to comply with this version of the RDF specifications. See Transition From RDF 3 to RDF 4 section of this document for further details

- *NOTE: The Pause special protocol was not supported with a user-friendly interface by IR program versions prior to version 8.*

SpecialProtocolName must be one of the following standard names in order for programs to provide a special interface for the special protocol:

DSM, **LDKP**, **UDSM**, **ULDKP**, **ToadTog**, **Multiplex**, **Pause**

Use of other names is allowed, but no special user-friendly interface will be provided. This list of special protocol names reflects those that are supported by special programming within some JP1 tools. These standard names have the following meanings:

DSM..... Device-Specific Macros, as implemented within an extender
LDKP..... Long/Double Key Press, as implemented within an extender
UDSM..... Un-extended Device-Specific Macros (stand-alone version)
ULDKP..... Un-extended Long/Double Key Press (stand-alone version)
ToadTog..... TOAD-Toggle, all versions (TOAD = Toggle-Only-Activated-Device)
Multiplex ..... Device Multiplexer, all versions
Pause..... Pause protocol, all versions

PID is the 4-digit hexadecimal Protocol ID number that is assigned to the special protocol. It is expressed here without a leading "$" even though the value is in hexadecimal. (The optional preceding minus sign **[-]** may only be used in RDF version 4 syntax.)

With this information, the JP1 program is prepared to provide a user interface for the special protocol. However, before the user interface is activated, the program verifies that all required elements are present. First, the upgrade protocols are checked to verify that a protocol using the specified PID is present. Next, the device upgrades are checked to see if one can be found that uses the specified protocol. Once these are found to be present, the user interface for that special protocol is enabled. If only the device upgrade is found, then the user interface will still be enabled, but the user will be warned that the protocol needs to be installed.

As an example, here is the **[SpecialProtocols]** section for one of the 15-1994 extenders:

```
[SpecialProtocols]
DSM=01FC
LDKP=01F9
Multiplex=01FE
Pause=01FB
ToadTog=0181
```

- *The syntax extensions discussed below are incompatible with programs not prepared to accept RDF version 4 syntax.*

In some circumstances, an extender may be implemented in such a way that no protocol upgrade is required for a special protocol. In this case, when the program attempts to verify that all elements are installed, it will not find a protocol upgrade, and will therefore prompt the user with a warning message to install the protocol upgrade. To prevent the program from issuing the warning message, precede the PID with a minus sign (dash):

```
DSM=-01FC
```

Some extenders embed the entire processing of a special protocol into the extender itself, and neither the protocol upgrade nor the device upgrade will be present. Since the program will be unable to find either element, it will not enable the user interface for the special protocol. An optional syntax is provided for this case, where the entry will specify the device type and setup code that the extender expects using the **DevName/SUCode** option.

```
SpecialProtocolName=DevName/SUCode:PID
```

DevName must be one of the device names defined in the **[DeviceTypes]** section. If a name in that section is a multiple one with parts separated by a forward slash character, then any one part may be used as the name, but not both together. For example, if the device is defined as **CBL/SAT**, **DevName** may be either **CBL** or **SAT**, but not **CBL/SAT**.

SUCode is the setup code number that the extender uses for the special protocol, expressed in decimal.

As an example, the DSM entry for the URC-6131 extender uses this option to allow the TV device setup code 1103 to be used for the device-specific macro special protocol:

```
DSM=TV/1103:-01FC
```

In the above example, the program will still enable the user interface for the DSM special protocol even though there is no device upgrade present for TV/1103. Note that the optional minus sign preceding the PID is also used here because a protocol upgrade for PID 01FC is not present.

UserNameList is an optional part of the entry, and is used to rename the special functions performed by a special protocol. This is useful when an RDF is set up to allow for use of more than one copy of a specific special protocol, and allows the program to display a different name for each copy. This part of the entry takes the form:

```
[ (Name1[, Name2[, Name3[…]]])]
```

Most special protocols only implement a single function, but some implement two or more functions. The entries in **UserNameList** reflect the number of functions, i.e., a special protocol with a single function would have a single name in the list, whereas one that implements two functions would have two names in the list. Where more than

one function exists, the entries in the list will be used in the order shown in the list of default function names shown below.

DSM special protocol .....DSM
LDKP special protocol .....LKP, DKP
UDSM special protocol .....DSM
ULDKP special protocol .....DSM, LKP, DKP
ToadTog special protocol .....ToadTog
Multiplex special protocol .....Multiplex
Pause special protocol .....Pause

As an example, if a special protocol author wished to implement two different ToadTog protocols and allow them to be active in the remote at the same time, the user would experience difficulty within a JP1 program in identifying which one to use, since both would be named "ToadTog". If the RDF file contained the following entries, then the user would see two different names within the program:

```
ToadTog=0181 (ToadTog1)
ToadTog=0182 (ToadTog2)
```

The above example illustrates the case where the two special protocols are entirely separate. Both the device and protocol upgrades must be present in each case for the special protocol user interface to be activated. If only one of the two has the upgrades present, then it will be the only one activated. The renaming of the functions still takes effect, however, so if the second one is the only one activated, then only "ToadTog2" will be available.

If the special protocol author chose to implement the two ToadTog protocols so that both ToadTogs used only a single protocol upgrade, then the entries could look like this:

```
ToadTog=VCR/1800:0181 (ToadTog1)
ToadTog=VCR/1801:0181 (ToadTog2)
```

In this case, it is necessary to specify the device type & setup code so that the program will be able to associate the names with the correct device upgrades. While this is a perfectly legitimate way to make these entries, it does have the disadvantage that the user interface for both will be activated even if the device upgrades are missing.

The **UserNameList** need not always be used to differentiate between multiple special protocols of the same type. It is valid to use the list if it is desired to simply rename the default function names. For example:

```
LDKP=01F9 (LongKP,DblKP)
```

Another situation where it is useful to define more than one instance of a special protocol occurs with the Pause special protocol. A user might find it useful to use either the built-in version supplied with an extender or the stand-alone version. This is often the case because each provides a different range of pause timings. This is defined in a manner similar to the above examples, but since both use different protocol upgrades with the same PID, only one may be installed at a time.

XPID (rather than **PID**) is used in a situation such as just described to give the JP1 program a way to identify which of two (or more) protocol upgrades is present. This is especially important with the Pause special protocol because the program will need to know how to encode the desired time interval based on which protocol executor is in use.

The **XPID** is an eight-digit hexadecimal number defined as follows:

PPPPXXYY

where

PPPP is the 4-digit PID of the protocol.
XX is a zero-based offset that points to one of the bytes in the protocol executor code.
YY is the value of the byte pointed to by **XX** that must match for the protocol to be recognized.

Going back to the example of the two different Pause protocols, the RDF file would have entries like these:

```
Pause=01FB07A1 (PauseEXT)
Pause=01FB07B3 (PauseRMKM)
```

The first entry uses an **XPID** value of **01FB07A1**, meaning the PID of the protocol upgrade is **$01FB**, and that the 8th byte of the protocol executor code must be **$A1** to match this entry. Similarly, the second entry specifies that the PID of the protocol upgrade is also **$01FB**, and that the 8th byte of the protocol executor code must be **$B3** to match this entry. Assuming that the extender's built-in Pause special protocol matches the first entry, and that it is the one currently installed, the program would activate the Pause interface and present the name "PauseEXT" to the user. If instead the stand-alone version were found to be installed, then the user would see only the name "PauseRMKM".

So far, so good. However, the Pause special protocol has a unique requirement in that the JP1 program still needs knowledge of how to encode the pause interval for each of the different versions. This is done with matching entries made in the **[General]** section of the RDF file, using the **PauseParams** entries. (See **[General]** section of this document for syntax discussion.) In the example above, the following entries would be needed:

```
[General]
PauseParams=PauseEXT,2/B,1000
PauseParams=PauseRMKM,2/1,16
```

Note that the names to the right of the equals sign match the names supplied in the **UserNameList** of the example Pause entries above. If the **UserNameList** were omitted, then the default name (**Pause**) would be used in the **PauseParams** entry. The following entries would be equivalent to the previous example, except the default name would be used instead of "PauseRMKM":

```
[General]
PauseParams=PauseEXT,2/B,1000
PauseParams=Pause,2/1,16

[SpecialProtocols]
Pause=01FB07A1 (PauseEXT)
Pause=01FB07B3
```

Also note that, strictly speaking, the second **PauseParams** entry is not required in either example, since the default values assumed by the JP1 program are correct for the stand-alone Pause protocol.

- *NOTE: The following alternate syntax for special protocols entries only applies to extenders specifically written to use this form. As of this writing, only the URC-7780/7781 extender uses this syntax.*

An extender may be written in such a manner that the special protocol functions are embedded within the extender itself, and do not use a protocol or device upgrade. The required functions are accessed via a specially encoded macro that the extender recognizes, rather than with keymoves as is normally the case. Such internal special protocols are only possible for remotes that have the entry:

```
AdvCodeBindFormat=LONG
```

in the **[General]** section of the RDF file.

These types of special protocol functions are accessed by an ID number that specifies the function. The range of acceptable ID numbers depends upon the type of macro encoding used by the remote, which is defined in the **[General]** section of the RDF file with the **MacroCodingType** entry. Valid ID number ranges are:

0 to **5**, when **MacroCodingType=1**
0 to **4**, when **MacroCodingType=2**

Here is an example of the entries for this type of extender:

```
[SpecialProtocols]
DSM=Internal:0
LDKP=Internal:1
Multiplex=Internal:2
ToadTog=Internal:3
Pause=Internal:4
```

The alternate syntax can be intermixed with the normal syntax as applicable. For example, two different DSM special protocols, one internal (DSM1), and one using a protocol upgrade (DSM2), could be defined:

```
DSM=Internal:0 (DSM1)
DSM=01FC (DSM2)
```

[Settings] section

The [Settings] section contains information about remote-specific settings defined in the remote's data storage area (usually in the addresses below **$100**). The entries are in the form:

Title=ByteAddr.BitNo.NumBits.InitValue.Inverted [(OptionList) | SectionName]

where:

Title is the name of the setting that programs such as IR will display.
ByteAddr is the address of the byte that contains the information.
BitNo is the starting bit position of the data within the byte (0-based).
NumBits is the number of bits that make up the entry.
InitValue is the value to assign to the bits when the memory is cleared.
Inverted determines whether the value must be complemented.
(A non-zero value means that it's inverted.)

The remainder of the line is optional. If specified, then it will be used to populate a combo list that IR will use for user input. The **OptionList**, if specified, is enclosed in parentheses and contains a semicolon-separated list of values. The **SectionName** can be the name of any RDF section, such as **DeviceButtons**.

For example, the VPT device for the 15-1994 remote is defined as:

VPT Device=$018.7.8.0 DeviceButtons

and the VPT status is defined as:

VPT Status=$019.3.1.1 (On;Off)

Note that the VPT status could also be defined as:

VPT Status=$019.3.1.0 (Off;On)

The only functional difference between the two involves the memory clear operation. When the buffer is cleared, the first value in the list will be the one used to set the byte. Also, all the bytes are normally set to **$FF** when the buffer is cleared, but bytes that contain non-inverted status information will be initialized to **$00**.

See also the **StartReadOnlySettings** entry in the **[General]** section.

[Checksums] section

The **[Checksums]** section contains one or more entries formatted in one of the following ways:

+CkAddr:BeginAddr..EndAddr

^CkAddr:BeginAddr..EndAddr

where:

+ is used to specify that the checksum is generated by adding numbers.
^ is used to specify that the checksum is generated by XORing numbers.
CkAddr is the address where the computed checksum byte is stored.
BeginAddr is beginning of the range.
EndAddr is the end of the range.

The complement of the computed checksum is always stored in the byte at **CkAddr+1**.

For example, the entry:

+$000:$002..$03A

means that a checksum computed by adding the values from memory locations **$002** to **$03A** is stored at **$000**, and its complement is stored at **$001**. Spaces are not allowed in checksum specifications.

[FixedData] section

- *Note: The [FixedData] and [AutoSset] sections are quite similar in function. Which should be used in any given circumstance depends upon the functionality desired. [FixedData] will prompt the user before changing any data, while [AutoSet] will change the data without prompting the user. [FixedData] is sometimes used in matching an RDF file with downloaded data, while [AutoSet] is not.*

The **[FixedData]** section contains information about data that is considered to be invariant. If a remote's data storage area contents are found to differ with the values specified in this section, a warning message will be displayed to the user asking if the values should be corrected to match those specified here.

This section is formatted as follows:

```
[Addr=]Value...
```

where:

Addr is the address of the data.
Value is the data that is stored there.

Addr is optional. It will initially default to **0** if not specified, but will increase by 1 for each byte specified until a new address is specified. In this section and all others described below, spacing is irrelevant (except that a blank line indicates the end of the section). This means that multiple addresses can be placed on a line, or you can place one on each line. Commas and semicolons are considered whitespace, so there's no difference between:

```
$005=50,51,52,53
```

and

```
$005 = 50 51; $007 =52
$53
```

How you format these is a matter of personal preference.

In certain circumstances, the fixed data is used to help identify which RDF applies to a remote. For example, many older JP1 remotes have a split signature where the RDF filename only specifies the first four characters of the signature. These remotes store the second four-character part of the signature at another location in the EEPROM. When IR downloads the EEPROM contents of such a remote, and finds more than one RDF file applies (based on a match of the first four characters of the signature), it will then examine the fixed data in the RDF to see if it matches what was downloaded. For this reason, remotes with split signatures should always have the second part of the signature identified in the **[FixedData]** section.

[AutoSet] section

The **[AutoSet]** section contains information about data that is forced to be constant. If a remote's data storage area contents are found to differ with the values specified in this section, the values will be corrected to match those specified here without notifying the user.

This section is formatted in the same manner as the **[FixedData]** section; see above.

- *Note: Unlike data in the [FixedData] section, values specified in this section do not play any part in matching an RDF file with a remote.*

[StaticUpgrades]

- *The **[StaticUpgrades]** section is retired, and no longer in use. Do not include this section in any RDF file.*

The intended functionality of this section was never documented, and seems to have been only partially implemented in IR.exe. It appears that this was an early attempt at providing a means to hide an extender activation upgrade and protocol from the remote so that IR could automatically install it when needed. It is not clear that it was ever tested or used in an RDF file.

The only example of the intended syntax is found as a comment in the IR.exe source code:

```
Device=TV:1800@$692..$694,Protocol=$180[@$695..$6BF]
```

This section is officially retired and should not be present in any RDF file.

[DeviceTypes] section

The **[DeviceTypes]** section contains entries in the form:

DevType[=MapNum[, DevTypeNum]]

where:

DevType is the name of the device type.
MapNum is the number of the button map that is associated with this type of device.
DevTypeNum is the (0-based) number used by the remote that corresponds to the **DevType**.

There should be an entry for each device button listed in the **[DeviceButtons]** section, and these entries must be in the same order. If fewer entries are made in this section than in the **[DeviceButtons]** section, then the last device type entry will apply to all remaining device buttons.

DevType is short description of the type of device(s) that the device button supports. For example, **CBL/SAT** might be used to describe the device type supported by a remote's CBL and SAT device buttons. A **DevType** can appear more than once in this section in order to define the default device type for each device button.

DevTypeNum is the number (device index) used internally by the remote to reference the device type. If a **TypeAddr** is specified in the **[DeviceButtons]** section, then **DevTypeNum** is a 16-bit number where the high-order byte contains the value to be placed at the address specified by **TypeAddr** when a device of that type is chosen. If **TypeAddr** is not specified, then only the low-order byte is significant.

If **MapNum** is omitted then it will default to **-1**, effectively disabling mapping for that device type. If specified, the number must correspond with one of the entries in the **[ButtonMaps]** section.

If **DevTypeNum** is omitted, it will initially default to **0** and increase by **$0101** for each subsequent entry. (Remember, the high-order byte will be ignored if **TypeAddr** is omitted.)

- *Note that entries in this section and entries in the [DeviceButtons] section MUST be in the same order.*
- *Every device type defined in this section must be assigned to a standard device in the [DeviceTypeAliases] section.*

On the left is an example of a typical **[DeviceTypes]** section where **DevTypeNum** is omitted, and on the right is an equivalent section where **DevTypeNum** is expressed explicitly. Note that both sections produce the same result.

```
[DeviceTypes]
Cable   = 0
TV      = 1
VCR/DVD = 2
CD      = 3
Audio   = 3
```

```
[DeviceTypes]
Cable   = 0,0
TV      = 1,1
VCR/DVD = 2,2
CD      = 3,3
Audio   = 3,4
```

The **[DeviceTypes]** section for the URC-7070 repeats device types and uses 16-bit **DevTypeNum** values, and looks like this:

```
[DeviceTypes]
Cable/SAT = 0,$0000
TV        = 0,$0101
Cable/SAT = 0,$0000
VCR/DVD   = 1,$0202
CD/Audio  = 1,$0303
CD/Audio  = 1,$0303
CD/Audio  = 1,$0303
VCR/DVD   = 1,$0202
```

[DeviceTypeAliases] section

The **[DeviceTypeAliases]** section assigns each of the standard JP1 device categories to a particular device type in the remote. Each of the device types specified in the **[DeviceTypes]** section should also be listed in this section. This information is used by RM and other programs to make automatic device type translations across different remotes. For example, if a user creates a PVR upgrade for a 15-1994, and then another user changes the upgrade to a different remote, this data determines which device type will be assigned.

Each of the 16 standard JP1 device categories should be assigned to a device type in this section. If the remote has an OEM mode, the OEM Mode standard category should also be assigned.

The list of standard JP1 device categories is:

Cable, TV, VCR, CD, Tuner, DVD, SAT, Tape, Laserdisc, DAT, Home Auto, Misc Audio, Phono, Video Acc, Amp, PVR, OEM Mode (where applicable)

Entries in this section are of the form:

DeviceTypeName = Category1, Category2, …

where:

DeviceTypeName is one of the device names listed in the **[DeviceTypes]** section.
CategoryN is one of the standard JP1 device categories listed above.

For example, the section for the 15-1994 looks like this:

```
[DeviceTypeAliases]
Cable   = Cable, SAT, Video Acc
TV      = TV
VCR/DVD = VCR, DVD, Tape, Laserdisc, DAT, PVR
CD      = CD, Home Auto, Phono
Audio   = Tuner, Misc Audio, Amp
```

[DeviceButtons] section

There should be an entry for each device button on the remote. The order in which they are listed in this section is not important, but it does affect the order of the entries in the **[DeviceTypes]** section.

- *Note that entries in this section and entries in the [DeviceTypes] section MUST be in the same order.*

Each entry in the **[DeviceButtons]** section is formatted as follows:

```
ButtonName=HiAddr LoAddr[ TypeAddr][, SUCode]
```

HiAddr and **LoAddr** contain the addresses to place the hi-order and lo-order device data (respectively) for each button. (The device data stored in these locations contains both the setup code and device type, and sometimes a bit that is part of the protocol ID.)

The **TypeAddr** is optional and should be specified only if the remote needs to store additional device type information for each button (mostly used on older 6805 and 740 remotes).

SUCode is optional, and specifies a default setup code for the device button. The setup code number is used to assign a default setup code to the device button when a new remote image is created without downloading from the remote first.

An entry using **SUCode** (default setup code) may make the RDF incompatible with programs that have not been updated to comply with RDF specification version 4. An alternate syntax is allowed in this case to allow for a transitional period while various JP1 programs are updated to comply with this version of the RDF specifications. See Transition From RDF 3 to RDF 4 section of this document for further details

- ***Warning:*** *An RDF that contains a ButtonName entry that omits TypeAddr but includes SUCode should not be used with any version of IR.exe prior to version 8.00, or with versions of RemoteMaster prior to v1.89 in RMIR mode. Earlier versions of these programs will interpret the third parameter as TypeAddr, resulting in corruption of the remote memory image. Because the TypeAddr parameter must not be present in most cases, it is strongly suggested that the alternate syntax detailed in the Transition From RDF 3 to RDF 4 section of this document be used when SUCode is provided, until such time as older versions of these programs are no longer in common use.*

The **ButtonName** for a device button cannot contain any of the following characters (and may not be quoted):

```
(     )     ,     ;     =
```

(These characters are not permitted in any of the string values in any of the RDF sections unless quoted strings are specified.)

The **ButtonName** specified in this section MUST be spelled the same way in the **[Buttons]** section.

Some remotes support extra devices for which there is no physical button on the remote. These extra devices can be further divided into two categories:

Devices for which a button code exists even though the remote has no physical button — these should be treated as if the device button actually existed. A **ButtonName** defined here should also be defined in the **[Buttons]** section. It may be helpful to use a button name that helps the user to understand that the button does not actually exist.

Devices for which a button code does not exist — in this case, any suitable **ButtonName** may be used, and the button name will not exist in the **[Buttons]** section.

Here is an example of the **[DeviceButtons]** section for the Radio Shack 15-2116 remote, where the **ButtonNames db-09** through **db-14** are extra devices (no physical button or button code exists for these devices):

```
[DeviceButtons]
TV      = $00A $00B
VCR     = $00C $00D
CBL     = $00E $00F
SAT     = $010 $011
CD      = $012 $013
DVD     = $014 $015
AUX     = $016 $017
AUDIO   = $018 $019
MYSYS   = $01A $01B
db-09   = $01C $01D
db-10   = $01E $01F
db-11   = $020 $021
db-12   = $022 $023
db-13   = $024 $025
db-14   = $026 $027
```

Here is another example, using default setup codes, for the URC-8550:

```
[DeviceButtons]
SAT    = $026 $01E $02E,$1C7
TV     = $027 $01F $02F,$025
VCR    = $028 $020 $030,$048
CD     = $029 $021 $031,$09D
AMP    = $02A $022 $032,$10D
TUNER  = $02B $023 $033,$0BD
AUX1   = $02C $024 $034,$00A
AUX2   = $02D $025 $035,$01D
```

[Buttons] section

The **[Buttons]** section contains a list of the button names used by the remote, and defines the button code ("key code") number and other characteristics of the button.

Entries are in the form:

[GenericName:]ButtonName[=BtnCode[:ButtonRestriction[{+,-}ButtonRestriction...]]]

Button names may be entered in any order, with the order possibly having an effect on where it will show up on the list of buttons within programs. When more than one button is defined on a line, each entry should be separated with a comma.

If a button name or generic name contains a space, or any of the following characters:

( ) , ; =

the name should be enclosed in either single- or (preferred) double-quotes. In general, it is a good idea to use quotes on any name that contains characters other than letters or numbers, for future compatibility in the event the RDF specification is modified to use other characters for special purposes.

- *Note that button names for the device buttons and multi-macro buttons cannot be quoted, and therefore may not use any of the special characters noted above.*

Shifted button codes (>$80) can also be specified in order to allow customized names to be given to shifted keys.

The optional **GenericName** parameter is used by RM to allow for easier conversion between remotes and to reliably import KM upgrades, but is not yet used by IR. A generic name need not be entered if the button name already matches one of the standard generic names, shown in the list below. Case is not significant, and only the names in the list below are valid. Other text values for the **GenericName** are invalid, and should not be used as they may cause unpredictable results.

An example will help to illustrate the importance of supplying the **GenericName**. In the case where a user obtains an upgrade that was created for the 15-1994, and wants to change the upgrade for use with a URC-8820, RM will have to re-map the buttons from the 15-1994 to the URC-8820. The 15-1994 has a button named DISPLAY, while the URC-8820 has a button named INFO. If the URC-8820 RDF file defines the INFO button as:

display:info=$26

Standard Generic Button Names					
(Where appropriate, the generic names below are shown quoted)					
"vol up"	"up arrow"	pause	"next track"	rear	setup
"vol down"	"down arrow"	rewind	"prev track"	phantom1	light
mute	"left arrow"	"fast fwd"	"shift-left"	phantom2	theater
"channel up"	"right arrow"	stop	"shift-right"	phantom3	macro1
"channel down"	select	record	"pip freeze"	phantom4	macro2
power	sleep	exit	slow	phantom5	macro3
enter	"pip on/off"	surround	eject	phantom6	macro4
"tv/vcr"	display	input	"slow+"	phantom7	learn1
"prev ch"	"pip swap"	"+100"	"slow-"	phantom8	learn2
menu	"pip move"	"fav/scan"	x2	phantom9	learn3
guide	play	"device button"	center	phantom10	learn4

When creating or updating an RDF file for a remote already supported by KM, care should be taken to assign the same set of standard generic names used by KM so that RM can properly import KM upgrades.

If no **BtnCode** is specified then the button code number will be one higher than that of the previously defined button, and the button restriction will be the same as that of the previously defined button. (For the first entry, the **BtnCode** will default to **1** and the button restriction will default to **DefaultRestriction** as defined in the **[General]** section.) If a **BtnCode** is defined but the button restriction is omitted, then the restriction will be

DefaultRestriction. (If no **DefaultRestriction** is specified in the **[General]** section, then the button will be unrestricted.)

A button restriction applies to the button as it is defined. (See the Button Restrictions listed at the end of this section for a list of the button restriction constants.) For example, consider a button defined as:

Bar=$83

Even though this button's **BtnCode** has its shift bit set (the high-order bit), you would restrict it from having a key move assigned to it by adding the restriction **MoveBind** (not **ShiftMoveBind**).

Note that the Shift and XShift versions of the button restriction constants restrict the ability of the user to associate shifted states with the button in question. Depending on the button's definition, however, a shifted restriction may be superfluous. Given the example in the above paragraph, **ShiftMoveBind** is assumed because the button already has its shift bit set. As such, shifting the button would have no effect. (**ShiftBind+ShiftData** is assumed for buttons whose shift bits are already set in their **BtnCode**.)

It is legal to define 2 buttons with different **BtnCode** values that can resolve to the same **BtnCode**. For example, the following is legal:

Foo=$03, Bar=$83

even though SHIFT-Foo has the same **BtnCode** as Bar. It is also legal to have 2 buttons with the exact same **BtnCode**, but in that case they should be restricted so they will not show up on the same list. For example:

Foo=$03:AllMoveBind, Bar=$03:All-MoveBind

would be legal, but:

Foo=$03:MoveBind, Bar=$03:All-MoveBind

would be illegal because both Foo and Bar would still be listed in the KeyMove binding ComboBox. (In this case, if IR saw a KeyMove bound to **BtnCode $03**, it wouldn't know which name to attach to it.)

Just as it is legal to have two different button names with the same **BtnCode** (as long as they are properly restricted), it is also legal to have the same button name associated with two different **BtnCode** values. Again, this requires that the buttons be restricted such that they never appear on the same list, but there are several remotes (mostly older P8-style) that use different **BtnCode** values for the same button based on its use. In other words, the following is legal:

Foo=$83:All-MacroData, Foo=$C3:All-MacroBind

In this case, Foo would show up in the macro binding ComboBox and in the macro key list, but selecting Foo from the ComboBox would bind a macro to $C3 and putting Foo into the macro would use $83.

The **DefaultRestriction** entry in the **[General]** section specifies the button restriction that should be applied to any key for which a restriction is not explicitly defined (see **[General]** section above).

Button Restriction Constants	
MoveBind	Key moves can't be bound to this button in its unshifted state
ShiftMoveBind	Key moves can't be bound to this button in its shifted state
XShiftMoveBind	Key moves can't be bound to this button in its xshifted state
MacroBind	Macros can't be bound to this button in its unshifted state
ShiftMacroBind	Macros can't be bound to this button in its shifted state
XShiftMacroBind	Macros can't be bound to this button in its xshifted state
LearnBind	Learned commands can't be bound to this button in its unshifted state
ShiftLearnBind	Learned commands can't be bound to this button in its shifted state
XShiftLearnBind	Learned commands can't be bound to this button in its xshifted state
MoveData	Key moves can't contain this button in its unshifted state
ShiftMoveData	Key moves can't contain this button in its shifted state
XShiftMoveData	Key moves can't contain this button in its xshifted state
MacroData	Macros can't contain this button in its unshifted state
ShiftMacroData	Macros can't contain this button in its shifted state
XShiftMacroData	Macros can't contain this button in its xshifted state
TMacroData	Timed macros can't contain this button in its unshifted state
ShiftTMacroData	Timed macros can't contain this button in its shifted state
XShiftTMacroData	Timed macros can't contain this button in its xshifted state
FavData	Fav lists can't contain this button in its unshifted state
ShiftFavData	Fav lists can't contain this button in its shifted state
XShiftFavData	Fav lists can't contain this button in its xshifted state
AllMoveBind	Key moves can't be bound to this button regardless of shift state
AllMacroBind	Macros can't be bound to this button regardless of shift state
AllLearnBind	Learned commands can't be bound to this button regardless of shift state
AllMoveData	Key moves can't contain this button regardless of shift state
AllMacroData	Macros can't contain this button regardless of shift state
AllTMacroData	Timed macros can't contain this button regardless of shift state
AllFavData	Fav lists can't contain this button regardless of shift state
Bind	Nothing can be bound to the button in its unshifted state
ShiftBind	Nothing can be bound to the button in its shifted state
XShiftBind	Nothing can be bound to the button in its xshifted state
Data	The button can't be contained in anything in its unshifted state
ShiftData	The button can't be contained in anything in its shifted state
XShiftData	The button can't be contained in anything in its xshifted state
Shift	Nothing can be bound to the button, nor can it be contained in anything in its shifted state
XShift	Nothing can be bound to the button, nor can it be contained in anything in its xshifted state
AllBind	Nothing can be bound to the button regardless of shift state
AllData	Nothing can contain this button regardless of shift state
All	The button cannot be used anywhere, in any shift state

A button restriction defined as:

MoveBind+XShiftMoveBind

will prevent a key move from being assigned to a button in its unshifted or xshifted states, but it will allow all other uses of the button (including being able to assign a key move to it in its shifted state). A restriction of:

All-MoveBind

will allow the button to be used only to bind key moves to. (In this case, the minus operator negates the specified restriction.)

[MultiMacros] section

The **[MultiMacros]** section contains a list of buttons that can contain multiple macros, one of which fires (in order) each time the button is pressed. It contains entries in the form:

ButtonName=Address1 [, Address2]

where:
ButtonName is the name of the button that can contain multiple macro entries. This name must match the name specified in the **[Buttons]** section, and may not be a quoted name.

If only **Address1** is specified, then the byte at that address will contain the number of macros in the high-order nibble, and the number of the next macro to fire in the low-order nibble. (Both values are encoded.) If **Address1** and **Address2** are both specified, then the first address will contain the number of macros assigned, and the second address will contain the number of the next of the next macro to fire. (Again, both are encoded.)

Note that this section must be defined AFTER the **[Buttons]** section, and should be omitted entirely if the remote does not support MultiMacros.

[ButtonMaps] section

The **[ButtonMaps]** section reflects the remote's internal button maps, and is in the form:

MapNum=BtnCodeList

where:
MapNum is a button map number referred to in the **[DeviceTypes]** section.
BtnCodeList is a list of button code numbers, some of which can be enclosed in parentheses to indicate that they are mapped to the same bit. Each entry (button codes surrounded by parentheses are considered a single entry) corresponds to one mapping bit.

Each remote contains an internal set of button maps. A button map is a list of button codes for buttons that may be included within a device upgrade. Button code numbers in this section may be expressed in hex or decimal, though hex is preferred. (Creating these entries will typically require the help of a JP1 expert.)

Here is an example of a **MapNum** entry:

```
0 = ($1F, $15, $16, $17, $19, $1A, $1B, $1C, $1D, $1E), ($04, $05, $08), ($06, $07)
    $03, $14, $13, $36, $37, $31, $32, $33, $34, $35, $38, $2A, $2B, $18, $20, $0B,
    $0C, $0D, $10, $0F, $0E, $27, $26, $30, $2D, $2E, $2F, $25, $28, $39
```

This entry is for map number 0. The first group of button codes in parenthesis is for the numeric keys (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), the second group is for the volume keys (volume up, volume down & mute), and the third group is for the channel select keys (channel up & channel down). These three groups each use one bit in the upgrade bitmap bytes. The remainder of the button codes listed each use one bit in the upgrade bitmap bytes.

[DigitMaps] section

Almost all JP1 remotes contain an internal table of digit maps. A digit map is a list of commonly used hex function codes for the digit buttons. Using these digit maps allows device upgrades to be smaller since the functions for all ten digit keys can be defined with a single byte of data.

This section contains a space-delimited list of pointers to a table of digit maps internal to the JP1 programs. The pointers are usually expressed as decimal values, and the list may span more than one line. Here is an example of a **[DigitMaps]** section:

```
[DigitMaps]
072 053 038 047 094 092 003 001 090 046 077 086 021 011 087 075
059 058 064 391 392 187 188 146 083 239 240 126 034 397 030 291
063 334 069 093 227 228 219 220 294 205
```

Creating these entries will typically require the help of a JP1 expert, and is beyond the scope of this document.

[Protocols] section

The **[Protocols]** section lists the PID (Protocol ID) number of all the protocol executors resident in the remote. JP1 programs use this information to determine if a protocol upgrade is required, and also to determine the nature of the fixed and variable data for a device upgrade. Both the PID and the variant play a part in this determination.

The entries are always in hex and preferably four digits, and are separated by commas. The list may span more than one line. The $ is not to be used to indicate hex numbers in this section, since hex is mandated. Each protocol listed may optionally specify a protocol variant after the protocol PID number. Normally the entries are listed in PID number order, but that is not mandatory.

The entries are of the form:

PID#[:VariantText]

For example, a **[Protocols]** section might look like this:

```
[Protocols]
0000, 0002:5, 0006, 0007, 000A, 000C, 000D, 000E, 0011, 0013,
0014, 0015, 0018, 001A, 001B, 001C, 001D, 001E, 001F:8, 0021,
0022, 0027:new, 0029, 002A, 002D, 002F, 0032, 0034, 0039, 003A,
003D, 003F, 0045, 0046, 0056, 0058, 005A, 005B, 005C, 005D,
005E:2, 005F, 0060, 0061, 0065:2, 0067, 0068, 006A, 006E, 0073,
0078, 007E:3, 0083, 0087, 008D, 0092:3 0093, 0098, 009C, 009E,
00A4, 00AF, 00B6, 00BE, 00C4, 00C9, 00CA, 00CD:2, 00CE, 00D0,
00D7, 00DB, 00DE, 00E2, 00E3, 00E7, 00E8, 00F0, 00F2, 00F5,
00F8:3, 00FC, 0103, 0109, 010C, 010E, 010F, 0111, 0114, 0115,
0117, 0118, 0119, 011A, 011B, 0125, 012A:2, 0161, 016D, 0174,
017E, 0182, 0184:2, 0186, 018B, 0190, 0194, 01A4
```

In this example, the protocol executor for PID **001F** is present in the remote, and is variant type **8**. Similarly, the executor for PID **0027** is present and is variant type **new**.

By convention, omitting the variant implies that the remote contains the oldest known variant. The set of variants that exist for each protocol executor is defined by the JP1 experts, and is constantly evolving. The variant text in an RDF file must match the variant name present in the **protocols.ini** file (used by the RM program).

Creating these entries will typically require the help of a JP1 expert, and is beyond the scope of this document.

[SetupCodes] section

The **[SetupCodes]** section is optional in most cases (see below), and lists the setup code numbers of all the setup codes resident in the remote. JP1 programs may optionally use this information to enforce validation of setup codes entered by the user, and also to provide a convenient means for the user to select resident setup codes.

The entries are of the form:

DevTypeNum=SetupCodeList

where:
DevTypeNum is one of the device type numbers defined in the **[DeviceTypes]** section.
SetupCodeList is a list of setup code numbers that are resident in the remote for the device type.

DevTypeNum refers to one of the device types defined in the **[DeviceTypes]** section. There is normally an entry in the **[SetupCodes]** section for each device type number; however, a device type number with no setup codes should be omitted.

The **SetupCodeList** entries are always in decimal (hex is not allowed), and are separated by commas. Leading zeroes are allowed, but not required; intervening spaces are allowed but not needed. The list may span more than one line. Normally the entries are listed in numeric order, but that is not mandatory.

Note that only resident setup codes are allowed in this list, and the values of the setup codes are those used internally by the remote, i.e., the value after the **DevCodeOffset** setting in the **[General]** section has been applied. For example, if **DevCodeOffset=17**, and the user manual for the remote documents setup code 1017, the internal setup code value is **1000** (1017 – 17 = 1000).

If the **[SetupCodes]** section is present in an RDF file, JP1 programs may use it to construct a list of setup codes to present to the user as a convenience. In some cases, it is necessary to prevent a user from entering invalid setup codes because a remote reacts badly to invalid codes. In those cases, validation of entered setup codes may be enforced by inclusion of the **[SetupCodes]** section and making the appropriate **SetupValidation** entry in the **[General]** section.

As an example, a **[SetupCodes]** section might look like this:

```
[SetupCodes}
0 = 0,3,8,9,12,14,17,52,53,74,76,82,99,107,113,143,144,152,159,180,209,210,
    212,216,237,238,247,269,273,276,279,280,295,305,317,392,476,477,525,533,
    566,637,639,701,722,724,749,772,775,790,810,819,855,856,869,877,883,899,
    1005,1006,1010,1076,1106,1109,1120,1126,1142,1170,1190,1219,1226,1246,
    1250,1254,1267,1268,1270,1272,1276,1285,1309,1324,1329,1344,1363,1364,
    1365,1376,1383,1392,1393,1403,1442,1443,1444,1490,1505,1639,1640,1749,
    1775,1856,1877
1 = 0,1,3,16,17,18,19,20,21,24,27,30,32,38,39,46,47,51,52,53,54,55,56,60,80,
    88,90,92,93,96,111,135,136,145,146,150,151,154,156,157,159,165,166,171,
    177,178,179,180,185,186,187,217,236,250,280,282,360,381,386,442,451,463,
    466,491,497,511,587,603,623,628,632,638,650,672,679,683,688,689,690,700,
    702,703,704,706,707,717,720,748,751,761,765,766,767,768,769,774,783,799,
    802,809,812,813,814,815,817,824,832,833,834,836,839,840,842,843,845,847,
    849,851,853,854,855,856,857,864,865,866,868,870,871,872,875,879,1047,
    1060,1100,1145,1147,1154,1156,1178,1247,1250,1253,1254,1256,1347,1356,
    1410,1447,1454,1547,1656,1661,1704,1755,1756,1903,1904,1905,1906,1907,
    1909,1911,1913,1914,1917,1918,1919,1920,1922,1923,1924,1925,1926,1927,
    1928,1929,1931,1933,1935,1936,1937,1938,1939,1940,1941,1943,1944,1945,
    1946,1947,1948,1950,1951,1952,1953,1958
2 = 0,2,8,20,32,33,35,37,38,39,41,42,43,45,46,47,48,60,61,67,72,81,89,104,
    105,106,121,124,149,159,162,175,184,202,208,209,222,225,240,271,278,307,
```

```
|          432,479,490,503,511,521,522,525,526,533,534,539,545,558,561,563,571,573,
           582,591,593,614,616,618,623,627,630,632,633,634,636,641,646,651,655,662,
           664,670,672,674,675,682,692,695,698,699,702,705,711,715,717,719,720,721,
           736,737,739,744,752,755,760,761,769,770,778,782,783,784,785,792,794,796,
           797,798,799,800,801,803,804,807,809,815,816,820,821,822,823,826,830,833,
           839,845,848,854,860,864,867,868,869,872,873,876,880,885,899,1001,1003,
           1004,1008,1014,1016,1020,1022,1023,1024,1027,1031,1032,1033,1035,1037,
           1041,1043,1044,1045,1048,1049,1051,1056,1057,1058,1060,1061,1062,1064,
           1068,1071,1072,1073,1074,1075,1077,1078,1081,1082,1085,1086,1087,1089,
           1094,1099,1100,1105,1107,1117,1121,1122,1145,1162,1181,1232,1237,1262,
           1278,1362,1462,1479,1490,1503,1521,1533,1593,1762,1781,1901,1902,1903,
           1904,1906,1907,1908,1909,1910,1912,1913,1914,1915,1916,1917,1918,1919,
           1923,1924,1925,1926,1931,1932,1934,1937,1938,1939,1940,1943,1944,1945,
           1946,1949,1950,1951,1954,1956,1957,1962,1964,1965,1967,1970,1972,1974,
           1975,1976,1977,1979,1980,1981,1984,1985,1986,1988,1989,1990,1992,1995,
           1996,1998,2000,2001,2002,2003,2006,2007,2010,2016,2017,2020
     3 = 4,8,10,11,13,14,27,31,39,42,52,54,62,73,74,76,77,78,80,85,97,106,110,121,
           128,133,135,143,150,158,159,160,163,165,168,176,177,181,186,189,193,195,
           208,211,219,220,224,235,239,244,246,251,264,269,273,300,308,309,313,314,
           320,321,322,331,346,347,354,356,360,367,380,382,391,395,404,405,406,424,
           459,460,463,474,491,502,504,518,520,521,530,531,569,576,577,582,606,616,
           630,639,647,670,674,689,738,765,771,801,815,842,857,891,892,1023,1042,
           1051,1052,1058,1074,1076,1077,1100,1120,1136,1142,1154,1176,1181,1189,
           1229,1243,1250,1251,1253,1254,1255,1257,1258,1263,1266,1267,1269,1273,
           1286,1288,1289,1293,1295,1298,1306,1308,1313,1316,1331,1349,1352,1360,
           1366,1370,1374,1375,1383,1384,1388,1389,1390,1393,1405,1406,1408,1412,
           1420,1423,1426,1428,1430,1437,1441,1442,1461,1462,1464,1469,1483,1485,
           1491,1495,1497,1500,1508,1511,1512,1514,1517,1518,1519,1528,1530,1531,
           1532,1548,1555,1556,1558,1561,1563,1569,1570,1605,1609,1641,1653,1658,
           1758,1759,1763,1764,1798,1801,1858,1869
```

Creating these entries will typically require the help of a JP1 expert, and is beyond the scope of this document.

Transition From RDF 3 to RDF 4

The RDF version 4 specifications introduce a number of new items that, if used, will make an RDF file incompatible with existing versions of various JP1 programs. Depending on the circumstances, these programs may respond with error messages, crash, or by corrupting a remote memory image.

Several new items have been added to the **[General]** section that will not function with older program versions. In general, older programs should safely ignore these entries, but it is possible that they could generate errors if, for example, they contain syntax errors. The new items are:

ExtenderVersionAddr
MacroCodingType
PauseParams
RDFVersionAddr
SetupValidation
SoftDev
SoftHT
StartReadOnlySettings

Some existing items in the **[General]** section have had their syntax modified. Use of the newer syntax for these entries will create compatibility problems with older programs. The items with modified syntax are:

Labels
TimeAddr

Entries in the **[SpecialProtocols]** and **[DeviceButtons]** sections have new syntax options that can cause problems with older programs.

A new **[SetupCodes]** section has been added. Older programs should safely ignore entries in this section, but care should be taken to make sure there are no syntax errors.

Because of the volunteer nature of the programming efforts within the JP1 group, it may take some time before all the JP1 programs and tools that utilize RDF files can be updated to be compliant with this version of the RDF specifications. During this period, an alternate syntax is provided that will enable older versions to operate correctly, while at the same time allowing newer versions to take advantage of the additional functionality.

Notes within this document that display this symbol
indicate items that can create compatibility problems.

The alternate syntax detailed here may only be used in an RDF file that is otherwise compliant with the RDF version 3 specifications. Therefore, the first requirement for using the alternate syntax is that the RDF file contain this entry in the **[General]** section:

[General]
RDFSync=3

The alternate syntax may NOT be used in an RDF file where **RDFSync=4**.

Programs that are compliant with the RDF version 4 specifications are required to allow use of version 3 RDF files.

Each of the potential problem areas is discussed below, and an alternate syntax provided that can be used to maintain compatibility with version 3 RDF files. Refer to the relevant sections of this document for a more detailed discussion of these items.

Labels ([General] section entry)

The **Labels** entry is in the form:

```
Labels=UserLblAddr, Length [, [PadByte][, DfltLblAddr]]
```

Compatibility is affected when the options parameters **PadByte** and/or **DfltLblAddr** are used. If these parameters are required, use the following entry form in a version 3 RDF file:

```
Labels+=UserLblAddr, Length [, [PadByte][, DfltLblAddr]]
```

This entry will only be recognized by programs compliant with version 4 RDF files, and will be ignored otherwise.

It is not possible to create an entry that will fully enable this functionality in older program versions. Partial compatibility can be accomplished (assuming the remote requirements are satisfied) by adding a second entry using the older syntax:

```
Labels+=UserLblAddr, Length [, [PadByte][, DfltLblAddr]]
Labels=UserLblAddr, Length
```

In this case, note that the **Labels+** entry must be first.

TimeAddr ([General] section entry)

The **TimeAddr** entry is in the form:

```
TimeAddr=Addr[, Format]
```

Compatibility is affected only when the optional **Format** parameter is used, and the **Format** value is either **BCD12** or **BCD24**. In this case, use the following entry form in a version 3 RDF file:

```
TimeAddr+=Addr, Format
```

This entry will only be recognized by programs compliant with version 4 RDF files, and will be ignored otherwise. It is not possible to create an entry that will enable this function in older program versions because they can only supply the time value in the default **Hex** format.

If the **Format** value is **Hex**, the entry can be made compatible by simply omitting the **Format** parameter, as **Hex** is the default value.

[SpecialProtocols] section

Programs compliant with version 3 RDF files are only prepared to accept entries in the **[SpecialProtocols]** section that use this more restrictive form:

```
SpecialProtocolName=PID
```

In addition, the Pause special protocol is allowed, but not supported with a user-friendly interface.

A compatible RDF file can be constructed by providing two sections for special protocols, as shown here:

[SpecialProtocols+]
Entries compatible with version 4 RDF files are in this section.

[SpecialProtocols]
Entries compatible with version 3 RDF files are in this section.

The "+" section must be placed before the "non-+" section in the RDF file.

As an example, a compatible RDF file for the URC-6131 extender could be constructed as follows:

```
[SpecialProtocols+]
DSM=TV/1103:-01FC
LDKP=01F9 (LongKP,DblKP)
Multiplex=01FE
Pause=01FB058D (PauseEXT)
Pause=01FB03E4 (PauseRMKM)
ToadTog=0181

[SpecialProtocols]
DSM=01FC
LDKP=01F9
Multiplex=01FE
ToadTog=0181
```

In some cases, where it may not be necessary to support any special protocols that use the older syntax, the second ("non-+") section may be omitted entirely.

[DeviceButtons] section

Each entry in the **[DeviceButtons]** section is formatted as follows:
ButtonName=HiAddr LoAddr[TypeAddr][, SUCode]

Compatibility is affected only when the optional **SUCode** parameter is used. Specifically, the problem occurs when **SUCode** is present and **TypeAddr** is omitted (as is the case for most remotes). In this case, a compatible RDF file may be constructed by providing two section as shown here:

[DeviceButtons+]
Entries compatible with version 4 RDF files are in this section.

[DeviceButtons]
Entries compatible with version 3 RDF files are in this section.

The "+" section must be placed before the "non-+" section in the RDF file.

As an example, a compatible RDF file for the 15-1994 could be constructed as follows:

```
[DeviceButtons+]
CBL/SAT = $00A $00B, 0003
TV      = $00C $00D, 0047
VCR     = $00E $00F, 0060
CD      = $010 $011, 0032
AUX1    = $012 $013, 0013
AUX2    = $014 $015, 0080
P&P     = $016 $017, 0167

[DeviceButtons]
CBL/SAT = $00A $00B
TV      = $00C $00D
VCR     = $00E $00F
CD      = $010 $011
AUX1    = $012 $013
AUX2    = $014 $015
P&P     = $016 $017
```

Revision History

February 2, 2004	ME	Initial rough draft combining old RDF 3 Specification document and Addendum.
November 25, 2004	ME	Revisions to bring up to date with current versions of RM & IR (v5.xx).
August 28, 2005	ME	Further modifications & revisions to initial draft. Bring up to date with features of IR v6.01.
October 13, 2007	ME	Format revisions & corrections. Submitted for final review as Rev 004.
March 16, 2009	ME	Update to include new functionality included in IR v8.00. Initial public release.
April 18, 2009	ME	Further updates for IR v8.00.
May 4, 2009	ME	Minor revisions for IR v8.00